Faster Algorithms for Alternating Refinement Relations

Krishnendu Chatterjee[1], Siddhesh Chaubal[2], and Pritish Kamath[2]

1 **IST Austria (Institute of Science and Technology Austria)**
2 **IIT Bombay**

Abstract

One central issue in the formal design and analysis of reactive systems is the notion of *refinement* that asks whether all behaviors of the implementation is allowed by the specification. The local interpretation of behavior leads to the notion of *simulation*. Alternating transition systems (ATSs) provide a general model for composite reactive systems, and the simulation relation for ATSs is known as alternating simulation. The simulation relation for fair transition systems is called fair simulation. In this work our main contributions are as follows: (1) We present an improved algorithm for fair simulation with Büchi fairness constraints; our algorithm requires $O(n^3 \cdot m)$ time as compared to the previous known $O(n^6)$-time algorithm, where n is the number of states and m is the number of transitions. (2) We present a game based algorithm for alternating simulation that requires $O(m^2)$-time as compared to the previous known $O((n \cdot m)^2)$-time algorithm, where n is the number of states and m is the size of transition relation. (3) We present an iterative algorithm for alternating simulation that matches the time complexity of the game based algorithm, but is more space efficient than the game based algorithm.

1998 ACM Subject Classification D.2.4 Formal methods

Keywords and phrases Simulation and fair simulation, Alternating simulation, Graph games

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.167

1 Introduction

Simulation relation and extensions. One central issue in formal design and analysis of reactive systems is the notion of refinement relations. The refinement relation (system A refines system A') intuitively means that every behavioral option of A (the implementation) is allowed by A' (the specification). The *local* interpretation of behavorial option in terms of successor states leads to refinement as *simulation* [15]. The simulation relation enjoys many appealing properties, such as it has a denotational characterization, it has a logical characterization and it can be computed in polynomial time (as compared to trace containment which is PSPACE-complete). While the notion of simulation was originally developed for transition systems [15], it has many important extensions. Two prominent extensions are as follows: (a) extension for composite systems and (b) extension for fair transition systems.

Alternating simulation relation. Composite reactive systems can be viewed as multi-agent systems [16, 9], where each possible step of the system corresponds to a possible move in a game which may involve some or all component moves. We model multi-agent systems as *alternating transition systems* (ATSs) [1]. In general a multi-agent system consists of a set I of agents, but for algorithmic purposes for simulation we always consider a subset $I' \subseteq I$ of agents against the rest, and thus we will only consider two-agent systems (one agent is the collection I' of agents, and the other is the collection of the rest of the agents). Consider the composite systems $A||B$ and $A'||B$, in environment B. The relation that A refines A'

without constraining the environment B is expressed by generalizing the simulation relation to *alternating simulation relation* [2]. Alternating simulation also enjoys the appealing properties of denotational and logical characterization along with polynomial time computability. We refer the readers to [2] for an excellent exposition of alternating simulation and its applications in design and analysis of composite reactive systems. We briefly discuss some applications of alternating simulation relation. Given a composite system with many components, the problem of refinement of a component (i.e., a component C can be replaced with its implementation C') without affecting the correctness of the composite system is an alternating simulation problem. Similarly, refinement for open reactive systems is also an alternating simulation problem. Finally, graph games provide the mathematical framework to analyze many important problems in computer science, specially in relation to logic, as there is a close connection of automata and graph games (see [17, 8] for details). Alternating simulation provides the technique for state space reduction for graph games, which is a pre-requisite for efficient algorithmic analysis of graph games. Thus computing alternating simulation for ATSs is a core algorithmic question in the formal analysis of composite systems, as well as in the heart of efficient algorithmic analysis of problems related to logic in computer science.

Fair simulation relation. Fair transition systems are extension of transition systems with fairness constraint. A *liveness* (or weak fairness or Büchi fairness) constraint consists of a set B of live states, and requires that runs of the system visit some live state infinitely often. In general the fairness constraint can be a strong fairness constraint instead of a liveness constraint. The notion of simulation was extended to fair transition systems as *fair simulation* [11]. It was shown in [11] that fair simulation also enjoys the appealing properties of denotational and logical characterization, and polynomial time computability (see [11] for many other important properties and discussion on fair simulation). Again the computation of fair simulation with Büchi fairness constraints is an important algorithmic problem for design and analysis of reactive systems with liveness requirements.

Our contributions. In this work we improve the algorithmic complexities of computing fair simulation with Büchi fairness constraints and alternating simulation. In the descriptions below we will denote by n the size of the state space of systems, and by m the size of the transition relation. Our main contributions are summarized below.

1. *Fair simulation.* First we extend the notion of fair simulation to alternating fair simulation for ATSs with Büchi fairness constraints. There are two natural ways of extending the definition of fair simulation to alternating fair simulation, and we show that both the definitions coincide. We present an algorithm to compute the alternating fair simulation relation by a reduction to a game with parity objectives with three priorities. As a special case of our algorithm for fair simulation, we show that the fair simulation relation can be computed in $O(n^3 \cdot m)$ time, as compared to the previous known $O(n^6)$-time algorithm of [11]. Observe that m is at most $O(n^2)$ and thus the worst case running time of our algorithm is $O(n^5)$. Moreover, in many practical examples systems have constant out-degree (for examples see [5]) (i.e., $m = O(n)$), and then our algorithm requires $O(n^4)$ time.
2. *Game based alternating simulation.* We present a game based algorithm for alternating simulation. Our algorithm is based on a reduction to a game with reachability objectives, and requires $O(m^2)$ time, as compared to the previous known algorithm that requires $O((n \cdot m)^2)$ time [2]. One key step of the reduction is to construct the game graph in time linear in the size of the game graph.
3. *Iterative algorithm for alternating simulation.* We present an iterative algorithm to com-

pute the alternating simulation relation. The time complexity of the iterative algorithm matches the time complexity of the game based algorithm, however, the iterative algorithm is more space efficient. (see paragraph on space complexity of Section 4.2 for the detailed comparision). Moreover, both the game based algorithm and the iterative algorithm when specialized to transition systems match the best known algorithms to compute the simulation relation.

We remark that the game based algorithms we obtain for alternating fair simulation and alternating simulation are reductions to standard two-player games on graphs with parity objectives (with three priorities) and reachability objectives. Since such games are well-studied, standard algorithms developed for games can now be used for computation of refinement relations. Our key technical contribution is establishing the correctness of the efficient reductions, and showing that the game graphs can be constructed in linear time in the size of the game graphs. For the iterative algorithm we establish an alternative characterization of alternating simulation, and present an iterative algorithm that simultaneously prunes two relations, without explicitly constructing game graphs (thus saving space), to compute the relation obtained by the alternative characterization. Proofs omitted due to lack of space are available in [4].

2 Definitions

In this section we present all the relevant definitions, and the previous best known results. We present definitions of labeled transition systems (Kripke structures), labeled alternating transitions systems (ATS), fair simulation, and alternating simulation. All the simulation relations we will define are closed under union (i.e., if two relations are simulation relations, then so is their union), and we will consider the maximum simulation relation. We also present relevant definitions for graph games that will be later used for the improved results.

▶ **Definition 1** (Labeled transition systems (TS)). A labeled *transition system (TS)* (Kripke structure) is a tuple $K = \langle \Sigma, W, \widehat{w}, R, L \rangle$, where Σ is a finite set of observations; W is a finite set of states and $\widehat{w}$ is the initial state; $R \subseteq W \times W$ is the transition relation; and $L : W \to \Sigma$ is the labeling function that maps each state to an observation. For technical convenience we assume that for all $w \in W$ there exists $w' \in W$ such that $(w, w') \in R$.

Runs, fairness constraint, and fair transition systems. For a TS K and a state $w \in W$, a w-run of K is an infinite sequence $\overline{w} = w_0, w_1, w_2, \ldots$ of states such that $w_0 = w$ and $R(w_i, w_{i+1})$ for all $i \geq 0$. We write $\mathsf{Inf}(\overline{w})$ for the set of states that occur infinitely often in the run $\overline{w}$. A run of K is a $\widehat{w}$-run for the initial state $\widehat{w}$. In this work we will consider *Büchi fairness constraints*, and a Büchi fairness constraint is specified as a set $F \subseteq W$ of Büchi states, and defines the fair set of runs, where a run $\overline{w}$ is *fair* iff $\mathsf{Inf}(\overline{w}) \cap F \neq \emptyset$ (i.e., the run visits F infinitely often). A *fair transition system* $\mathcal{K} = \langle K, F \rangle$ consists of a TS K and a Büchi fairness constraint $F \subseteq W$ for K. We consider two TSs $K_1 = \langle \Sigma, W_1, \widehat{w}_1, R_1, L_1 \rangle$ and $K_2 = \langle \Sigma, W_2, \widehat{w}_2, R_2, L_2 \rangle$ over the same alphabet, and the two fair TSs $\mathcal{K}_1 = \langle K_1, F_1 \rangle$ and $\mathcal{K}_2 = \langle K_2, F_2 \rangle$. We now define the fair simulation between $\mathcal{K}_1$ and $\mathcal{K}_2$ [11].

▶ **Definition 2** (Fair simulation). A binary relation $S \subseteq W_1 \times W_2$ is a *fair simulation* of $\mathcal{K}_1$ by $\mathcal{K}_2$ if the following two conditions hold for all $(w_1, w_2) \in W_1 \times W_2$:

1. If $S(w_1, w_2)$, then $L_1(w_1) = L_2(w_2)$.
2. There exists a strategy $\tau : (W_1 \times W_2)^+ \times W_1 \to W_2$ such that if $S(w_1, w_2)$ and $\overline{w} = u_0, u_1, u_2, \ldots$ is a fair w_1-run of $\mathcal{K}_1$, then the following conditions hold: (a) the outcome $\tau[\overline{w}] = u'_0, u'_1, u'_2, \ldots$ is a fair w_2-run of $\mathcal{K}_2$ (where the outcome $\tau[\overline{w}]$ is defined as follows:

for all $i \geq 0$ we have $u'_i = \tau((u_0, u'_0), (u_1, u'_1), \ldots, (u_{i-1}, u'_{i-1}), u_i))$; and (b) the outcome $\tau[\overline{w}]$ S-matches $\overline{w}$; that is, $S(u_i, u'_i)$ for all $i \geq 0$. We say τ is a *witness* to the fair simulation S.

We denote by $\preceq_{\mathsf{fair}}$ the maximum fair simulation relation between $\mathcal{K}_1$ and $\mathcal{K}_2$. We say that the fair TS $\mathcal{K}_2$ *fairly simulates* the fair TS $\mathcal{K}_1$ iff $(\widehat{w}_1, \widehat{w}_2) \in \preceq_{\mathsf{fair}}$.

We have the following result for fair simulation from [11] (see item 1 of Theorem 4.2 from [11]).

▶ **Theorem 3.** *Given two fair TSs $\mathcal{K}_1$ and $\mathcal{K}_2$, the problem of whether $\mathcal{K}_2$ fairly simulates $\mathcal{K}_1$ can be decided in time $O((|W_1| + |W_2|) \cdot (|R_1| + |R_2|) + (|W_1| \cdot |W_2|)^3)$.*

▶ **Definition 4** (Labeled alternating transition systems (ATS))**.** A labeled *alternating transitions system (ATS)* is a tuple $K = \langle \Sigma, W, \widehat{w}, A_1, A_2, P_1, P_2, L, \delta \rangle$, where (i) Σ is a finite set of observations; (ii) W is a finite set of states with $\widehat{w}$ the initial state; (iii) A_i is a finite set of actions for Agent i, for $i \in \{1, 2\}$; (iv) $P_i : W \to 2^{A_i} \setminus \emptyset$ assigns to every state w in W the non-empty set of actions available to Agent i at w, for $i \in \{1, 2\}$; (v) $L : W \to \Sigma$ is the labeling function that maps every state to an observation; and (vi) $\delta : W \times A_1 \times A_2 \to W$ is the transition relation that given a state and the joint actions gives the next state.

Observe that a TS can be considered as a special case of ATS with A_2 singleton (say $A_2 = \{\perp\}$), and the transition relation R of a TS is described by the transition relation $\delta : W \times A_1 \times \{\perp\} \to W$ of the ATS.

▶ **Definition 5** (Alternating simulation)**.** Given two ATS, $K = \langle \Sigma, W, \widehat{w}, A_1, A_2, P_1, P_2, L, \delta \rangle$ and $K' = \langle \Sigma, W', \widehat{w}', A'_1, A'_2, P'_1, P'_2, L', \delta' \rangle$ a binary relation $S \subseteq W \times W'$ is an alternating simulation from $\mathcal{K}$ to $\mathcal{K}'$ if for all states w and w' with $(w, w') \in S$, the following conditions hold :

1. $L(w) = L'(w')$
2. For every action $a \in P_1(w)$, there exists an action $a' \in P'_1(w')$ such that for every action $b' \in P'_2(w')$, there exists an action $b \in P_2(w)$ such that $(\delta(w, a, b), \delta'(w', a', b')) \in S$.

We denote by $\preceq_{\mathsf{altsim}}$ the maximum alternating simulation relation between K and K'. We say that the ATS K' *simulates* the ATS K iff $(\widehat{w}, \widehat{w}') \in \preceq_{\mathsf{altsim}}$.

The following result was shown in [2] (see proof of Theorem 3 of [2]).

▶ **Theorem 6.** *For two ATSs K and K', the alternating simulation relation $\preceq_{\mathsf{altsim}}$ can be computed in time $O(|W|^2 \cdot |W'|^2 \cdot |A_1| \cdot |A'_1| \cdot |A_2| \cdot |A'_2|)$.*

In the following section we will present an extension of the notion of fair simulation for TSs to alternating fair simulation for ATSs, and present improved algorithms to compute $\preceq_{\mathsf{fair}}$ and $\preceq_{\mathsf{altsim}}$. Some of our algorithms will be based on reduction to two-player games on graphs. We present the required definitions below.

Two-player Game graphs. A *two-player game graph* $G = ((V, E), (V_1, V_2))$ consists of a directed graph (V, E) with a set V of n vertices and a set E of m edges, and a partition (V_1, V_2) of V into two sets. The vertices in V_1 are *player 1 vertices*, where player 1 chooses the outgoing edges; and the vertices in V_2 are *player 2 vertices*, where player 2 (the adversary to player 1) chooses the outgoing edges. For a vertex $u \in V$, we write $\mathsf{O}ut(u) = \{v \in V \mid (u, v) \in E\}$ for the set of successor vertices of u and $\mathsf{I}n(u) = \{v \in V \mid (v, u) \in E\}$ for the set of incoming edges of u. We assume that every vertex has at least one out-going edge. i.e., $\mathsf{O}ut(u)$ is non-empty for all vertices $u \in V$.

Plays. A game is played by two players: player 1 and player 2, who form an infinite path in the game graph by moving a token along edges. They start by placing the token on an initial vertex, and then they take moves indefinitely in the following way. If the token is on a vertex in V_1, then player 1 moves the token along one of the edges going out of the vertex. If the token is on a vertex in V_2, then player 2 does likewise. The result is an infinite path in the game graph, called a *play*. We write Ω for the set of all plays.

Strategies. A strategy for a player is a rule that specifies how to extend plays. Formally, a *strategy* α for player 1 is a function $\alpha\colon V^* \cdot V_1 \to V$ such that for all $w \in V^*$ and all $v \in V_1$ we have $\alpha(w \cdot v) \in \mathsf{Out}(v)$, and analogously for player 2 strategies. We write $\mathcal{A}$ and $\mathcal{B}$ for the sets of all strategies for player 1 and player 2, respectively. A *memoryless* strategy for player 1 is independent of the history and depends only on the current state, and can be described as a function $\alpha : V_1 \to V$, and similarly for player 2. Given a starting vertex $v \in V$, a strategy $\alpha \in \mathcal{A}$ for player 1, and a strategy $\beta \in \mathcal{B}$ for player 2, there is a unique play, denoted $\omega(v, \alpha, \beta) = \langle v_0, v_1, v_2, \ldots \rangle$, which is defined as follows: $v_0 = v$ and for all $k \geq 0$, if $v_k \in V_1$, then $\alpha(v_k) = v_{k+1}$, and if $v_k \in V_2$, then $\beta(v_k) = v_{k+1}$. We say a play ω is *consistent* with a strategy of a player, if there is a strategy of the opponent such that given both the strategies the unique play is ω.

Objectives. An objective Φ for a game graph is a desired subset of plays. For a play $\omega = \langle v_0, v_1, v_2, \ldots \rangle \in \Omega$, we define $\mathsf{Inf}(\omega) = \{v \in V \mid v_k = v \text{ for infinitely many } k \geq 0\}$ to be the set of vertices that occur infinitely often in ω. We define reachability, safety and parity objectives with three priorities.

1. *Reachability and safety objectives.* Given a set $T \subseteq V$ of vertices, the reachability objective $\mathrm{Reach}(T)$ requires that some vertex in T be visited, and dually, the safety objective $\mathrm{Safe}(F)$ requires that only vertices in F be visited. Formally, the sets of winning plays are $\mathrm{Reach}(T) = \{\langle v_0, v_1, v_2, \ldots \rangle \in \Omega \mid \exists k \geq 0.\ v_k \in T\}$ and $\mathrm{Safe}(F) = \{\langle v_0, v_1, v_2, \ldots \rangle \in \Omega \mid \forall k \geq 0.\ v_k \in F\}$. The reachability and safety objectives are dual in the sense that $\mathrm{Reach}(T) = \Omega \setminus \mathrm{Safe}(V \setminus T)$.
2. *Parity objectives with three priorities.* Consider a priority function $p : V \to \{0, 1, 2\}$ that maps every vertex to a priority either 0, 1 or 2. The parity objective requires that the minimum priority visited infinitely often is even. In other words, the objectives require that either vertices with priority 0 are visited infinitely often, or vertices with priority 1 are visited finitely often. Formally the set of winning plays is $\mathrm{Parity}(p) = \{\omega \mid \mathsf{Inf}(\omega) \cap p^{-1}(0) \neq \emptyset \text{ or } \mathsf{Inf}(\omega) \cap p^{-1}(1) = \emptyset\}$.

Winning strategies and sets. Given an objective $\Phi \subseteq \Omega$ for player 1, a strategy $\alpha \in \mathcal{A}$ is a *winning strategy* for player 1 from a vertex v if for all player 2 strategies $\beta \in \mathcal{B}$ the play $\omega(v, \alpha, \beta)$ is winning, i.e., $\omega(v, \alpha, \beta) \in \Phi$. The winning strategies for player 2 are defined analogously by switching the role of player 1 and player 2 in the above definition. A vertex $v \in V$ is winning for player 1 with respect to the objective Φ if player 1 has a winning strategy from v. Formally, the set of *winning vertices for player 1* with respect to the objective Φ is the set $W_1(\Phi) = \{v \in V \mid \exists \alpha \in \mathcal{A}.\ \forall \beta \in \mathcal{B}.\ \omega(v, \alpha, \beta) \in \Phi\}$. Analogously, the set of all winning vertices for player 2 with respect to an objective $\Psi \subseteq \Omega$ is $W_2(\Psi) = \{v \in V \mid \exists \beta \in \mathcal{B}.\ \forall \alpha \in \mathcal{A}.\ \omega(v, \alpha, \beta) \in \Psi\}$.

▶ **Theorem 7** (Determinacy and complexity). *The following assertions hold.*

1. *For all game graphs $G = ((V, E), (V_1, V_2))$, all objectives Φ for player 1 where Φ is reachability, safety, or parity objectives with three priorities, and the complementary objective $\Psi = \Omega \setminus \Phi$ for player 2, we have $W_1(\Phi) = V \setminus W_2(\Psi)$; and memoryless winning strategies exist for both players from their respective winning set [7].*
2. *The winning set $W_1(\Phi)$ can be computed in linear time ($O(|V| + |E|)$) for reachability*

and safety objectives Φ [12, 3]; and in quadratic time ($O(|V| \cdot |E|)$) for parity objectives with three priorities [13].

3 Fair Alternating Simulation

In this section we will present two definitions of fair alternating simulation, show their equivalence, present algorithms for solving fair alternating simulations, and our algorithms specialized to fair simulation will improve the bound of the previous algorithm (Theorem 3). Similar to fair TSs, a *fair ATS* $\mathcal{K} = \langle K, F \rangle$ consists of an ATS K and a Büchi fairness constraint F for K.

To extend the definition of fair simulation to fair alternating simulation we consider the notion of strategies for ATSs. Consider two ATSs $K = \langle \Sigma, W, \widehat{w}, A_1, A_2, P_1, P_2, L, \delta \rangle$ and $K' = \langle \Sigma, W', \widehat{w}', A'_1, A'_2, P'_1, P'_2, L', \delta' \rangle$ and the corresponding fair ATSs $\mathcal{K} = \langle K, F \rangle$ and $\mathcal{K}' = \langle K', F' \rangle$. We use the following notations:

- $\tau : (W \times W')^+ \to A_1$ is a strategy employed by Agent 1 in $\mathcal{K}$. The aim of the strategy is to choose transitions in $\mathcal{K}$ to make it difficult for Agent 1 in $\mathcal{K}'$ to match them. The strategy acts on the past run on both systems.
- $\tau' : (W \times W')^+ \times A_1 \to A'_1$ is a strategy employed by Agent 1 in $\mathcal{K}'$. The aim of this strategy is to match actions in $\mathcal{K}'$ to those made by Agent 1 in $\mathcal{K}$. The strategy acts on the past run on both the systems, as well as the action chosen by Agent 1 in $\mathcal{K}$.
- $\xi' : (W \times W')^+ \times A_1 \times A'_1 \to A'_2$ is a strategy employed by Agent 2 in $\mathcal{K}'$. The aim of this strategy is to choose actions in $\mathcal{K}'$ to make it difficult for Agent 2 to match them in $\mathcal{K}$. The strategy acts on the past run of both the systems, as well as the actions chosen by Agent 1 in $\mathcal{K}$ and $\mathcal{K}'$.
- $\xi : (W \times W')^+ \times A_1 \times A'_1 \times A'_2 \to A_2$ is a strategy employed by Agent 2 in $\mathcal{K}$. Intuitively, the aim of this strategy of Agent 2 is to choose actions in $\mathcal{K}$ to show that Agent 1 is not as powerful in $\mathcal{K}$ as in $\mathcal{K}'$, i.e., in some sense the strategy of Agent 2 will witness that the strategy of Agent 1 in $\mathcal{K}$ does not satisfy certain desired property. The strategy acts on the past run of both the systems, as well as the actions chosen by Agent 1 in $\mathcal{K}$ and both the agents in $\mathcal{K}'$.
- $\rho(w, w', \tau, \tau', \xi, \xi')$ is the run that emerges in $\mathcal{K}$ if the game starts with $\mathcal{K}$ on state w, $\mathcal{K}'$ on state w' and the agents employ strategies τ, τ', ξ and ξ' as described above, and $\rho'(w, w', \tau, \tau', \xi, \xi')$ is the corresponding run that emerges in $\mathcal{K}'$.

▶ **Definition 8** (Weak fair alternating simulation)**.** A binary relation $S \subseteq W \times W'$ is a *weak fair alternating simulation (WFAS)* of $\mathcal{K}$ by $\mathcal{K}'$ if the following two conditions hold for all $(w, w') \in W \times W'$:

1. If $S(w, w')$, then $L(w) = L'(w')$.
2. There exists a strategy $\tau' : (W \times W')^+ \times A_1 \to A'_1$ for Agent 1 in $\mathcal{K}'$, such that for all strategies $\tau : (W \times W')^+ \to A_1$ for Agent 1 in $\mathcal{K}$, there exists a strategy $\xi : (W \times W')^+ \times A_1 \times A'_1 \times A_2 \to A'_2$ for Agent 2 in $\mathcal{K}$, such that for all strategies $\xi' : (W \times W')^+ \times A_1 \times A'_1 \to A'_2$ for Agent 2 on $\mathcal{K}'$, if $S(w, w')$ and $\rho(w, w', \tau, \tau', \xi, \xi')$ is a fair w-run of $\mathcal{K}$, then
   - $\rho'(w, w', \tau, \tau', \xi, \xi')$ is a fair w'-run of $\mathcal{K}'$; and
   - $\rho'(w, w', \tau, \tau', \xi, \xi')$ S-matches $\rho(w, w', \tau, \tau', \xi, \xi')$.

We denote by $\preceq^{\mathsf{weak}}_{\mathsf{fairalt}}$ the maximum WFAS relation between $\mathcal{K}$ and $\mathcal{K}'$. We say that the fair ATS $\mathcal{K}'$ *weak-fair-alternate simulates* the fair ATS $\mathcal{K}$ iff $(\widehat{w}, \widehat{w}') \in \preceq^{\mathsf{weak}}_{\mathsf{fairalt}}$.

▶ **Definition 9** (Strong fair alternating simulation)**.** A binary relation $S \subseteq W \times W'$ is a *strong fair alternating simulation (SFAS)* of $\mathcal{K}$ by $\mathcal{K}'$ if the following two conditions hold for all $(w, w') \in W \times W'$:

1. If $S(w, w')$, then $L(w) = L'(w')$.
2. There exist strategies $\tau' : (W \times W')^+ \times A_1 \to A'_1$ for Agent 1 in $\mathcal{K}'$ and $\xi : (W \times W')^+ \times A_1 \times A'_1 \times A_2 \to A'_2$ for Agent 2 in $\mathcal{K}$, such that for all strategies $\tau : (W \times W')^+ \to A_1$ for Agent 1 in $\mathcal{K}$ and $\xi' : (W \times W')^+ \times A_1 \times A'_1 \to A'_2$ for Agent 2 on $\mathcal{K}'$, if $S(w, w')$ and $\rho(w, w', \tau, \tau', \xi, \xi')$ is a fair w-run of $\mathcal{K}$, then
   - $\rho'(w, w', \tau, \tau', \xi, \xi')$ is a fair w'-run of $\mathcal{K}'$; and
   - $\rho'(w, w', \tau, \tau', \xi, \xi')$ S-matches $\rho(w, w', \tau, \tau', \xi, \xi')$.

We denote by $\preceq^{\mathsf{strong}}_{\mathsf{fairalt}}$ the maximum SFAS relation between $\mathcal{K}$ and $\mathcal{K}'$. We say that the fair ATS $\mathcal{K}'$ *strong-fair-alternate simulates* the fair ATS $\mathcal{K}$ iff $(\widehat{w}, \widehat{w}') \in \preceq^{\mathsf{strong}}_{\mathsf{fairalt}}$.

The difference in the definitions of weak and strong alternating fair simulation is in the order of the quantifiers in the strategies. In the weak version the quantifier order is exists forall exists forall, whereas in the strong version the order is exists exists forall forall. Thus strong fair alternating simulation implies weak fair alternating simulation. We will show that both the definitions coincide and present algorithms to compute the maximum fair alternating simulation. Also observe that both the weak and strong version coincide with fair simulation for TSs. We will present a reduction of weak and strong fair alternating simulation problem to games with parity objectives with three priorities. We now present a few notations related to the reduction.

Successor sets. Given an ATS K, for a state w and an action $a \in P_1(w)$, let $\mathsf{Succ}(w, a) = \{w' \mid \exists b \in P_2(w) \text{ such that } w' = \delta(w, a, b)\}$ denote the possible successors of w given an action a of Agent 1 (i.e., successor set of w and a). Let $\mathsf{Succ}(K) = \{\mathsf{Succ}(w, a) \mid w \in W, a \in P_1(w)\}$ denote the set of all possible successor sets. Note that $|\mathsf{Succ}(K)| \leq |W| \cdot |A_1|$.

Game construction. Let $K = \langle \Sigma, W, \widehat{w}, A_1, A_2, P_1, P_2, L, \delta \rangle$ and $K' = \langle \Sigma, W', \widehat{w}', A'_1, A'_2, P'_1, P'_2, L', \delta' \rangle$ be two ATSs, and let $\mathcal{K} = \langle K, F \rangle$ and $\mathcal{K}' = \langle K', F' \rangle$ be the two corresponding fair ATSs. We will construct a game graph $G = ((V, E), (V_1, V_2))$ with a parity objective. Before the construction we assume that from every state $w \in K$ there is an Agent-1 strategy to ensure fairness in K. The assumption is without loss of generality because if there is no such strategy from w, then trivially all states w' with same label as w simulate w (as Agent 2 can falsify the fairness from w). The states in K from which fairness cannot be ensured can be identified with a quadratic time pre-processing step in K (solving Büchi games), and hence we assume that in all remaining states in K fairness can be ensured. The game construction is as follows:

- *Player 1 vertices:* $V_1 = \{\langle w, w' \rangle \mid w \in W, w' \in W' \text{ such that } L(w) = L'(w')\} \cup \big(\mathsf{Succ}(K) \times \mathsf{Succ}(K')\big) \cup \{\frowny\}$
- *Player 2 vertices:* $V_2 = \mathsf{Succ}(K) \times W' \times \{\#, \$\}$
- *Edges.* We specify the edges as the following union: $E = E_1 \cup E_2 \cup E_3 \cup E^1_4 \cup E^2_4 \cup E_5$

$$
\begin{aligned}
E_1 &= \{(\langle w, w' \rangle, \langle \mathsf{Succ}(w, a), w', \# \rangle) \mid \langle w, w' \rangle \in V_1, a \in P_1(w)\} \\
E_2 &= \{(\langle T, w', \# \rangle, \langle T, \mathsf{Succ}(w', a') \rangle) \mid \langle T, w', \# \rangle \in V_2, a' \in P'_1(w')\} \\
E_3 &= \{(\langle T, T' \rangle, \langle T, r', \$ \rangle) \mid \langle T, T' \rangle \in V_1, r' \in T'\} \\
E^1_4 &= \{(\langle T, r', \$ \rangle, \langle r, r' \rangle) \mid \langle T, r', \$ \rangle \in V_2, r \in T, L(r) = L'(r')\} \\
E^2_4 &= \{(\langle T, r', \$ \rangle, \frowny) \mid \langle T, r', \$ \rangle \in V_2 \text{ such that } \forall r \in T \cdot L(r) \neq L'(r')\} \\
E_5 &= \{(\frowny, \frowny)\}
\end{aligned}
$$

The intuitive description of the game graph is as follows: (i) the player 1 vertices are either state pairs $\langle w, w' \rangle$ with same label, or pairs $\langle T, T' \rangle$ of successor sets, or a state ☹; and (ii) the player 2 vertices are tuples $\langle T, w', \bowtie \rangle$ where T is a successor set in $\mathsf{Succ}(K)$, w' a state in K' and $\bowtie \in \{\#, \$\}$. The edges are described as follows: (i) E_1 describes that in vertices $\langle w, w' \rangle$ player 1 can choose an action $a \in P_1(w)$, and then the next vertex is the player 2 vertex $\langle \mathsf{Succ}(w, a), w', \# \rangle$; (ii) E_2 describes that in vertices $\langle T, w', \# \rangle$ player 2 can choose an action $a' \in P_1(w')$ and then the next vertex is $\langle T, \mathsf{Succ}(w', a') \rangle$; (iii) E_3 describes that in states $\langle T, T' \rangle$ player 1 can choose a state $r' \in T'$ (which intuitively corresponds to an action $b' \in P_2'(w')$) and then the next vertex is $\langle T, r', \$ \rangle$; (iv) the edges $E_4^1 \cup E_4^2$ describes that in states $\langle T, r', \$ \rangle$ player 2 can either choose a state $r \in T$ that matches the label of r' and then the next vertex is the player 1 vertex $\langle r, r' \rangle$ (edges E_4^1) or if there is no match, then the next vertex is ☹; and (v) finally E_5 specifies that the vertex ☹ is an absorbing (sink) vertex with only self-loop. The three-priority parity objective Φ^* for player 2 with the priority function p is specified as follows: for vertices $v \in (W \times F') \cap V_1$ we have $p(v) = 0$; for vertices $v \in ((F \times W' \setminus W \times F') \cap V_1) \cup \{☹\}$ we have $p(v) = 1$; and all other vertices have priority 2. The following proposition establishes the equivalence of the winning set for player 2 with strong and weak fair alternating simulation.

▶ Proposition 10. Let $\mathsf{Win}_2 = \{(w_1, w_2) \mid \langle w_1, w_2 \rangle \in V_1, \langle w_1, w_2 \rangle \in W_2(\Phi^*)\}$ be the winning set for player 2. Then we have $\mathsf{Win}_2 = \preceq^{\mathsf{weak}}_{\mathsf{fairalt}} = \preceq^{\mathsf{strong}}_{\mathsf{fairalt}}$.

▶ **Lemma 11.** *For the game graph constructed for fair alternating simulation we have* $|V_1| + |V_2| \leq O(|W| \cdot |W'| \cdot |A_1| \cdot |A_1'|)$*; and* $|E| \leq O(|W| \cdot |W'| \cdot |A_1| \cdot (|A_1'| \cdot |A_2'| + |A_2|))$.

The above lemma bounds the size of the game, and we require that the game graph can be constructed in time quadratic in the size of the game graph and in the following section we will present a more efficient (than quadratic) construction of the game graph. Proposition 10, along with the complexity to solve parity games with three priorities gives us the following theorem. The result for fair simulation follows as a special case and the details are presented in [4].

▶ **Theorem 12.** *We have* $\preceq^{\mathsf{weak}}_{\mathsf{fairalt}} = \preceq^{\mathsf{strong}}_{\mathsf{fairalt}}$*, the relation* $\preceq^{\mathsf{strong}}_{\mathsf{fairalt}}$ *can be computed in time* $O(|W|^2 \cdot |W'|^2 \cdot |A_1|^2 \cdot |A_1'| \cdot (|A_1'| \cdot |A_2'| + |A_2|))$ *for two fair ATSs* $\mathcal{K}$ *and* $\mathcal{K}'$*. The fair simulation relation* $\preceq_{\mathsf{fair}}$ *can be computed in time* $O(|W| \cdot |W'| \cdot (|W| \cdot |R'| + |W'| \cdot |R|))$ *for two fair TSs* $\mathcal{K}$ *and* $\mathcal{K}'$*.*

▶ Remark. We consider the complexity of fair simulation, and let $n = |W| = |W'|$ and $m = |R| = |R'|$. The previous algorithm of [11] requires time $O(n^6)$ and our algorithm requires time $O(n^3 \cdot m)$. Since m is at most n^2, our algorithm takes in worst case time $O(n^5)$ and in most practical cases we have $m = O(n)$ and then our algorithm requires $O(n^4)$ time as compared to the previous known $O(n^6)$ algorithm.

4 Alternating Simulation

In this section we will present two algorithms to compute the maximum alternating simulation relation for two ATSs K and K'. The first algorithm for the problem was presented in [2] and we refer to the algorithm as the basic algorithm. The basic algorithm iteratively considers pairs of states and examines if they are already witnessed to be not in the alternating simulation relation, removes them and continues until a fix-point is reached (see Theorem 3 of [2]). The correctness of the basic algorithm was shown in [2], and the time complexity of the algorithm is $O(|W|^2 \cdot |W'|^2 \cdot |A_1| \cdot |A_1'| \cdot |A_2| \cdot |A_2'|)$ (see fuller version for further explanation).

4.1 Improved Algorithm Through Games

In this section we present an improved algorithm for alternating simulation by reduction to reachability-safety games.

Game construction. Given two ATSs $K = (\Sigma, W, \widehat{w}, A_1, A_2, P_1, P_2, L, \delta)$ and $K' = (\Sigma, W', \widehat{w}', A'_1, A'_2, P'_1, P'_2, L', \delta')$, we construct a game graph $G = ((V, E), (V_1, V_2))$ as follows:

- *Player 1 vertices:* $V_1 = (W \times W') \cup \big(\mathsf{Succ}(K) \times \mathsf{Succ}(K')\big)$;
- *Player 2 vertices:* $V_2 = \mathsf{Succ}(K) \times W' \times \{\#, \$\}$;
- *Edges:* The edge set E is specified as the following union: $E = E_1 \cup E_2 \cup E_3 \cup E_4$

$$
\begin{aligned}
E_1 &= \{(\langle w, w'\rangle, \langle \mathsf{Succ}(w, a), w', \#\rangle) \mid w \in W, w' \in W', a \in P_1(w)\} \\
E_2 &= \{(\langle T, w', \#\rangle, \langle T, \mathsf{Succ}(w', a')\rangle) \mid T \in \mathsf{Succ}(K), w' \in W', a' \in P'_1(w')\} \\
E_3 &= \{(\langle T, T'\rangle, \langle T, r', \$\rangle) \mid T \in \mathsf{Succ}(K), T' \in \mathsf{Succ}(K'), r' \in T'\} \\
E_4 &= \{(\langle T, r', \$\rangle, \langle r, r'\rangle) \mid T \in \mathsf{Succ}(K), r' \in W', r \in T\}
\end{aligned}
$$

Let $T = \{\langle w, w'\rangle \mid L(w) \neq L'(w')\}$ be the state pairs that does not match by the labeling function, and let $F = V \setminus T$. The objective for player 1 is to reach T (i.e., Reach(T)) and the objective for player 2 is the safety objective Safe(F). In the following proposition we establish the connection of the winning set for player 2 and $\preceq_{\mathsf{altsim}}$.

▶ Proposition 13. Let $\mathsf{Win}_2 = \{(w, w') \mid w \in W, w' \in W', \langle w, w'\rangle \in W_2(\mathrm{Safe}(F))\}$ Then we have $\mathsf{Win}_2 = \preceq_{\mathsf{altsim}}$.

The algorithmic analysis will be completed in two steps: (1) estimating the size of the game graph; and (2) analyzing the complexity to construct the game graph from the ATSs.

▶ **Lemma 14.** *For the game graph constructed for alternating simulation, we have* $|V_1| + |V_2| \leq O(|W| \cdot |W'| \cdot |A_1| \cdot |A'_1|)$ *and* $|E| \leq O(|W| \cdot |W'| \cdot |A_1| \cdot (|A'_1| \cdot |A'_2| + |A_2|))$.

Game graph construction complexity. We now show that the game graph can be constructed in time linear in the size of the game graph. The data strucutre for the game graph is as follows: we map every vertex in $V_1 \cup V_2$ to a unique integer, and construct the list of edges. Given this data structure for the game graph, the winning sets for reachability and safety objectives can be computed in linear time [3, 12]. We now present the details of the construction of the game graph data structure.

Basic requirements. We start with some basic facts. For two sets A and B, if we have two bijective functions $f_A : A \leftrightarrow \{0, \ldots, |A| - 1\}$ and $f_B : B \leftrightarrow \{0, \ldots, |B| - 1\}$, then we can assign a unique integer to elements of $A \times B$ in time $O(|A| \cdot |B|)$. Since it is easy to construct bijective functions for W and W', we need to construct such bijective functions for $\mathsf{Succ}(K)$ and $\mathsf{Succ}(K')$ to ensure that every vertex has a unique integer. We will present data structure that would achieve the following: (i) construct bijective function $f_K : \mathsf{Succ}(K) \leftrightarrow \{0, \ldots, |\mathsf{Succ}(K)| - 1\}$; (ii) construct function $h_K : W \times A_1 \to \{0, \ldots, |\mathsf{Succ}(K)| - 1\}$ such that for all $w \in W$ and $a \in P_1(w)$ we have $h_K((w, a)) = f_K(\mathsf{Succ}(w, a))$, i.e., it gives the unique number for the successor set of w and action a; (iii) construct function $g_K : \{0, 1, \ldots, |\mathsf{Succ}(K)| - 1\} \to 2^W$ such that for all $T \in \mathsf{Succ}(K)$ we have $g_K(f_K(T))$ is the list of states in T. We will construct the same for K', and also ensure that for all T we compute $g_K(f_K(T))$ in time proportional to the size of T. We first argue how the above functions are sufficient to construct every edge in constant time: (i) edges in E_1 can be constructed by considering state pairs $\langle w, w'\rangle$, actions $a \in P_1(w)$, and with the function $h_K((w, a))$ we add

the required edge, and the result for edges E_2 is similar with the function $h_{K'}$; (ii) edges in E_3 and E_4 are generated using the function g_K that gives the list of states for $g_K(f_K(T))$ in time proportional to the size of T. Hence every edge can be generated in constant time, given the functions, and it follows that with the above functions the game construction is achieved in linear time. We now present the data structure to support the above functions.

Binary tree data structure. Observe that $\mathsf{Succ}(K)$ is a set such that each element is a successor set (i.e., elements are set of states). Without efficient data structure the requirements for the functions f_K, h_K, and g_K cannot be achieved. The data structure we use is a *binary tree data structure*. We assume that states in W are uniquely numbered from 1 to $|W|$. Consider a binary tree, such that every leaf has depth $|W|$, i.e., the length of the path from root to a leaf is $|W|$. Each path from the root to a leaf represents a set — every path consists of a $|W|$ length sequence of *left* and *right* choices. Consider a path π in the binary tree, and the path π represents a subset W_π of W as follows: if the i-th step of π is *left*, then $w_i \notin W_\pi$, if the i-th step is *right*, then $w_i \in W_\pi$. Thus, $\mathsf{Succ}(K)$ is the collection of all sets represented by paths (from root to leaves) in this tree. We have several steps and we describe them below.

1. *Creation of binary tree.* The binary tree BT is created as follows. Initially the tree BT is empty. For all $w \in W$ and all $a \in P_1(w)$ we generate the set $\mathsf{Succ}((w,a))$ as a Boolean array Ar of length $|W|$ such that $\mathsf{Ar}[i] = 1$ if $w_i \in \mathsf{Succ}(w,a)$ and 0 otherwise. We use the array Ar to add the set $\mathsf{Succ}((w,a))$ to BT as follows: we proceed from the root, if $\mathsf{Ar}[0] = 0$ we add left edge, else the right edge, and proceed with $\mathsf{Ar}[1]$ and so on. For every $w \in W$ and $a \in P_1(w)$, the array Ar is generated by going over actions in $P_2(w)$, and the addition of the set $\mathsf{Succ}(w,a)$ to the tree is achieved in $O(|W|)$ time. The initialization of array Ar also requires time $O(|W|)$. Hence the total time required is $O(|W| \cdot |A_1| \cdot (|W| + |A_2|))$. The tree has at most $|W| \cdot |A_1|$ leaves and hence the size of the tree is $O(|W|^2 \cdot |A_1|)$.
2. *The functions f_K, g_K and h_K.* Let Lf denote the leaves of the tree BT, and note that every leaf represents an element of $\mathsf{Succ}(K)$. We do a DFT (depth-first traversal) of the tree BT and assign every leaf the number according to the order of leaves in which it appears in the DFT. Hence the function f_K is constructed in time $O(|W|^2 \cdot |A_1|)$. Moreover, when we construct the function f_K, we create an array GAr of lists for the function g_K. If a leaf is assigned number i by f_K, we go from the leaf to the root and find the set $T \in \mathsf{Succ}(K)$ that the leaf represents and $\mathsf{GAr}[i]$ is the list of states in T. Hence the construction of g_K takes time at most $O(|W| \cdot |A_1| \cdot |W|)$. The function h_K is stored as a two-dimensional array of integers with rows indexed by numbers from 0 to $|W| - 1$, and columns by numbers 0 to $|A_1| - 1$. For a state w and action a, we generate the Boolean array Ar, and use the array Ar to traverse BT, obtain the leaf for $\mathsf{Succ}((w,a))$, and assign $h_K((w,a)) = f_K(\mathsf{Succ}(w,a))$. It follows that h_K is generated in time $O(|W| \cdot |A_1| \cdot (|W| + |A_2|))$.

From the above graph construction, Proposition 13, Lemma 14, and the linear time algorithms to solve games with reachability and safety objectives we have the following result for computing alternating simulation.

▶ **Theorem 15.** *The relation $\preceq_{\mathsf{altsim}}$ can be computed in time $O(|W| \cdot |W'| \cdot |A_1| \cdot (|A_1'| \cdot |A_2'| + |A_2|) + |W|^2 \cdot |A_1| + |W'|^2 \cdot |A_1'|)$ for two ATSs K and K'. The relation $\preceq_{\mathsf{altsim}}$ can be computed in time $O(|W| \cdot |R'| + |W'| \cdot |R|)$ for two TSs K and K'.*

The result for the special case of TSs is obtained by noticing that for TSs we have both $|V|$ and $|E|$ at most $|W| \cdot |R'| + |W'| \cdot |R|$ (see [4] for details), and our algorithm matches the

complexity of the best known algorithm of [10] for simulation for transition systems. Let us denote by $n = |W|$ and $n' = |W'|$ the size of the state spaces, and by $m = |W| \cdot |A_1| \cdot |A_2|$ and $m' = |W'| \cdot |A'_1| \cdot |A'_2|$ the size of the transition relations. Then the basic algorithm requires $O(n \cdot n' \cdot m \cdot m')$ time, whereas our algorithm requires at most $O(m \cdot m' + n \cdot m + n' \cdot m')$ time, and when $n = n'$ and $m = m'$, then the basic algorithm requires $O((n \cdot m)^2)$ time and our algorithm takes $O(m^2)$ time.

4.2 Iterative Algorithm

In this section we will present an iterative algorithm for alternating simulation. For our algorithm we will first present a new and alternative characterization of alternating simulation through successor set simulation.

▶ **Definition 16** (Successor set simulation). Given two ATSs $K = (\Sigma, W, \widehat{w}, A_1, A_2, P_1, P_2, L, \delta)$ and $K' = (\Sigma, W', \widehat{w}', A'_1, A'_2, P'_1, P'_2, L', \delta')$, a relation $\approx \subseteq W \times W'$ is a *successor set simulation* from K to K', if there exists a companion relation $\approx^S \subseteq \mathsf{Succ}(K') \times \mathsf{Succ}(K)$, such that the following conditions hold:

- for all $(w, w') \in \approx$ we have $L(w) = L'(w')$;
- if $(w, w') \in \approx$, then for all actions $a \in P_1(w)$, there exists an action $a' \in P'_1(w')$ such that $(\mathsf{Succ}(w', a'), \mathsf{Succ}(w, a)) \in \approx^S$; and
- if $(T', T) \in \approx^S$, then for all $r' \in T'$, there exists $r \in T$ such that $(r, r') \in \approx$.

We denote by $\approx^*$ the maximum successor set simulation.

The following lemma shows that successor set simulation and alternating simulation coincide. We present the iterative algorithm to compute the maximum successor set simulation $\approx^*$.

▶ **Lemma 17.** *The following assertions hold: (1) Every successor set simulation is an alternating simulation, and every alternating simulation is a successor set simulation. (2) We have* $\approx^* = \preceq_{\mathsf{altsim}}$.

We will now present our iterative algorithm to compute $\approx^*$, and we will denote by $\approx^S$ the witness companion relation of $\approx^*$. Our algorithm will use the following graph construction: Given an ATS K, we will construct the graph $G_K = (V_K, E_K)$ as follows: (1) $V_K = W \cup \mathsf{Succ}(K)$, where W is the set of states; and (2) $E_K = \{(w, \mathsf{Succ}(w, a)) \mid w \in W \wedge a \in P_1(w)\} \cup \{(T, r) \mid T \in \mathsf{Succ}(K) \wedge r \in T\}$. The graph G_K can be constructed in time $O(|W|^2 \cdot |A_1|)$ using the binary tree data structure described earlier. Our algorithm will use the standard notation of Pre and Post: given a graph $G = (V, E)$, for a set U of states, $\mathsf{Post}(U) = \{v \mid \exists u \in U, (u, v) \in E\}$ is the set of successor states of U, and similarly, $\mathsf{Pre}(U) = \{v \mid \exists u \in U, (v, u) \in E\}$ is the set of predecessor states. If $U = \{q\}$ is singleton, we will write $\mathsf{Post}(q)$ instead of $\mathsf{Post}(\{q\})$. Note that in the graph G_K for the state $T \in \mathsf{Succ}(K)$ we have $\mathsf{Post}(T) = \{q \mid q \in T\} = T$. Given ATSs K and K' our algorithm will work simultaneously on the graphs G_K and $G_{K'}$ using three data structures, namely, sim, count and remove for the relation $\approx^*$ (resp. sim^S, count^S and remove^S for the companion relation $\approx^S$). The data structures are as follows: (1) Intuitively sim will be an overapproximation of $\approx^*$, and will be maintained as a two-dimensional Boolean array so that whenever the i, j-th entry is false, then we have a witness that the j-th state w'_j of K' does not simulate the i-th state w_i of K (similary we have sim^S over $\mathsf{Succ}(K')$ and $\mathsf{Succ}(K)$ for the relation $\approx^S$). (2) The data structure count is two-dimensional array, such that for a state $w' \in W'$ and $T \in \mathsf{Succ}(K)$ we have $\mathsf{count}(w', T)$ is the number of elements in the intersection of the successor states of w' and the set of all states that T simulates according

to sim^S; and we also have similar array count^S for T, w' elements. (3) Finally, the data structure remove is a list of sets, where for every $w' \in W'$ we have $\mathsf{remove}(w')$ is a set such that every element of the set belongs to $\mathsf{Succ}(K)$. Similarly for every $T \in \mathsf{Succ}(K)$ we have $\mathsf{remove}^S(T)$ is a set of states. Intuitively the interpretation of remove data structure will be as follows: if $T \in \mathsf{Succ}(K)$ belongs to $\mathsf{remove}(w')$, then no element w of T is simulated by w'. Our algorithm will always maintain sim (resp. sim^S) as overapproximation of $\approx^*$ (resp. $\approx^S$), and will iteratively prune them. Our algorithm is iterative and we denote by $\mathsf{prevsim}$ (resp. $\mathsf{prevsim}^S$) the sim (resp. sim^S) of the previous iteration. To give an intuitive idea of the invariants maintained by the algorithm (Algorithm 1) let us denote by $\mathsf{sim}(w)$ the set of w' such that $\mathsf{sim}(w, w')$ is true, and let us denote by $\mathsf{invsim}(w')$ the inverse of $\mathsf{sim}(w')$, i.e., the set of states w such that (w, w')-th element of sim is true (similar notation for $\mathsf{invprevsim}(w'), \mathsf{invsim}^S(T)$ and $\mathsf{invprevsim}^S(T)$). The algorithm will ensure the following invariants at different steps:

1. For $w \in W, w' \in W'$ and $T \in \mathsf{Succ}(K), T' \in \mathsf{Succ}(K')$,
   a. if $\mathsf{sim}(w, w')$ is false, then $(w, w') \notin \approx^*$;
   b. similarly, if $\mathsf{sim}^S(T', T)$ is false, then $(T', T) \notin \approx^S$.
2. For $w' \in W'$ and $T \in \mathsf{Succ}(K)$,
   a. $\mathsf{count}(w', T) = |\mathsf{Post}(w') \cap \mathsf{invsim}^S(T)|$; and
   b. $\mathsf{count}(T, w') = |\mathsf{Post}(T) \cap \mathsf{invsim}(w')| = |T \cap \mathsf{invsim}(w')|$
3. For $w' \in W'$ and $T \in \mathsf{Succ}(K)$,
   a. $\mathsf{remove}(w') = \mathsf{Pre}(\mathsf{invprevsim}(w')) \setminus \mathsf{Pre}(\mathsf{invsim}(w'))$
   b. $\mathsf{remove}(T) = \mathsf{Pre}(\mathsf{invprevsim}^S(T)) \setminus \mathsf{Pre}(\mathsf{invsim}^S(T))$.

The algorithm has two phases: the initialization phase, where the data structures are initialized; and then a while loop. The while loop consists of two parts: one is pruning of sim and the other is the pruning of sim^S and both the pruning steps are similar. The initialization phase initializes the data structures and is described in Steps 1, 2, and 3 of Algorithm 1. Then the algorithm calls the two pruning steps in a while loop. The condition of the while loop checks whether $\mathsf{prevsim}$ and sim are the same, and it is done in constant time by simply checking whether remove is empty. We now describe one of the pruning procedures and the other is similar. The pruning step is similar to the pruning step of the algorithm of [10] for simulation on transition systems. We describe the pruning procedure PruneSimStrSucc. For every state $w' \in W'$ such that the set $\mathsf{remove}(w')$ is non-empty, we run a for loop. In the for loop we first obtain the predecessors T' of w' in $G_{K'}$ (each predecessor belongs to $\mathsf{Succ}(K')$) and an element T from $\mathsf{remove}(w')$. If $\mathsf{sim}^S(T', T)$ is true, then we do the following steps: (i) We set $\mathsf{sim}^S(T', T)$ to false, because we know that there does not exist any element $w \in T$ such that w' simulates w. (ii) Then for all s' that are predecessors of T' in $G_{K'}$ we decrement $\mathsf{count}(s', T)$, and if the count is zero, then we add s' to the remove set of T. Finally we set the remove set of w' to $\emptyset$. The description of PruneSimStr to prune sim is similar.

Correctness. Our correctness proof will be in two steps. First we will show that invariant 1 (both about sim and sim^S) and invariant 2 (both about count and count^S) are true at the beginning of step 4.1. The invariant 3.(a) (on remove) is true after the procedure call PruneSimStr (step 4.4) and invariant 3.(b) (on remove^S) is true after the procedure call PruneSimStrSucc (step 4.3). We will then argue that these invariants ensure correctness of the algorithm.

Maintaining invariants. We first consider invaraint 1, and focus on invariant 1.(b) (as the other case is symmetric). In procedure PruneSimStrSucc when we set $\mathsf{sim}^S(T', T)$ to false, we need to show that $(T', T) \notin \approx^S$. The argument is as follows: when we set $\mathsf{sim}^S(T', T)$

Algorithm 1 Iterative Algorithm

Input: $K = (\Sigma, W, \widehat{w}, A_1, A_2, P_1, P_2, L, \delta)$, $K' = (\Sigma, W', \widehat{w}', A'_1, A'_2, P'_1, P'_2, L', \delta')$.
Output: $\approx^*$.

1. **Initialize** sim **and** sim^S**:**
   1.1. **for all** $w \in W, w' \in W'$
   $\mathsf{prevsim}(w, w') \leftarrow$ true;
   if $L(w) = L'(w')$, then $\mathsf{sim}(w, w') \leftarrow$ true;
   else $\mathsf{sim}(w, w') \leftarrow$ false;
   1.2. **for all** $T \in \mathsf{Succ}(K)$ and $T' \in \mathsf{Succ}(K')$
   $\mathsf{prevsim}^S(T', T) = \mathsf{sim}^S(T', T) \leftarrow$ true;
2. **Initialize** count **and** count^S**:**
   2.1. **for all** $w' \in W'$ and $T \in \mathsf{Succ}(K)$
   $\mathsf{count}(w', T) \leftarrow |\mathsf{Post}(w') \cap \mathsf{invsim}^S(T)| = |\mathsf{Post}(w')|$;
   $\mathsf{count}^S(T, w') \leftarrow |\mathsf{Post}(T) \cap \mathsf{invsim}(w')|$;
3. **Initialize** remove **and** remove^S**:**
   3.1. **for all** $w' \in W'$
   $\mathsf{remove}(w') \leftarrow \mathsf{Succ}(K) \setminus \mathsf{Pre}(\mathsf{invsim}(w'))$;
   3.2. **for all** $T \in \mathsf{Succ}(K)$
   $\mathsf{remove}^S(T) \leftarrow \emptyset$;

Pruning while loop:

4. **while** $\mathsf{prevsim} \neq \mathsf{sim}$
   4.1 $\mathsf{prevsim} \leftarrow \mathsf{sim}$;
   4.2 $\mathsf{prevsim}^S \leftarrow \mathsf{sim}^S$;
   4.3 **Procedure** PruneSimStrSucc;
   4.4 **Procedure** PruneSimStr;
5. **return** $\{(w, w') \in W \times W' \mid \mathsf{sim}(w, w')$ is true$\}$;

to false, we know that since $T \in \mathsf{remove}(w')$ we have $\mathsf{count}^S(T, w') = 0$ (i.e., $\mathsf{Post}(T) \cap \mathsf{invsim}(w') = \emptyset$). This implies that for every $w \in T$ we have that w' does not simulate w. Also note that since count^S is never incremented, if it reaches zero, it remains zero. This proves the correctness of invariant 1.(b) (and similar argument holds for invariant 1.(a)). The correctness for invariant 2.(a) and 2.(b) is as follows: whenever we decrement $\mathsf{count}(s', T)$ we have set $\mathsf{sim}^S(T', T)$ to false, and T' was earlier both in $\mathsf{Post}(s')$ as well as in $\mathsf{invsim}^S(T)$, and is now removed from $\mathsf{invsim}^S(T)$. Hence from the set $\mathsf{Post}(s') \cap \mathsf{invsim}^S(T)$ we remove the element T' and its cardinality decreases by 1. This establishes correctness of invariant 2.(a) (and invariant 2.(b) is similar). Finally we consider invariant 3.(a): when we add s' to $\mathsf{remove}^S(T)$, then we know that $\mathsf{count}(s', T)$ was decremented to zero, which means T' belongs to $\mathsf{invprevsim}^S(T)$, but not to $\mathsf{invsim}^S(T)$. Thus s' belongs to $\mathsf{Pre}(\mathsf{invprevsim}^S(T))$ (since s' belongs to $\mathsf{Pre}(T')$), but not to $\mathsf{Pre}(\mathsf{invsim}^S(T))$. This shows that s' belongs to $\mathsf{remove}^S(T)$, and establishes correctness of the desired invariant (argument for invariant 3.(b) is similar).

Invariants to correctness. The initialization part ensures that sim is an overapproximation of $\approx^*$ and it follows from invariant 1 that the output is an overapproximation of $\approx^*$. Similarly we also have that sim^S in the end is an overapproximation of $\approx^S$. To complete the correctness proof, let sim and sim^S be the result when the while loop iterations end. We will now show that sim and sim^S are witnesses to satisfy successor set simulation. We know that when

Algorithm 2 Procedure PruneSimStrSucc

1. **forall** $w' \in W'$ such that $\mathsf{remove}(w') \neq \emptyset$
   1.1. **forall** $T' \in \mathsf{Pre}(w')$ and $T \in \mathsf{remove}(w')$
      1.1.1 **if** $(\mathsf{sim}^S(T', T))$
         $\mathsf{sim}^S(T', T) \leftarrow$ false;
         1.1.1.A. **forall** $(s' \in \mathsf{Pre}(T'))$
            $\mathsf{count}(s', T) \leftarrow \mathsf{count}(s', T) - 1$;
            if $(\mathsf{count}(s', T) = 0\ )$
               $\mathsf{remove}^S(T) \leftarrow \mathsf{remove}^S(T) \cup \{s'\}$;
   1.2. $\mathsf{remove}(w') \leftarrow \emptyset$;

Algorithm 3 Procedure PruneSimStr

1. **forall** $T \in \mathsf{Succ}(K)$ such that $\mathsf{remove}^S(T) \neq \emptyset$
   1.1. **forall** $w \in \mathsf{Pre}(T)$ and $w' \in \mathsf{remove}^S(T)$
      1.1.1 **if** $(\mathsf{sim}(w, w'))$
         $\mathsf{sim}(w, w') \leftarrow$ false;
         1.1.1.A. **forall** $(D \in \mathsf{Pre}(w))$
            $\mathsf{count}^S(D, w') \leftarrow \mathsf{count}^S(D, w') - 1$;
            if $(\mathsf{count}^S(D, w') = 0\ )$
               $\mathsf{remove}(w') \leftarrow \mathsf{remove}(w') \cup \{D\}$;
   1.2. $\mathsf{remove}^S(T) \leftarrow \emptyset$;

the algorithm terminates, $\mathsf{remove}(w') = \emptyset$ for every $w' \in W'$, and $\mathsf{remove}^S(T) = \emptyset$ for every $T \in \mathsf{Succ}(K)$ (this follows since $\mathsf{sim} = \mathsf{prevsim}$). To show that sim and sim^S are witness to satisfy successor set simulation, we need to show the following two properties: (i) If $\mathsf{sim}(w, w')$ is true, then for every $a \in P_1(w)$, there exists $a' \in P_1'(w')$ such that $\mathsf{sim}^S(\mathsf{Succ}(w', a'), \mathsf{Succ}(w, a))$ is true. (ii) If $\mathsf{sim}^S(T', T)$ is true, then for every $s' \in T'$, there exists $s \in T$ such that $\mathsf{sim}(s, s')$ is true. The property (i) holds because for every $a \in P_1(w)$, we have that $\mathsf{count}(w', T) > 0$, where $T = \mathsf{Succ}(w, a)$, (because otherwise, w' would have been inserted in $\mathsf{remove}(T)$, but since $\mathsf{remove}(T)$ is empty, consequently $\mathsf{sim}(w, w')$ must have been made false). Hence we have that $\mathsf{Post}(w') \cap \mathsf{invsim}^S(T)$ is non-empty and hence there exists $T' \in \mathsf{Post}(w')$ such that $\mathsf{sim}^S(T', T)$ is true. Similar argument works for (ii). Thus we have established that sim is both an overapproximation of $\approxeq^*$ and also a witness successor set relation. Since $\approxeq^*$ is the maximum successor set relation, it follows that Algorithm 1 correctly computes $\approxeq^* = \preceq_{\mathsf{altsim}}$ ($\approxeq^* = \preceq_{\mathsf{altsim}}$ by Lemma 17).

Space complexity. We now argue that the space complexity of the iterative algorithm is superior as compared to the game based algorithm. We first show that the space taken by Algorithm 1 is $O(|W|^2 \cdot |A_1| + |W'|^2 \cdot |A_1'| + |W| \cdot |W'| \cdot |A_1| \cdot |A_1'|)$. For the iterative algorithm, the space requirements are, (i) sim and sim^S require at most $O(|W| \cdot |W'|)$ and $O(|W| \cdot |W'| \cdot |A_1| \cdot |A_1'|)$ space, respectively; (ii) count and count^S require at most $O(|W| \cdot |W'| \cdot |A_1|)$ space each; (iii) remove and remove^S maintained as an array of sets require at most $O(|W| \cdot |W'| \cdot |A_1|)$, space each. Also, for the construction of graphs G_K and $G_{K'}$ using the binary tree data structure as described earlier, the space required is at most $O(|W|^2 \cdot |A_1|)$ and $O(|W'|^2 \cdot |A_1'|)$, respectively. As compared to the space requirement of

the iterative algorithm, the game based algorithm requires to store the entire game graph and requires at least $O(|W|\cdot|W'|\cdot|A_1|\cdot|A'_1|\cdot|A'_2|)$ space (to store edges in E_3) as well as space $O(|W|^2\cdot|A_1|+|W'|^2\cdot|A'_1|)$ for the binary tree data structure. The iterative algorithm can be viewed as an efficient simultaneous pruning algorithm that does not explicitly construct the game graph (and thus saves at least a factor of $|A'_2|$ in terms of space). We now show that the iterative algorithm along with being space efficient matches the time complexity of the game based algorithm.

Time complexity. The data structures for sim (also sim^S) and count (also count^S) are as described earlier. We store remove and remove^S as a list of lists (i.e., it is a list of sets, and sets are stored as lists). It is easy to verify that all the non-loop operations take unit cost, and thus for the time complexity, we need to estimate the number of times the different loops could run. Also the analysis of the initialization steps are straight forward, and we present the analysis of the loops below: (1) The **while** loop (Step 4) of Algorithm 1 can run for at most $|W|\cdot|W'|$ iterations because in every iteration (except the last iteration) at least one entry of sim changes from true to false (otherwise the iteration stops), and sim has $|W|\cdot|W'|$-entries. (2) The **forall** loop (Step 1) in Algorithm 2 can overall run for at most $|W'|\cdot|W|\cdot|A_1|$ iterations. This is because elements of $\mathsf{remove}(w')$ are from $\mathsf{Succ}(K)$ and elements T from $\mathsf{Succ}(K)$ are included in $\mathsf{remove}(w')$ at most once (when $\mathsf{count}^S(T,w')$ is set to zero, and once $\mathsf{count}^S(T,w')$ is set to zero, it remains zero). Thus $\mathsf{remove}(w')$ can be non-empty at most $|\mathsf{Succ}(K)|$ times, and hence the loop runs at most $|W|\cdot|A_1|$ times for states $w'\in W'$. (3) The **forall** loop (Step 1.1) in Algorithm 2 can overall run for at most $|W'|\cdot|A'_1|\cdot|A'_2|\cdot|W|\cdot|A_1|$ iterations. The reasoning is as follows: for every edge $(T',w')\in G_{K'}$ and $T\in\mathsf{Succ}(K)$ the loop runs at most once (since every T is included in $\mathsf{remove}(w')$ at most once). Hence the number of times the loop runs is at most the number of edges in $G_{K'}$ (at most $|W'|\cdot|A'_1|\cdot|A'_2|$) times the number of elements in $\mathsf{Succ}(K)$ (at most $|W|\cdot|A_1|$). Thus overall the number of iterations of Step 1.1 of Algorithm 2 is at most $|W'|\cdot|A'_1|\cdot|W|\cdot|A_1|$. (4) The **forall** loop (Step 1.1.1.A) in Algorithm 2 can overall run for at most $|W'|\cdot|A'_1|\cdot|A'_2|\cdot|W|\cdot|A_1|$ iterations because every edge (s',T') in $G_{K'}$ would be iterated at most once for every $T\in\mathsf{Succ}(K)$ (as for every T,T' we set $\mathsf{sim}^S(T,T')$ false at most once, and the loop gets executed when such an entry is set to false). The analysis of the following items (5), (6), and (7), are similar to (2), (3), and (4), respectively. (5) The **forall** loop (Step 1) in Algorithm 3 can overall run for at most $|W|\cdot|A_1|\cdot|W'|$ iterations, because $\mathsf{remove}^S(T)$ can be non-empty at most $|W'|$ times (i.e., the number of different T is at most $|\mathsf{Succ}(K)|=|W|\cdot|A_1|$). (6) The **forall** loop (Step 1.1) in Algorithm 3 can overall run for at most $|W|\cdot|A_1|\cdot|A_2|\cdot|W'|$ iterations because every edge (w,T) in G_K can be iterated over at most once for every w' (the number of edges in G_K is $|W|\cdot|A_1|\cdot|A_2|$ and number of states w' is at most $|W'|$). (7) The **forall** loop (Step 1.1.1.A) in Algorithm 3 can overall run for at most $|W|\cdot|A_1|\cdot|A_2|\cdot|W'|$ iterations because every edge (w,D) in G_K would be iterated over at most once for every $w'\in W'$. Adding the above terms, we get that the total time complexity is $O\big(|W|\cdot|W'|\cdot|A_1|\cdot(|A'_1|\cdot|A'_2|+|A_2|)\big)$, i.e., the time complexity matches the time complexity of the game reduction based algorithm. We also remark that for transition systems (TSs), the procedure PruneSimStrSucc coincides with PruneSimStr and our algorithm simplifies to the algorithm of [10], and thus matches the complexity of computing simulation for TSs.

▶ **Theorem 18.** *Algorithm 1 correctly computes* $\preceq_{\mathsf{altsim}}$ *in time* $O\big(|W|\cdot|W'|\cdot|A_1|\cdot(|A'_1|\cdot|A'_2|+|A_2|)+|W|^2\cdot|A_1|+|W'|^2\cdot|A'_1|\big)$.

5 Conclusion

In this work we presented faster algorithms for alternating simulation and alternating fair simulation which are core algorithmic problems in analysis of composite and open reactive systems, as well as state space reduction for graph games (that has deep connection with automata theory and logic). Moreover, our algorithms are obtained as efficient reductions to graph games with reachability and parity objectives with three priorities, and efficient implementations exist for all these problems (for example, see [14] for implementation of games with reachability and parity objectives, and [6] for specialized implementation of games with parity objectives with three priorities).

Acknowledgements. The research was supported by Austrian Science Fund (FWF) Grant No P 23499-N23 on Modern Graph Algorithmic Techniques in Formal Verification, FWF NFN Grant No S11407-N23 (RiSE), ERC Start grant (279307: Graph Games), and Microsoft faculty fellows award.

References

1 R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *JACM*, 49:672–713, 2002.
2 R. Alur, T.A. Henzinger, O. Kupferman, and M.Y. Vardi. Alternating refinement relations. In *CONCUR'98*, LNCS 1466, pages 163–178. Springer, 1998.
3 C. Beeri. On the membership problem for functional and multivalued dependencies in relational databases. *ACM Trans. on Database Systems*, 5:241–259, 1980.
4 K. Chatterjee, S. Chaubal, and P Kamath. Faster algorithms for alternating refinement relations. *CoRR*, abs/1201.4449, 2012.
5 E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
6 L. de Alfaro and M. Faella. An accelerated algorithm for 3-color parity games with an application to timed games. In *CAV*, pages 108–120, 2007.
7 E.A. Emerson and C. Jutla. Tree automata, mu-calculus and determinacy. In *FOCS'91*, pages 368–377. IEEE, 1991.
8 Y. Gurevich and L. Harrington. Trees, automata, and games. In *STOC'82*, pages 60–65. ACM Press, 1982.
9 J. Y. Halpern and R. Fagin. Modeling knowledge and action in distributed systems. *Distributed Computing*, 3:159–179, 1989.
10 M.R. Henzinger, T.A. Henzinger, and P.W. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, pages 453–462. IEEE, 1995.
11 T.A. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. *I & C.*, 173:64–81, 2002.
12 N. Immerman. Number of quantifiers is better than number of tape cells. *JCSS*, 22:384–406, 1981.
13 M. Jurdzinski. Small progress measures for solving parity games. In *STACS'00*, pages 290–301. LNCS 1770, Springer, 2000.
14 M. Lange and O. Friedmann. The pgsolver collection of parity game solvers. 2009.
15 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.
16 L.S. Shapley. Stochastic games. *Proc. Nat. Acad. Sci. USA*, 39:1095–1100, 1953.
17 W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, Beyond Words, chapter 7, pages 389–455. Springer, 1997.