An Abstraction-Refinement Framework for Multi-Agent Systems

Thomas Ball
Microsoft Research*

Orna Kupferman
Hebrew University†

Abstract

Abstraction is a key technique for reasoning about systems with very large or even infinite state spaces. When a system is composed of reactive components, the interaction between the components is modeled by a multi-player game and verification corresponds to finding winners in the game. We describe an abstraction-refinement framework for multi-player games, with respect to specifications in the alternating μ-calculus (AMC). Our framework is based on abstract alternating transition systems (AATSs). Each agent in an AATS has transitions that over-approximate its power and transitions that under-approximate its power. We define the framework, define a 3-valued semantics for AMC formulas in an AATS, study the model-checking problem, define an abstraction preorder between AATSs, suggest a refinement procedure (in case model checking returns an indefinite answer), and study the completeness of the framework. For the case of predicate abstraction, we show how reasoning can be automated with a theorem prover.

Abstractions of multi-player games have been studied in the past. Our main contribution with respect to earlier work is that we study general (rather than only turn-based) ATSs, we add a refinement procedure on top of the model checking procedure, and our abstraction preorder is parameterized by a set of agents.

1 Introduction

We consider how to verify systems composed from *reactive* components. Each component is an *open system*, which interacts with its environment and whose behavior depends on the state of the system as well as the behavior of the environment. Modeling languages for open systems, such as CSP [16] and I/O Automata [22], distinguish between *internal* nondeterminism — choices made by the system, and *external* nondeterminism — choices made by the environment. Such a distinction exists naturally also in software, where processes have internal and external variables.

A game-theoretic property arises naturally: can the system resolve its internal choices so that the satisfaction of a property is guaranteed no matter how the environment resolves the external choices? For example, if P_1 and P_2 are processes that assign values to the variables x and y, we wish to verify properties like "it is possible for P_1 to make x always bigger than y, no matter how P_2 behaves" or "it is possible for P_1 to eventually prevent P_2 from making y positive". Such an *alternating* satisfaction can be viewed as a winning condition in a two-player game between the system and the environment [24].

Alternating transition systems (ATSs) model reactive components and their interactions, providing a general framework for verification of systems composed from reactive components [1]. *Alternating temporal logics* (ATLs) logically characterize ATSs and have, in addition to the usual universal and existential path quantifiers, a path quantifier that is parameterized by a set Ω of agents[1]. The path quantifier ranges over those paths that the agents in Ω can force the system into no matter how the other agents behave. For example, the ATL formula $\langle\langle\Omega\rangle\rangle\bigcirc(x = y)$ means that the agents in Ω can cooperate to make x and y equal in the next state. Dually, $[\![\Omega]\!]\bigcirc(x = y)$ means that the agents in Ω cannot prevent the next values of x and y from being equal.

The game theoretic-approach, which is the essence of ATS and ATL, has turned out to be very useful. In particular, games are used in compositional verification [9], reasoning about security protocols [19], multi-agent planning [28, 29], control and synthesis [24], and more. The complexity of game solving, however, is higher than that of model checking [1]. Thus, methods for coping with large state spaces are even more crucial than in verification of closed systems.

A key technique for coping with very large or even infinite state spaces is *abstraction*. Abstraction frameworks in the 3-valued semantics [3] are typically based on *modal transition systems* (MTS). Such systems have two types of transitions: *may* transitions, which over-approximate the transitions of the concrete system, and *must* transitions,

*Address: One Microsoft way, Redmond, WA 98052, USA, Email: tball@microsoft.com

†Address: School of Computer Science and Engineering, Jerusalem 91904, Israel. Email: orna@cs.huji.ac.il

[1] We adopt the terminology of game theory and refer to the underlying components as *agents*.

which under-approximate the transitions of the concrete system. Accordingly, verification of universal and existential properties is done with respect to may and must transitions, respectively. One can extend the abstraction framework to an *abstraction-refinement* framework, in which an indefinite answer carries with it information that enables the refinement of the abstract system. In the case of 3-valued semantics, the information comes from analyzing the source of the answer being unknown [25, 26].

We describe an abstraction-refinement framework for games, based on ATSs and ATL. Our abstraction framework for games is based on lifting the notions of may and must transitions to *abstract alternating transition systems* (AATSs), where the may transitions over-approximate the power of the agents, and the must transitions under-approximate them. Accordingly, must transitions are helpful for the verification of properties referring to the ability of the agents to achieve a goal ($\langle\!\langle\ \rangle\!\rangle$ properties), and may transitions are helpful for the verification of properties referring to their disability ($[\![\ ]\!]$ properties).

Two earlier works in this direction are [15] and [8]. In [15], the authors describe an abstract interpretation of game properties: the basic modalities $\langle\langle\Omega\rangle\rangle\bigcirc$ and $[\![\Omega]\!]\bigcirc$ of ATL correspond to the predicate transformers $CPre_\Omega$ and $UPre_\Omega$, which take as an argument a set of agents and return the controllable and uncontrollable predecessors of it. These predicates are extended in [15] to predicates that operate on sets of abstract states, and are used in an abstract model-checking procedure for the *alternating μ-calculus*. In [8], the authors suggest an abstraction framework for turn-based games. In a turn-based game, a single agent proceeds in each position. Thus, turn-based games can model systems with a limited type of concurrency – one in which a single component proceeds in each transition. As noted in [8], the extension of the turn-based setting described there to general concurrent games is technically difficult. As we explain below, the extension carries with it interesting theoretical observations and significantly extends the type of systems for which abstraction can be applied[2].

In addition to defining AATSs, a 3-valued semantics for the alternating μ-calculus with respect to them, and a corresponding model-checking procedure, we make the following contributions. In case the model-checking procedure returns an indefinite answer, we accompany the answer by a suggestion for a *refinement*. Such an automatic refinement procedure does not exist in previous works on abstract games. As in the case of MTS, our procedure analyzes the sources to the "unknown" answer [25, 26][3].

We define an abstraction preorder between AATSs. An *alternating-simulation* preorder between ATSs is defined in [2]. The preorder there is parameterized by a set Ω of agents and $\mathcal{S} \leq_\Omega \mathcal{S}'$ reflects the fact that the agents in Ω are more powerful in $\mathcal{S}$ than in $\mathcal{S}'$. That is, if an alternating μ-calculus formula ψ that expresses the ability of the agents in Ω to achieve some goal is satisfied in $\mathcal{S}'$, it also is satisfied in $\mathcal{S}$. In contrast, our order reflects the abstraction level of the agents in Ω. Thus, if ψ has a definite value in $\mathcal{S}'$, and this value may be either "true" or "false", then it would have a definite value also in $\mathcal{S}$. Our order also is different from the one in [8], which does not take a set of agents as a parameter.

We argue that our definition is the appropriate one in the context of abstraction. In particular, we show that our order, when parameterized with a set Ω of agents, is logically characterized by the fragment of the alternating μ-calculus in which all $\langle\!\langle\ \rangle\!\rangle$ and $[\![\ ]\!]$ quantifiers are parameterized by sets $\Omega' \subseteq \Omega$ of agents[4].

Finally, for the special case of predicate abstraction of software, we show how a theorem prover can be used in order to automatically generate the may and must transitions of the AATS. This involves an extension of the traditional notions of *weakest precondition* to programs with internal nondeterminism and expressing the existence of may and must transitions by means of first order logic formulas that use the extended notions. We demonstrate our approach by verifying properties of a program composed of two processes that concurrently assign variables to integers.

A nice theoretical contribution of our framework is that it unifies three games: the model-checking game (cf. [27]), the abstraction game (cf. [7]), and the game between the different agents. In particular, though the may and must transitions of an AATS have the same structure, which is similar to the one of an ATS, the special case of an AATS with a single agent corresponds to an MTS with hyper-must transitions. Thus, AATSs provide a good explanation, based on the game nature of model checking and abstraction, of the asymmetry between must and may transitions. The appropriateness of the model is also reflected in the fact that AATSs enjoy monotonicity [26] and completeness [6]. From a practical point of view, handling general ATSs broadens the scope of abstraction to systems with full concurrency. In particular, the success of the game-theoretic approach in the verification of security protocols and multi-agent planning is in systems with full concurrency [19, 29], thus the richer setting is the interesting one.

Due to the lack of space, some details are omitted. A full version can be found in the authors' URLs.

[2] In addition, the abstraction in the turn-based setting are limited to *agent-preserving* abstractions, where concrete states that correspond to the same abstract state agree on the agent that proceeds in them. Such a limitation does not exist in our general case.

[3] Note that the standard method of counterexample-based refinement cannot be applied in the 3-valued semantics.

[4] Note that we allow the formulas to refer to both the abilities and disabilities of the agents of Ω. This is in contrast to the "Ω-universal" fragment of [2], where the simulation relation refers to the truth-value lattice (rather than the information lattice), and only the $\langle\!\langle\ \rangle\!\rangle$ quantifier is allowed.

2 The Model

2.1 Alternating transition systems

In ordinary transition systems, each transition corresponds to a possible step of the system. In *alternating transition systems* (ATSs, for short) [1], each transition corresponds to a possible move in a game between the underlying components of the system. We refer to the components as *agents*. In each move of the game, every agent chooses a set of successor states. The game then proceeds to the state in the intersection of the sets chosen by all agents. Equivalently, each agent puts a constraint on the choice of the successor state, and the game proceeds to a state that satisfies the constraints imposed by all the agents.

Formally, an ATS is a 6-tuple $\mathcal{S} = \langle \Pi, \Sigma, S, s_{in}, \pi, \delta \rangle$, where Π is a set of propositions, Σ is a finite set of agents, S is a set of states, s_{in} is an initial state, $\pi : S \times \Pi \to \{\mathbf{T}, \mathbf{F}\}$ maps each state and proposition to the truth value of the proposition in the state, and $\delta : S \times \Sigma \to 2^{2^S}$ is a transition function that maps a state and an agent to a nonempty set of moves, where each move is a set of possible next states. Whenever the system is in state s, each agent σ chooses a set $S_\sigma \in \delta(s, \sigma)$. In this way, an agent σ ensures that the next state of the system will be in its move S_σ. However, which state in S_σ will be next depends on the moves made by the other agents, because the successor of s must lie in the intersection $\bigcap_{\sigma \in \Sigma} S_\sigma$ of the moves made by all the agents. We require that the transition function is nonblocking and that the agents together choose a unique next state: assuming $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$, for every state $s \in S$ and every set $S_1, \ldots, S_n$ of moves $S_i \in \delta(s, \sigma_i)$, the intersection $S_1 \cap \ldots \cap S_n$ is a singleton.

For two states s and s', we say that s' is a *successor* of s if whenever the system $\mathcal{S}$ is in state s, the agents in Σ can cooperate so that s' will be the next state. Thus, for each $\sigma \in \Sigma$, there is $S_\sigma \in \delta(s, \sigma)$ such that $\{s'\} = \bigcap_{\sigma \in \Sigma} S_\sigma$. Consider a state $s \in S$, an agent $\sigma \in \Sigma$, and a set $A \in \delta(s, \sigma)$. If A contains a state s' such that the transition to s' is disabled no matter how the other agents proceed, we can remove s' from A. Accordingly, we assume that the transitions of the ATS contains no redundancy, in the sense that all the states in A are successors of s.

Example 2.1 Consider two variables x and y ranging over the integers $\mathcal{Z}$. We use the predicate s to indicate whether x and y agree on their sign (that is, they are both positive or both negative) and the predicate p to indicate whether x and y agree on their parity (that is, they are both odd or both even). Figure 1 describes a program that assigns values to x and y. For clarity, the next values of x and y are termed x' and y', respectively. The program is a synchronous composition of two processes P_1 and P_2. The processes have internal nondeterministic choices. For example, when P_1 executes $x' := x - 1 \mid x \mid x + 1$, it can resolve the nondeterministic choice in three possible ways and it can make x' either $x - 1$, x, or $x + 1$.

P_1**:**
  while $true$ do
    if $s \wedge p$ then $x' := x - 1 \mid x \ ; \ y' := y - 1 \mid y$;
    if $\neg s \wedge p$ then $x' := x - 1 \mid x \mid x + 1$;
    if $s \wedge \neg p$ then $x' := x - 1 \mid x \mid x + 1$

P_2**:**
  while $true$ do
    if $\neg s \wedge p$ then $y' := y \mid y + 1$;
    if $\neg s \wedge \neg p$ then $x' := x \mid x + 1 \ ; \ y' := y \mid y + 1$

Figure 1. The processes P_1 and P_2.

Note that in some cases both P_1 and P_2 assign values to the variables (for example, when $\neg s \wedge p$ then P_1 assigns a value to x and P_2 assigns a value to y) and in some cases only P_1 or P_2 assigns value (for example, when $s \wedge \neg p$, only P_1 assigns a value to x, and the value of y is unchanged).

The ATS that corresponds to the composition of P_1 with P_2 has state space $\mathcal{Z} \times \mathcal{Z}$ and has the following transitions:

- If $s \wedge p$, then
  - $\delta((x, y), P_1) = \{\{(x - 1, y - 1)\}, \{(x - 1, y)\}, \{(x, y - 1)\}, \{(x, y)\}\}$.
  - $\delta((x, y), P_2) = \{\{(x - 1, y - 1), (x - 1, y), (x, y - 1), (x, y)\}\}$.
- If $\neg s \wedge p$, then
  - $\delta((x, y), P_1) = \{\{(x - 1, y), (x - 1, y + 1)\}, \{(x, y), (x, y + 1)\}, \{(x + 1, y), (x + 1, y + 1)\}\}$.
  - $\delta((x, y), P_2) = \{\{(x - 1, y), (x, y), (x + 1, y)\}, \{(x-1, y+1), (x, y+1), (x+1, y+1)\}\}$.
- If $s \wedge \neg p$, then
  - $\delta((x, y), P_1) = \{\{(x - 1, y)\}, \{(x, y)\}, \{(x + 1, y)\}\}$.
  - $\delta((x, y), P_2) = \{\{(x-1, y), (x, y), (x+1, y)\}\}$.
- If $\neg s \wedge \neg p$, then
  - $\delta((x, y), P_1) = \{\{(x, y), (x, y + 1), (x + 1, y), (x + 1, y + 1)\}\}$.
  - $\delta((x, y), P_2) = \{\{(x, y)\}, \{(x, y + 1)\}, \{(x + 1, y)\}, \{(x + 1, y + 1)\}\}$.

For example, if the current values of x and y are $(-6, 2)$, thus $\neg s$ and p, then P_1 can either decrease x by 1 and force (x', y') to be in $\{(-7, 2), (-7, 3)\}$, leave x unchanged and force (x', y') to be in $\{(-6, 2), (-6, 3)\}$, or increase x by 1 and force (x', y') to be in $\{(-5, 2), (-5, 3)\}$. Process P_1, however, cannot influence the next value of y and it therefore cannot influence which values inside the sets would be the next ones. Process P_2 can either leave y unchanged and force (x', y') to be in $\{(-7, 2), (-6, 2), (-5, 2)\}$, or increase y by 1 and force (x', y') to be in $\{(-7, 3), (-6, 3), (-5, 3)\}$. As with P_1, process P_2 cannot influence the next value of x and it therefore cannot influence which values inside these sets would be the next ones. Once P_1 and P_2 have made their choices, (x', y') is fixed. □

An ordinary *labeled transition system*, or Kripke structure, is the special case of an ATS where the set $\Sigma = \{sys\}$ of agents is a singleton set. In this special case, the sole agent sys can always determine the successor state: for all states $q \in S$, the transition $\delta(q, sys)$ must contain a nonempty set of moves, each of which is a singleton set.

Often, we are interested in the cooperation of a subset $\Omega \subseteq \Sigma$ of the agents. Given Ω, we define $\delta(q, \Omega) = \{T : \text{for each } \sigma \in \Omega \text{ there exists } S_\sigma \in \delta(q, \sigma) \text{ and } T = \bigcap_{\sigma \in \Omega} S_\sigma\}$. For example, if $\Sigma = \{a, b, c\}$, $\delta(q, a) = \{\{q_1, q_2, q_5\}, \{q_3, q_4\}\}$ and $\delta(q, b) = \{\{q_1, q_4, q_5\}, \{q_2, q_3\}\}$, then $\delta(q, \{a, b\}) = \{\{q_1, q_5\}, \{q_2\}, \{q_4\}, \{q_3\}\}$. Intuitively, whenever the system is in state q, the agents in Ω can choose a set $T \in \delta(q, \Omega)$ such that, no matter what the other agents do, the next state of the system is in T. In particular, when all agents cooperate, they can decide the next state, thus, $\delta(q, \Sigma)$ is a set of singletons. Likewise, $\delta(q, \emptyset)$ contains the single set of all successors of q.

2.2 Alternating μ-calculus

The temporal logic AMC (*Alternating μ-calculus*) is the alternating extension of the μ-calculus [18]. Formulas of AMC are defined with respect to a finite set Π of *propositions* and a finite set Σ of *agents*. Formulas of AMC are interpreted over states of an ATS. The $\forall\bigcirc$ and $\exists\bigcirc$ modalities of the μ-calculus are replaced in AMC by the modality $\langle\langle\Omega\rangle\rangle\bigcirc$, for a set Ω of agents. The path quantifier $\langle\langle\Omega\rangle\rangle$ ranges over computations that the agents in Ω can force the system into. Thus, the AMC formula $\langle\langle\Omega\rangle\rangle\bigcirc\theta$ intuitively means that the agents in Ω can cooperate to make θ true in the next state (they can "enforce" the next state to satisfy θ). Formally, $q \models \langle\langle\Omega\rangle\rangle\bigcirc\theta$ iff there is $T \in \delta(q, \Omega)$ such that $q' \models \theta$ for all $q' \in T$. It is often useful to express an AMC formula in a dual form. For this purpose, we use the path quantifier $[\![\Omega]\!]$, for a set Ω of agents. Then, $[\![\Omega]\!]\bigcirc\theta$ means that the agents in Ω cannot cooperate to make θ false in the next state (they cannot avoid θ). Note that[5] $[\![\Omega]\!]\bigcirc\theta = \neg\langle\langle\Omega\rangle\rangle\bigcirc\neg\theta$. The least and greatest fixed-point operators $\mu z.\theta(z)$ and $\nu z.\theta(z)$ can be applied to monotonic AMC formulas and enable the specification of global properties. For a full definition of the syntax and semantics of AMC see [1].

Example 2.2 Consider the ATS from Example 2.1. The state $(1, -1)$ satisfies $\langle\langle P_1\rangle\rangle\bigcirc(x \neq y)$. Indeed, by increasing x by 1 (or leaving it unchanged), the process P_1 can guarantee that, no matter how P_2 modifies the value of y, the next values of x and y would be different. The state $(1, -1)$ also satisfies $\langle\langle P_2\rangle\rangle\bigcirc(x \neq y)$. Indeed, by decreasing y by 1 (or leaving it unchanged), the process P_2 can guarantee that, no matter how P_1 modifies the value of x, the next values of x and y would be different. Finally, $(1, -1)$ also satisfies $\langle\langle\{P_1, P_2\}\rangle\rangle\bigcirc\nu z.(x = y) \wedge \langle\langle\{P_1\}\rangle\rangle\bigcirc z$. Indeed, by decreasing x by 1 and increasing y by 1, the two processes can collaborate and make $x = y = 0$, and then P_1 can keep $x = y = 0$ forever.

We would like to be able to answer questions like "can P_1 make sure that x and y eventually always agree on their parity?", "Can P_2 make y eventually negative?", "can P_1 and P_2 collaborate so that eventually x and y never have the same sign?", and so on. The way we do it is by reasoning about a finite state AATS that abstracts the interaction between the two processes. □

3 Abstraction

For finite state systems, abstraction frameworks often are based on *modal transition systems* (MTS) [20]. Traditional MTS have two types of transitions: $must$ (under-approximating transitions) and may (over-approximating transitions). The idea is that universal properties of a concrete system can be proven by referring to the may transitions of the abstract systems whereas existential properties can be proven by referring to the must transitions. In the case of multi-agent systems, we do not consider universal and existential properties. Instead, we refer to properties that the agents can force the system to satisfy and properties they cannot avoid. Accordingly, rather than using may and must transitions in order to under- and over- approximate the transitions, we are going to use them in order to under- and over- approximate the power of the agents.

[5] On the other hand, note that the path quantifiers $\langle\langle\ \rangle\rangle$ and $[\![\ ]\!]$ are not semantically dual with respect to the set of agents: if the agents in Ω can enforce a set τ of successor states, then the agents in $\Sigma \setminus \Omega$ cannot avoid τ. Therefore, $q \models \langle\langle\Omega\rangle\rangle\bigcirc\psi$ implies $q \models [\![\Sigma \setminus \Omega]\!]\bigcirc\psi$. The converse of this statement, however, is not necessarily true.

3.1 Abstract ATS

An AATS is an ATS $\mathcal{S}' = \langle \Pi, \Sigma, S_A, s_{in}, \pi, \delta_{must}, \delta_{may} \rangle$ in which the labeling function $\pi : S_A \times \Pi \to \{\mathbf{T}, \mathbf{F}, \bot\}$ is three-valued, and there are two types of transitions, $\delta_{must} : S_A \times \Sigma \to 2^{2^{S_A}}$ and $\delta_{may} : S_A \times \Sigma \to 2^{2^{S_A}}$.

The elements of $\{\mathbf{T}, \mathbf{F}, \bot\}$ can be arranged in an "information lattice" [17] in which $\bot \sqsubseteq \mathbf{T}$ and $\bot \sqsubseteq \mathbf{F}$. Note that for two values $v_1, v_2 \in \{\mathbf{T}, \mathbf{F}, \bot\}$, we have $v_1 \sqsubseteq v_2$ iff $v_1 \neq \bot$ implies $v_1 = v_2$.

Consider an ATS $\mathcal{S} = \langle \Pi, \Sigma, S_C, c_{in}, \pi, \delta \rangle$. Let S_A be a set of abstract states and let $\rho : S_C \to S_A$ be an abstraction function[6]. We extend ρ to subsets of S_C in the expected way, thus $\rho(C) = \bigcup_{c \in C} \rho(c)$. We also use $c \in a$ to indicate that $\rho(c) = a$.

An AATS $\mathcal{S}' = \langle \Pi, \Sigma, S_A, a_{in}, \pi', \delta_{must}, \delta_{may} \rangle$ is an abstraction of $\mathcal{S}$ if for all concrete states $c \in S_c$, we have $\pi'(\rho(c)) \sqsubseteq \pi(c)$, and for all abstract states $a \in S_A$ and agents $\sigma \in \Sigma$, the following hold:

- $\delta_{must}(a, \sigma) = \{A \subseteq S_A : \text{ for all } c \in a \text{ there is } C_c \in \delta(c, \sigma), \text{ and } A = \bigcup_{c \in a} \rho(C_c)\}$.
- $\delta_{may}(a, \sigma) = \{A \subseteq S_A : \text{ there is } c \in a \text{ and } C_c \in \delta(c, \sigma) \text{ and } A = \rho(C_c)\}$.

Intuitively, $A \in \delta_{must}(a, \sigma)$ if for each $c \in a$, the agent σ can force the successor of c to correspond to a state in A. Likewise, $A \in \delta_{may}(a, \sigma)$ if for some $c \in a$, the agent σ can force the successor of c to correspond to a state in A. Recall that in MTS, must transitions are used in order to prove existential properties or refute universal properties, whereas may transitions are used in order to prove universal properties or refute existential ones. In AATSs, must transitions are used in order to prove $\langle\!\langle\ \rangle\!\rangle$ properties and refute $[\![\ ]\!]$ properties, whereas may transitions are used in order to prove $[\![\ ]\!]$ properties and refute $\langle\!\langle\ \rangle\!\rangle$ properties.

As with the usual transitions of an ATS, we can refer to the $must$ and may transitions of a set of agents in an AATS. Thus, $\delta_{must}(q, \Omega)$ underapproximates the power of the agents in Ω when they cooperate, and $\delta_{may}(q, \Omega)$ overapproximates their power.

Remark 3.1 An MTS can be viewed as a special case of an AATS – one with a single agent sys. Recall that then, the ATS $\mathcal{S}$ is such that $\delta(c, sys)$ is a set of singletons. Accordingly, in an abstraction of $\mathcal{S}$, we have $A \in \delta_{must}(a, sys)$ iff for every $c \in a$, there exists $\{c'_c\} \in \delta(c, sys)$ and $A = \bigcup_{c \in a} \rho(c'_c)$. Also, $\{a'\} \in \delta_{may}(a, sys)$ iff there is $c \in a$ and $\{c'\} \in \delta(c, sys)$ such that $a' = \rho(c')$. Thus, the definition coincides with the standard definition for hypermust and may transitions [21]. The fact that we get hypermust highlights that AATSs naturally have the game nature of model checking and abstraction "built in": each of the sets $A \in \delta_{must}(a, sys)$ corresponds to a choice the system is making from each of the concrete states that correspond to a. In order for an existential property to hold in a, each of the concrete states should have a successor that satisfies the existential property, and thus $\delta_{must}(a, sys)$ should have a set A all of whose states satisfy the property. □

We define a 3-*valued semantics* of AMC formulas with respect to AATSs. The value of a formula θ in a state a of an AATS $\mathcal{A} = \langle \Pi, \Sigma, S_A, a_{in}, \pi', \delta_{must}, \delta_{may} \rangle$, denoted $[(\mathcal{A}, a) \models \theta]$, is defined as follows. Due to the lack of space, we do not include the semantics of fixed-point operators[7]. The latter is similar to the one described for 3-valued μ-calculus in [4], where the semantics we give below to the $\langle\!\langle\ \rangle\!\rangle$ operator, replaces the one described there for the usual modal operators of μ-calculus.

$$[(\mathcal{A}, a) \models p] = \pi(a, p).$$

$$[(\mathcal{A}, a) \models \neg\theta] = \begin{cases} \mathbf{T} & \text{if } [(\mathcal{A}, a) \models \theta] = \mathbf{F}. \\ \mathbf{F} & \text{if } [(\mathcal{A}, a) \models \theta] = \mathbf{T}. \\ \bot & \text{otherwise.} \end{cases}$$

$$[(\mathcal{A}, a) \models \theta_1 \wedge \theta_2] = \begin{cases} \mathbf{T} & \text{if } [(\mathcal{A}, a) \models \theta_1] = \mathbf{T} \text{ and } [(\mathcal{A}, a) \models \theta_2] = \mathbf{T}. \\ \mathbf{F} & \text{if } [(\mathcal{A}, a) \models \theta_1] = \mathbf{F} \text{ or } [(\mathcal{A}, a) \models \theta_2] = \mathbf{F}. \\ \bot & \text{otherwise.} \end{cases}$$

$$[(\mathcal{A}, a) \models \langle\!\langle \Omega \rangle\!\rangle \bigcirc \theta] = \begin{cases} \mathbf{T} & \text{if there is } A \in \delta_{must}(a, \Omega) \text{ such that } [(\mathcal{A}, a') \models \theta] = \mathbf{T} \text{ for all } a' \in A. \\ \mathbf{F} & \text{if for all } A \in \delta_{may}(a, \Omega), \text{ we have } [(\mathcal{A}, a') \models \theta] = \mathbf{F} \text{ for some } a' \in A. \\ \bot & \text{otherwise.} \end{cases}$$

Abstracting an ATS may cause the truth value of some formulas to become indefinite, but definite values are consistent with the values in the concrete ATS. Formally, we have the following:

Theorem 3.2 *Consider an ATS $\mathcal{S}$, an AATS $\mathcal{A}$ that is an abstraction of $\mathcal{S}$, a state a of $\mathcal{A}$, and an AMC formula θ. For all $c \in a$, we have $[(\mathcal{S}, c) \models \theta] \sqsupseteq [(\mathcal{A}, a) \models \theta]$.*

Remark 3.3 The semantics of the $\langle\!\langle \Omega \rangle\!\rangle \bigcirc$ operator corresponds to our intuition, where in order to prove that the agents in Ω can force the concrete system to a set of states that satisfy θ, one should check that they can achieve this task in the abstraction even if we under-approximate their power and over-approximate the power of the complementary set of agents. Indeed, the semantics of the $\langle\!\langle \Omega \rangle\!\rangle \bigcirc$ operator is equivalent to one in which the agents in Ω proceed with their must transitions and the agents in $\Sigma \setminus \Omega$

[6] Note that since $\mathcal{S}$ is a general ATS, we do not have to limit ρ to an agent preserving function, as is the case with the restricted case of turn-based ATSs [8].

[7] Note that this makes the description of the semantics much cleaner as we do not have to view a formula as a mapping from environments (3-valued assignments to the free variables) to mappings of S_A to $\{\mathbf{T}, \mathbf{F}, \bot\}$.

proceed with their may transitions. Formally, $[(\mathcal{A}, a) \models \langle\langle\Omega\rangle\rangle \bigcirc \varphi] = \mathbf{T}$ iff there is $A \in \delta_{must}(a, \Omega)$ such that for all $A' \in \delta_{may}(a, \Sigma \setminus \Omega)$, we have that $[(\mathcal{A}, a') \models \theta] = \mathbf{T}$ for all $a' \in A \cap A'$. □

Example 3.4 Consider the Processes P_1 and P_2 described in Example 2.1. We define an AATS $\mathcal{S}_s$ according to the predicate s. Thus, $\mathcal{S}_s$ has two states, which we denote by s and $-$. Formally $\mathcal{S}_s = \{\{s\}, \{P_1, P_2\}, \{s, -\}, s, \pi', \delta_{must}, \delta_{may}\}$, where $\pi'(s, s) = \mathbf{T}$, $\pi'(-, s) = \mathbf{F}$, and the transitions are as follows.

- $\delta_{must}(s, P_1) = \{\{s\}\}$
- $\delta_{must}(s, P_2) = \{\{s, -\}\}$
- $\delta_{may}(s, P_1) = \{\{s\}, \{-\}\}$
- $\delta_{may}(s, P_2) = \{\{s\}\}$
- $\delta_{must}(-, P_1) = \{\{s, -\}\}$
- $\delta_{must}(-, P_2) = \{\{s, -\}\}$
- $\delta_{may}(-, P_1) = \{\{s\}, \{-\}\}$
- $\delta_{may}(-, P_2) = \{\{s\}, \{-\}\}$

Let us explain the δ_{must} transition of P_1 from s. By the definition of δ_{must}, we have that $\{s\} \in \delta_{must}(s, P_1)$ iff for all (x, y) that satisfy s, the process P_1 can force (x', y') to satisfy s. This is true, as for all x and y that satisfy s, the set $\delta_{must}((x, y), P_1)$ contains the set $\{(x, y)\}$.

Note that the must transitions underapproximate the power of the processes and the may transitions overapproximate their power. For example, while the only must transition of P_1 from s is to $\{s\}$, it is possible for P_1 to resolve the nondeterminism in the state $(0, 0)$, which satisfies s, so that the next state will be $(-1, 0)$, which does not satisfy s. This is reflected in the may transitions, which overapproximate the power of the processes, and also contains the set $\{-\}$. Likewise, while the only must transition of P_2 from s is to $\{s, -\}$, indicating P_2 cannot influence the next values, there are states (in fact, all states except for those in which $x = 0$ or $y = 0$) that satisfy s for which s is guaranteed to stay true in the next state no matter how P_1 resolves its internal nondeterminism, thus $\{s\} \in \delta_{may}(s, P_2)$.

Even though our abstraction is based on a single predicate, we can verify some properties. For example, since $[(\mathcal{S}_s, s) \models \langle\langle P_1\rangle\rangle \bigcirc s] = \mathbf{T}$, Theorem 3.2, implies that $c \models \langle\langle P_1\rangle\rangle \bigcirc s$ for all concrete states c that satisfy s. In fact, $[(\mathcal{S}_s, s) \models \nu z.s \wedge \langle\langle P_1\rangle\rangle z] = \mathbf{T}$; thus once in a s state, P_1 can force s forever. □

3.2 AMC model checking

The standard symbolic μ-calculus model-checking algorithm of [11] can be extended to a symbolic model-checking algorithm for AMC formulas with respect to ATSs. As we show now, this can be done also with respect to AATSs, yielding a symbolic model-checking algorithm with respect to the abstraction. In more details, the algorithm starts with the innermost subformulas of the specification and computes, for each subformula θ, the sets $|\theta|_{\mathbf{T}}$ and $|\theta|_{\mathbf{F}}$ of abstract states a such that $[(\mathcal{A}, a) \models \theta] = \mathbf{T}$ and $[(\mathcal{A}, a) \models \theta] = \mathbf{F}$, respectively. For Boolean and fixed-point operators, the algorithm proceeds as known symbolic multi-valued model-checking algorithms (c.f., [5]). For the symbolic operator $\langle\langle\Omega\rangle\rangle$, the algorithm proceeds according to the following characterization:

- $|\langle\langle\Omega\rangle\rangle \bigcirc \theta|_{\mathbf{T}} = \{a : \exists A \in \delta_{must}(a, \Omega) \text{ s.t. } A \subseteq |\theta|_{\mathbf{T}}\}$,
- $|\langle\langle\Omega\rangle\rangle \bigcirc \theta|_{\mathbf{F}} = \{a : \forall A \in \delta_{may}(a, \Omega), A \cap |\theta|_{\mathbf{F}} \neq \emptyset\}$.

As discussed in [8], an alternative algorithm reduces the model checking of an AMC formula θ in an AATS $\mathcal{A}$ to model checking of an AMC formula θ' in an ATS $\mathcal{S}$ such that the transition from θ and $\mathcal{A}$ to θ' and $\mathcal{S}$ involves only a linear blow up. Such a reduction is possible also in our case (and is in fact simpler than the one described in [8], as our reduction does not have to end up in a turn-based ATS and does not need the technicality that the latter involves).

3.3 Completeness of abstraction

We now show that our abstraction framework is complete in the sense discussed in [6, 7]. Thus, we can model check a specification θ in an infinite ATS, by reasoning about finite abstractions of it. It is shown in [2] that two states of an ATS satisfy the same AMC formulas iff they are *alternating bisimilar*. An infinite ATS can, in general, have an infinite number of alternating-bisimulation equivalence classes. When, however, we are concerned with the ability of a finite number of AMC formulas to distinguish between states of an ATS, the number of equivalence classes is finite. This finiteness is the key to our completeness result.

In case θ is a safety property (in particular, if θ is in *safe-AMC* — the syntactic fragment of AMC in which formulas are in positive normal form and only the greatest fixed-point operator is allowed), things are simple, as θ induces a finite set of equivalence classes, each consisting of concrete states that are indistinguishable by the subformulas of θ. Formally, we have the following:

Theorem 3.5 *Consider an ATS $\mathcal{S}$, a state c of $\mathcal{S}$, and a safe-AMC formula θ. There is a finite AATS $\mathcal{A}$ such that $\mathcal{A}$ is an abstraction of $\mathcal{S}$ and $[(\mathcal{A}, \rho(c)) \models \theta] \in \{\mathbf{T}, \mathbf{F}\}$.*

Once we allow θ to include least fixed-points, things are more complicated, as the alternating-bisimulation equivalence classes described above are with respect to an AATS augmented with a fairness condition [1]. Thus, completeness is achievable, but goes beyond the model we study here.

4 Abstraction preorder

An *alternating simulation* preorder between two AATSs is defined in [2]. The order is parameterized by a set Ω of

agents and corresponds to the ability of the agents in Ω to restrict the ATS to a smaller set of behaviors in the simulated ATS. In this section we define an abstraction preorder that also is parameterized by a set of agents. Our order, however, corresponds to the agents in Ω being less abstract in the simulated ATS.

For a set S, consider two sets Δ and Δ' in 2^{2^S}. We say that Δ is *more refined than* Δ' if for every set $A' \in \Delta'$, there is $A \in \Delta$ such that $A \subseteq A'$. Thus, each of the sets in Δ' can be restricted to a set in Δ. For example, if $S = \{q_1, q_2, q_3, q_4\}$ then $\{\{q_1\}, \{q_2\}, \{q_3\}\}$ is more refined than $\{\{q_1, q_2\}, \{q_2, q_3\}\}$. Intuitively, if both Δ and Δ' describe the transitions of some agent σ from state q, then σ is more refined with the transitions in Δ than with these in Δ', as it can force the ATS into smaller sets (and possibly more sets) of next successors.

Every must transition is a may transition in the sense that if the agent can force a set in a must transition, it can force a subset of it in a corresponding may transition. Formally, we have the following:

Lemma 4.1 *For every state a and agent σ, we have that $\delta_{may}(a, \sigma)$ is more refined than $\delta_{must}(a, \sigma)$.*

We can now define a preorder $\preceq_\Omega$ between AATSs. The preorder is parameterized by a set Ω of agents. Intuitively, $\mathcal{S} \preceq_\Omega S'$ if the behavior of each of the agents in Ω is less abstract in $\mathcal{S}$ than in $\mathcal{S}'$.

We first extend the definition of "more refined" to sets over different, but related, domains. Consider two sets S and S', and a relation $H \subseteq S \times S'$. For a set $\Delta \in 2^{2^S}$, we use $H(\Delta)$ to denote the set of sets obtained by replacing each member s of a set in Δ by all elements $s' \in S'$ with $H(s, s')$. Thus, $A' \in H(\Delta)$ if there is $A \in \Delta$ and $A' = \bigcup_{s \in A} \{s' : H(s, s')\}$. Now, we say that Δ is *more refined than Δ' with respect to H* (more H-refined, for short) iff $H(\Delta)$ is more refined than Δ'. Thus, each of the sets in Δ' has a set in Δ that corresponds to it. Likewise, Δ' is *more H-refined than* Δ iff Δ' is more refined than $H(\Delta)$. Thus, each of the sets in Δ has a set in Δ' that corresponds to it. For example, if $S = \{q_1, q_2, q_3, q_4, q_5\}$, $S' = \{a_1, a_2, a_3\}$, and $H = \{(q_1, a_1), (q_2, a_1), (q_3, a_2), (q_4, a_3), (q_5, a_3)\}$, then $\{\{q_1, q_2\}, \{q_4\}\}$ is more H-refined than $\{\{a_1, a_2\}\}$ and $\{\{a_1\}, \{a_2\}\}$ is more H-refined than $\{\{q_1, q_2, q_3\}, \{q_3, q_4\}\}$.

Lemma 4.2 *Consider two sets S and S', and a relation $H \subseteq S \times S'$. Consider four sets $\Delta_1, \Delta_2 \in 2^{2^S}$ and $\Delta'_1, \Delta'_2 \in 2^{2^{S'}}$. If Δ_1 is more H-refined than Δ'_1 and Δ_2 is more H-refined than Δ'_2, then $\{A_1 \cap A_2 : A_1 \in \Delta_1$ and $A_2 \in \Delta_2\}$ is more H-refined than $\{A'_1 \cap A'_2 : A'_1 \in \Delta'_1$ and $A'_2 \in \Delta'_2\}$.*

Consider two AATSs $\mathcal{S} = \langle \Pi, \Sigma, S, s_{in}, \pi, \delta_{must}, \delta_{may} \rangle$ and $\mathcal{S}' = \langle \Pi', \Sigma, S', s'_{in}, \pi', \delta'_{must}, \delta'_{may} \rangle$. For a subset $\Omega \subseteq \Sigma$ of agents, a relation $H \subseteq S \times S'$ is an *Ω-abstraction relation from $\mathcal{S}$ to $\mathcal{S}'$* if for all pairs $\langle s, s' \rangle \in H$, the following conditions hold:

(1) $\pi(s) \sqsupseteq \pi'(s')$.

(2) For all $\sigma \in \Omega$, we have that $\delta_{must}(s, \sigma)$ is more H-refined than $\delta'_{must}(s', \sigma)$.

(3) For all $\sigma \in \Omega$, we have that $\delta'_{may}(s', \sigma)$ is more H-refined than $\delta_{may}(s, \sigma)$.

If H is an Ω-abstraction from $\mathcal{S}$ to $\mathcal{S}'$ and $\langle s, s' \rangle \in H$, we write $(\mathcal{S}, s) \preceq_\Omega (\mathcal{S}', s')$, which indicates that the agents in Ω are less abstract in $(\mathcal{S}, s)$ than in $(\mathcal{S}', s')$. That is, the must transitions, which under-approximate the agents' power, are more refined in $\mathcal{S}$ than in $\mathcal{S}'$ (so in $\mathcal{S}$, the under-approximation is "less under"). Dually, the may transitions, which over-approximate their power, are more refined in $\mathcal{S}'$ than in $\mathcal{S}$ (so in $\mathcal{S}$, the over-approximation is "less over").

When $(\mathcal{S}, s_{in}) \preceq_\Omega (\mathcal{S}', s'_{in})$, we write $\mathcal{S} \preceq_\Omega \mathcal{S}'$. Note that the definition of $\preceq_\Omega$ refers to the individual agents in Ω. Thus, by Lemma 4.2, we have the following:

Lemma 4.3 *Let H be an Ω-abstraction from $\mathcal{S}$ to $\mathcal{S}'$. For all $\langle s, s' \rangle \in H$ and $\Omega' \subseteq \Omega$, the following holds:*

(2) *$\delta_{must}(s, \Omega')$ is more H-refined than $\delta'_{must}(s', \Omega')$.*

(3) *$\delta'_{may}(s', \Omega')$ is more H-refined than $\delta_{may}(s, \Omega')$.*

When $\mathcal{S} \preceq_\sigma \mathcal{S}'$ for all agents σ, we say that $\mathcal{S}$ is less abstract than $\mathcal{S}'$, denoted $\mathcal{S} \preceq \mathcal{S}'$.

Remark 4.4 It may be that $\mathcal{S} \preceq_{\Omega_1} \mathcal{S}'$, $\mathcal{S} \not\preceq_{\Omega_2} \mathcal{S}'$, $\mathcal{S}' \preceq_{\Omega_2} \mathcal{S}$, and $\mathcal{S}' \not\preceq_{\Omega_1} \mathcal{S}$. For example, suppose that P_x assigns values to x, P_y assigns values to y, in $\mathcal{S}$ we maintain the concrete value of x and the parity of y, and in $\mathcal{S}'$ we maintain the parity of x and the concrete value of y. Then, $\mathcal{S} \preceq_{P_x} \mathcal{S}'$, $\mathcal{S} \not\preceq_{P_y} \mathcal{S}'$, $\mathcal{S}' \preceq_{P_y} \mathcal{S}$, and $\mathcal{S}' \not\preceq_{P_x} \mathcal{S}$. □

Remark 4.5 Recall that our definition refers to the abstraction level of the agents, and not the power of the agents with respect to each other. To emphasize this fact further, consider two programs, each being a composition of two processes P_1 and P_2. In the first program, P_1 can increase or decrease by 1 the value of both x and y, and P_2 does nothing. In the second program, P_1 can increase or decrease by 1 the value of x, and P_2 can increase or decrease by 1 the value of y. Clearly, P_1 is more powerful in the first program, and the simulation order of [2] would show that. On the other hand, the first program is not less abstract, with respect to either P_1 or P_2, than the second program. Accordingly, if we examine two AATSs, abstracted, say, according to a predicate referring to the parity of x and y, then there is no abstraction relation between the two AATSs. □

By viewing a concrete ATS as an AATS whose may and $must$ transition relations are equivalent to the transition relation of the ATS, we can use the abstraction preorder to relate a concrete system and its abstraction, with respect to all subsets of agents. Formally, we have the following:

Theorem 4.6 *Consider an ATS $\mathcal{S} = \langle \Pi, \Sigma, S_C, c_{in}, \pi, \delta \rangle$, a set of abstract states S_A, and a function $\rho : S_C \to S_A$. Let the AATS $\mathcal{S}' = \langle \Pi', \Sigma, S_A, a_{in}, \pi', \delta_{must}, \delta_{may} \rangle$ be the abstraction of $\mathcal{S}$ according to ρ, and let $H \subseteq S_C \times S_A$ be such that $H(c, a)$ iff $\rho(c) = a$. For all sets Ω of agents, H is an Ω-abstraction relation from $\mathcal{S}$ to $\mathcal{S}'$.*

While the μ-calculus logically characterizes the abstraction preorder on MTSs [13], AMC characterizes the abstraction preorder on AATSs. Formally, for a set Ω of agents, let AMC$^\Omega$ be the fragment of AMC in which all $\langle\!\langle\ \rangle\!\rangle$ and $[\![\ ]\!]$ quantifiers are parameterized by a set $\Omega' \subseteq \Omega$ of agents. Note that we do not require the formulas to be in a positive normal form. Thus, AMC$^\Omega$ formulas refer both to the strength and weakness of the agents of Ω. This is in contrast to the fragment Ω-AMC of [2], where the simulation relation refers to the truth-value lattice rather than the information lattice, and accordingly Ω-AMC formula are in positive normal form and can refer only to the power of the agents of Ω.

Theorem 4.7 *Let $\mathcal{S} = \langle \Pi, \Sigma, S, s_{in}, \pi, \delta_{must}, \delta_{may} \rangle$ and $\mathcal{S}' = \langle \Pi', \Sigma, S', s'_{in}, \pi', \delta'_{must}, \delta'_{may} \rangle$ be two AATSs. Consider a set Ω of agents. For every two states $a \in S$ and $a' \in S'$, we have that $(\mathcal{S}, a) \preceq_\Omega (\mathcal{S}', a')$ iff $[(\mathcal{S}, a) \models \theta] \sqsupseteq [(\mathcal{S}', a')) \models \theta]$ for all AML$^\Omega$ formulas θ.*

Note that, by Theorem 4.6, we have that Theorem 3.2 is a special case of Theorem 4.7.

As with usual simulation relations and alternating-simulation relations [23, 2], a maximal Ω-abstraction relation H between two AATSs can be calculated as a fixed-point of intermediate relations (the sequence $H_0, H_1, \ldots$ used in the proof of Theorem 4.7). Accordingly, we have the following:

Theorem 4.8 *Given two AATSs $\mathcal{S}$ and $\mathcal{S}'$ and a set Ω of agents, deciding whether $S \preceq_\Omega \mathcal{S}'$ can be done in polynomial time.*

5 Refinement

In case the model-checking procedure returns an indefinite answer, we accompany the answer by a suggestion for a *refinement*. As in the case of MTS, our procedure analyzes the sources to the "unknown" answer. Technically, as in [25, 26], the refinement procedure first finds a *failure state* – a state in which the evaluation of the specification became indefinite (with respect to some subformula), and then refines the AATS in a way that makes the satisfaction of this subformula definite. We first show that our model of AATSs enjoys monotonicity, thus the refined AATS gives a definite truth value to at least all formulas that have a definite truth value in the AATS before the refinement.

5.1 Monotonicity

As argued in [26], refining an MTS by splitting a state into two states may result in an MTS with fewer must transitions. As a result, formulas that have a definite value in the original MTS may have an indefinite value in the refined MTS. The solution to this annoying fact is to have hyper-must transitions. As we now show, splitting states of an AATS $\mathcal{S}_1$ that abstracts a concrete ATS results in an AATS $\mathcal{S}_2$ such that $\mathcal{S}_2 \preceq \mathcal{S}_1$. Thus, by Theorem 4.7, monotonicity holds in our framework.

Theorem 5.1 *Consider an ATS $\mathcal{S} = \langle \Pi, \Sigma, S, c_{in}, \pi, \delta \rangle$. Let S_1 and S_2 be sets of abstract states and let $\rho_1 : S \to S_1$ and $\rho_2 : S \to S_2$ be such that for all $c, c' \in S$, if $\rho_2(c) = \rho_2(c')$, then $\rho_1(c) = \rho_1(c')$. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be the AATSs induced by ρ_1 and ρ_2, respectively. Then, $\mathcal{S}_2 \preceq \mathcal{S}_1$.*

5.2 Refinement based on failure states

We can now turn to the problem of finding failure states and using them for refining the AATS. For simplicity, we first handle *alternating modal logic* (AML), that is, AMC without the fixed-point operator. We then discuss, in Section 5.3, the treatment of fixed points.

For an abstract state a and a formula φ, we say that a is a *failure state with respect to* φ if $[(\mathcal{A}, a) \models \varphi] = \bot$ even though $\mathcal{A}$ has definite value for subformulas of φ in the relevant states. Formally, a is a failure state with respect to φ if $[(\mathcal{A}, a) \models \varphi] = \bot$, and in addition, either $\varphi = p \in \Pi$ or $\varphi = \langle\!\langle\Omega\rangle\!\rangle \bigcirc \theta$ and $[(\mathcal{A}, a') \models \theta] \in \{\mathbf{T}, \mathbf{F}\}$, for all the successors a' of a.

Note that if a is a failure state with respect to $\langle\!\langle\Omega\rangle\!\rangle \bigcirc \theta$, then for all $A \in \delta_{must}(a, \Omega)$, there is $a' \in A$ with $[(\mathcal{A}, a') \models \theta] = \mathbf{F}$, and there is $A \in \delta_{may}(a, \Omega)$ such that for all $a' \in A$, we have that $[(\mathcal{A}, a') \models \theta] = \mathbf{T}$.

The drawback of the above definition is that it defines a to be a failure state with respect to θ even if the indefinite value of θ in a is irrelevant to the indefinite value of the specification in the initial state of the AATS. In order to restrict attention to *relevant* failure states, the procedure that searches for failure states proceeds in a top-down manner. The procedure FRFS (find relevant failure states) we describe is similar to the one in [26], only that the treatment of the $\forall\bigcirc$ modality there is generalized to our $\langle\!\langle\Omega\rangle\!\rangle\bigcirc$ modality.

The procedure $\mathsf{FRFS}(a, \psi)$ gets as input an abstract state a and a formula ψ such that $[(\mathcal{A}, a) \models \theta] = \bot$ and return an abstract state a' and a subformula ψ' of ψ such that a' is a failure state with respect to ψ', and the indefinite value of ψ' in a' is relevant to the value of ψ in a being indefinite.

Formally, $\mathsf{FRFS}(a, \psi)$ proceeds as follows.

- If $\psi = p$, then return $\langle a, \psi \rangle$.
- If $\psi = \neg\theta$, then return $\mathsf{FRFS}(a, \theta)$.
- If $\psi = \theta_1 \vee \theta_2$, then let i be $\min\{1, 2\}$ such that $[(\mathcal{A}, a) \models \theta_i] = \bot$; return $\mathsf{FRFS}(a, \theta_i)$.
- If $\psi = \langle\langle\Omega\rangle\rangle\bigcirc\theta$, then if for all the successors a' of a, we have $[(\mathcal{A}, a) \models \theta] \in \{\mathbf{T}, \mathbf{F}\}$, return $\langle a, \psi \rangle$. Otherwise, let a' be a successor of a for which $[(\mathcal{A}, a) \models \theta] = \bot$; return $\mathsf{FRFS}(a', \theta)$.

It is not hard to see that since the initial call to FRFS is with a pair $\langle a, \psi \rangle$ for which $[(\mathcal{A}, a) \models \psi] = \bot$, the "let" statements in the procedure are guaranteed to be satisfied, and it eventually returns a relevant failure state.

Let a be a relevant failure state with respect to φ. We describe a separation of a into two abstract states a_T and a_F such that the value of φ in both states is definite. Intuitively, a_T abstracts the set of concrete states in a that satisfy φ, and a_F abstracts those states that do not satisfy φ. Formally, we have the following:

- If $\varphi = p$, then $conc_T(a) = \{c \in a : p \in L(c)\}$ and $conc_F(a) = \{c \in a : p \notin L(c)\}$.
- If $\varphi = \langle\langle\Omega\rangle\rangle\bigcirc\theta$, we define
  - $conc_T(a) = \{c \in a :$ there is $C_c \in \delta(c, \Omega)$ such that $[(\mathcal{A}, a') \models \theta] = \mathbf{T}]$ for all $a' \in \rho(C_c)\}$.
  - $conc_F(a) = \{c \in a :$ for all $C_c \in \delta(c, \Omega)$, there is $a' \in \rho(C_c)$ with $[(\mathcal{A}, a') \models \theta] = \mathbf{F}]\}$.

Note that $conc_T$ and $conc_F$ form a partition of the concrete states in a. We refine ρ to map the states in $conc_T(a)$ to a_T and map states in $conc_F(a)$ to a_F.

Theorem 5.2 *Iterating the abstraction-refinement process with respect to an abstraction of a finite ATS is guaranteed to terminate with a definite answer.*

The proof, detailed in the full version, shows that in each of the cases, the suggested separation of a causes the value of φ in a_T and a_F to become definite. In addition, the monotonicity of our framework implies that no truth value of other formulas with respect to other states becomes indefinite.

5.3 Handling fixed-points

Consider a fixed-point formula $\psi = \mu z.\theta(z)$. The model-checking algorithm in Section 3.2 calculates the set $|\psi|_{\mathbf{T}}$ as the fixed point of the sequence $|\psi|^0_{\mathbf{T}} = \emptyset$, $|\psi|^1_{\mathbf{T}} = \theta(|\psi|^0_{\mathbf{T}})$, ..., $|\psi|^{i+1}_{\mathbf{T}} = \theta(|\psi|^i_{\mathbf{T}})$, and it calculates $|\psi|_{\mathbf{F}}$ as the fixed point of the sequence $|\psi|^0_{\mathbf{F}} = S_A$, $|\psi|^1_{\mathbf{F}} = \neg\theta(|\psi|^1_{\mathbf{F}})$, ..., $|\psi|^{i+1}_{\mathbf{F}} = \neg\theta(|\psi|^i_{\mathbf{F}})$. If $|\psi|_{\mathbf{T}} \cup |\psi|_{\mathbf{F}} \neq S_A$, then there is a minimal index i such that $|\psi|^i_{\mathbf{T}} \cup |\psi|^i_{\mathbf{F}} \neq S_A$. Accordingly, when we define a to be a failure state with respect to a fixed-point formula ψ with variable z, we parameterize the definition also with an integer i – the iteration in which the value of the variable z becomes indefinite. The reasoning then is along the same lines described for AML formulas.

In fact, every refinement algorithm of MTSs that is based on a symbolic model-checking procedure can be adjusted to AATSs. Indeed, as demonstrated above, such an adjustment replaces the treatment of the modal operator $\forall\bigcirc$ with the one described in Section 5.2 for $\langle\langle\Omega\rangle\rangle\bigcirc$. We note, however, that while such a refinement procedure exists for the temporal logic CTL [26], the refinement procedure for the μ-calculus is based on Zielonka's enumerative algorithm for solving parity games, and thus it is not symbolic [14].

6 Predicate Abstraction

In this section we focus on the special case where the ATS models several concurrent processes, each given as a program. Each program location is associated with a statement $s = s_1 \mid s_2 \mid \cdots \mid s_n$, which denotes an internal nondeterminism: when the process executes s, it chooses $1 \leq i \leq n$ and executes s_i.

When each abstract state is associated with a program location, and thus it also is associated with a statement, we can calculate the may and must transitions by a theorem prover. For a statement s and a predicate e over the state space, the *weakest precondition* $\mathrm{WP}(s, e)$ is such that the execution of s from every state that satisfies $\mathrm{WP}(s, e)$ results in a state that satisfies e, and $\mathrm{WP}(s, e)$ is the weakest predicate for which the above holds [10]. For example, for an assignment statement $x := v$, we have that $\mathrm{WP}(x := v, e) = e[x/v]$ (that is, e with all occurrences of x replaced by v). In the case of MTSs, weakest preconditions can be used in order to automate the generation of must and may transitions [12]. As we show now, the same can be done in AATSs, given a definition of weakest precondition that takes internal nondeterminism into an account.

For a statement $s = s_1 \mid s_2 \mid \cdots \mid s_n$ with internal nondeterminism, we have that $\mathrm{WP}(s, e) = \bigvee_{1 \leq i \leq n} \mathrm{WP}(s_i, e)$. Note that since the nondeterminism is internal, taking the disjunctions of the different weakest preconditions reflects the fact that satisfying one of them is sufficient in order to guarantee that the process can resolve the nondetermin-

istic choices and reach a state satisfying e. For example, $\mathrm{WP}(x := x + 2 \mid x := x - 4, x = 5)$ is $x = 3 \vee x = 9$. In other words, if the agent can choose between increasing x by 2 or decreasing x by 4, the weakest condition with which it can force the system into a state satisfying $x = 5$ is that $x = 3$ or $x = 9$.

For a concrete state c, let $s = s_1^c \mid s_2^c \mid \cdots \mid s_{n^c}^c$ be the statement that agent σ can choose at c. For a set τ of abstract states, $\tau \in \delta_{must}(a, \sigma)$ iff for all $c \in a$, we have that c implies $\bigvee_{1 \leq i \leq n^c} \mathrm{WP}(s, \tau)$. Also, $\tau \in \delta_{may}(a, \sigma)$ iff there is $c \in a$ for which c implies $\bigvee_{1 \leq i \leq n^c} \mathrm{WP}(s, \tau)$.

Example 6.1 Consider again the ATS from Example 2.1. We define an AATS $\mathcal{S}_{s,p}$ according to the predicates s and p. Thus, the AATS has four states, which we denote by sp, s, p, and $-$. In the full version, we describe the $\mathcal{S}_{s,p}$ in detail. Finding the transitions of $\mathcal{S}_{s,p}$ is not an easy task, and we used a theorem prover to generate them. For example, the fact that $\{-\}$ does not belong to $\delta_{may}(s, P_1)$ follows from the validity of the FOL formula $\neg\exists x, y.s(x, y) \wedge \neg p(x, y) \wedge [(\neg s(x - 1, y) \wedge \neg p(x - 1, y)) \vee (\neg s(x, y) \wedge \neg p(x, y)) \vee (\neg s(x + 1, y) \wedge \neg p(x + 1, y))]$. □

In the full version, we show how useful properties of the program from Example 2.1 can be proven by reasoning about $\mathcal{S}_{s,p}$. We also relate the AATS $\mathcal{S}_{s,p}$ with the AATS $\mathcal{S}_s$ described in Example 3.4, and show that $\mathcal{S}_{s,p} \preceq \mathcal{S}_s$.

References

[1] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *JACM*, 49(5):672–713, 2002.

[2] R. Alur, T.A. Henzinger, O. Kupferman, and M.Y. Vardi. Alternating refinement relations. In *Proc. 9th CONCUR*, LNCS 1466, pages 163–178, 1998.

[3] G. Bruns and P. Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *Proc 11th CAV*, pages 274–287, 1999.

[4] G. Bruns and P. Godefroid. Model checking with 3-valued temporal logics. In *Proc 31st ICALP*, LNCS 3142, pages 281–293, 2004.

[5] M. Chechik, B. Devereux, and S. Easterbrook. Implementing a multi-valued symbolic model checker. In *Proc. 7th TACAS*, LNCS 2031, pages 404–419, 2001.

[6] D. Dams and K.S. Namjoshi. The existence of finite abstractions for branching time model checking. In *Proc. 19th LICS*, pages 335–344, 2004.

[7] D. Dams and K.S. Namjoshi. Automata as abstractions. In *Proc. 6th VMCAI*, LNCS 3385, pages 216–232, 2005.

[8] L. de Alfaro, P. Godefroid, and R. Jagadeesan. Three-valued abstractions of games: Uncertainty, but with precision. In *Proc. 19th LICS*, pages 170–179, 2004.

[9] L. de Alfaro, T.A. Henzinger, and F.Y.C. Mang. Detecting errors before reaching them. In *Proc. 12th CAV*, LNCS 1855, pages 186–201, 2000.

[10] E.W. Dijksta. *A Discipline of Programming*. Prentice-Hall, 1976.

[11] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *Proc. 1st LICS*, pages 267–278, 1986.

[12] P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based model checking using modal transition systems. In *Proc. 12th CONCUR*, LNCS 2154, pages 426–440, 2001.

[13] P. Godefroid and R. Jagadeesan. Automatic abstraction using generalized model checking. In *Proc. 14th CAV*, LNCS 2404, pages 137–150, 2002.

[14] O. Grumberg, M. Lange, M. Leucker, and S. Shoham. Don't know in the μ-calculus. In *Proc. 6th CVMAI*, LNCS 3385, pages 233–249, 2005.

[15] T.A. Henzinger, R. Majumdar, F.Y.C. Mang, and J-F Raskin. Abstract interpretation of game properties. In *Proc. 7th SAS*, LNCS 1824, pages 245–252, 2000.

[16] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[17] S.C. Kleene. *Introduction to Metamathematics*. North Holland, 1987.

[18] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[19] S. Kremer and J.-F. Raskin. A game-based verification of non-repudiation and fair exchange protocols. In *Proc. 12th CONCUR*, LNCS 2154, pages 551–565, 2001.

[20] K.G. Larsen and G.B. Thomsen. A modal process logic. In *Proc. 3rd LICS*, Edinburgh, 1988.

[21] K.G. Larsen and L. XinXin. Equation solving using modal transition systems. In *Proc. 5th LICS*, pages 108–117, 1990.

[22] N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[23] R. Milner. *A Calculus of Communicating Systems*, LNCS 92, Springer Verlag, 1980.

[24] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pages 179–190, 1989.

[25] S. Shoham and O. Grumberg. A game-based framework for CTL counterexamples and 3-valued abstraction-refinement. In *Proc. 15th CAV*, LNCS 2725, pages 275–287, 2003.

[26] S. Shoham and O. Grumberg. Monotonic abstraction-refinement for CTL. In *Proc. TACAS*, LNCS 2988, pages 546–560, 2004.

[27] C. Stirling. Games and modal μ-calculus. In *Proc. 13th STACS*, LNCS 1055, pages 298–312, 1996.

[28] W. van der Hoek and M. Wooldridge. Tractable multi agent planning for epistemic goals. In *Proc. 1st International Conference on Autonomous Agents and Multiagent Systems*, pages 1167 – 1174. ACM Press, 2002.

[29] W. van der Hoek and M. Wooldridge. Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. *Studia Logica*, 75(1):125– 157, 2003.