Making Weighted Containment Feasible: A Heuristic Based on Simulation and Abstraction

Guy Avni and Orna Kupferman

School of Computer Science and Engineering, Hebrew University, Israel

Abstract. Weighted automata map input words to real numbers and are useful in reasoning about quantitative systems and specifications. The containment problem for weighted automata asks, given two weighted automata $\mathcal{A}$ and $\mathcal{B}$, whether for all words w, the value that $\mathcal{A}$ assigns to w is less than or equal to the value $\mathcal{B}$ assigns to w. The problem is of great practical interest, yet is known to be undecidable. Efforts to approximate weighted containment by weighted variants of the simulation pre-order still have to cope with large state spaces. One of the leading approaches for coping with large state spaces is abstraction. We introduce an abstraction-refinement paradigm for weighted automata and show that it nicely combines with weighted simulation, giving rise to a feasible approach for the containment problem. The weighted-simulation pre-order we define is based on a quantitative two-player game, and the technical challenge in the setting origins from the fact the values that the automata assign to words are unbounded. The abstraction-refinement paradigm is based on under- and over-approximation of the automata, where approximation, and hence also the refinement steps, refer not only to the languages of the automata but also to the values they assign to words.

1 Introduction

Traditional automata accept or reject their input, and are therefore Boolean. A weighted finite automaton (WFA, for short) has real-valued weights on its transitions and it maps each word to a real value. Applications of weighted automata include formal verification, where they are used for the verification of quantitative properties [10,11,17,25,32], as well as text, speech, and image processing, where the weights of the automaton are used in order to account for the variability of the data and to rank alternative hypotheses [15,30].

Technically, each transition in a WFA is associated with a weight, the value of a run is the sum of the weights of the transitions traversed along the run, and the value of a word is the value of the maximal run on it.[1] The rich structure of weighted automata makes them intriguing mathematical objects. Fundamental problems that have been solved decades ago for Boolean automata are still open or known to be undecidable in the weighted setting [29]. For example, while in the Boolean setting, nondeterminism does not add to the expressive power of the automata, not all weighted automata can be

[1] The above semantics, which we are going to follow in the paper, is a special case of the general setting, were each weighted automaton is defined with respect to an algebraic semiring.

determinized, and the problem of deciding whether a given nondeterministic weighted automaton can be determinized is still open, in the sense we do not even know whether it is decidable.

A problem of great interest in the context of automata is the *containment* problem. In the Boolean setting, the containment problem asks, given two automata $\mathcal{A}$ and $\mathcal{B}$, whether all the words in Σ^* that are accepted by $\mathcal{A}$ are also accepted by $\mathcal{B}$. In the weighted setting, the "goal" of words is not just to get accepted, but also to do so with a maximal value. Accordingly, the containment problem for WFAs asks, given two WFAs $\mathcal{A}$ and $\mathcal{B}$, whether every word accepted by $\mathcal{A}$ is also accepted by $\mathcal{B}$, and its value in $\mathcal{A}$ is less than or equal to its value in $\mathcal{B}$. We then say that $\mathcal{B}$ contains $\mathcal{A}$, denoted $\mathcal{A} \subseteq \mathcal{B}$. In the Boolean setting, the containment problem is PSPACE-complete [31]. In the weighted setting, the problem is in general undecidable [1,24]. The problem is indeed of great interest: In the automata-theoretic approach to reasoning about systems and their specifications, containment amounts to correctness of systems with respect to their specifications. The same motivation applies for weighted systems, with the specifications being quantitative [10].

Even in the Boolean setting, where the containment problem is decidable, its PSPACE complexity is an obstacle in practice and researchers have suggested two orthogonal methods for coping with it. One is to replace containment by a pre-order that is easier to check, with the leading such pre-order being the *simulation* preorder [28]. Simulation can be checked in polynomial time and symbolically [20,28], and several heuristics for checking containment by variants of simulation have been studied and used in practice [23,26]. A second method, useful also in other paradigms for reasoning about the huge, and possibly infinite, state space of systems is *abstraction* [3,8]. Essentially, in abstraction we hide some of the information about the system. This enables us to reason about systems that are much smaller, yet it gives rise to a 3-valued solution: yes, no, and unknown [5]. In the latter case, the common practice is to refine the abstraction, aiming to add the minimal information that would lead to a definite solution. In particular, in the context of model checking, the method of counterexample guided abstraction-refinement (CEGAR) has proven to be very effective [13].

In this paper we study a combination of the above two methods in the setting of weighted automata. Abstraction frameworks in the 3-valued Boolean semantics are typically based on *modal transition systems* (MTS) [27]. Such systems have two types of transitions: *may* transitions, which over-approximate the transitions of the concrete system, and *must* transitions, which under-approximate them. The over and under approximation refer to the ability of the automaton to take transitions, and hence to its language. In our weighted setting, we combine this with the weights of the transitions: may transitions over-approximate the actual weight and must transitions under-approximate it. This is achieved by defining the weight of may and must transitions according to the maximal and minimal weight, respectively, of the transitions that induce them.

The simulation preorder in the Boolean setting has a game-theoretic characterization. We extend this approach to the weighted setting and define weighted simulation between two WFAs $\mathcal{A}$ and $\mathcal{B}$ by means of a game played between an antagonist, who iteratively generates a word w and an accepting run r of $\mathcal{A}$ on it, and a protagonist, who replies with a run r' of $\mathcal{B}$ on w. The goal of the antagonist is to generate w and r so that

either r' is not accepting, or its value is smaller than the value of r. The goal of the protagonist is to continue the game forever without the antagonist reaching his goal. We say that $\mathcal{A}$ is simulated by $\mathcal{B}$, denoted $\mathcal{A} \leq \mathcal{B}$ iff the protagonist has a winning strategy. The above definition is similar to the definition of quantitative simulation in [12,14], and has the flavor of the *energy games* in [4]. In these works, however, the winning condition in the game refers only to the weight along the traversed edges. This corresponds to the case the WFAs in question are such that all states are accepting. Even richer than our setting are *energy parity games* [9]. Both energy games and parity energy games can be decided in NP $\cap$ co-NP. Our main challenge then is to develop an algorithm that would maintain the simplicity of the algorithm in [4] in the richer setting, which is simpler than the one of parity games. We do this by performing a preprocessing on the arena of the game, one after which we can perform only local changes in the algorithm of [4]. This is not easy, as like in parity energy games a winning strategy in the simulation game need not be memoryless. Our main contribution, however, is not the study of simulation games and their solution – the main ideas here are similar to these in [4,9], but the ability to combine simulation with abstraction and refinement, which we see as our main contribution.

Having defined over- and under-approximations of WFAs and the weighted simulation relation, we suggest the following heuristic for checking whether $\mathcal{A} \subseteq \mathcal{B}$. For a WFA $\mathcal{U}$ and an abstraction function α, let $\mathcal{U}^\alpha_\downarrow$ and $\mathcal{U}^\alpha_\uparrow$ be the weighted under and over approximations of $\mathcal{U}$ according to α. Let α and β be approximation functions for $\mathcal{A}$ and $\mathcal{B}$, respectively. It is not hard to see that if $\mathcal{A}^\alpha_\uparrow \subseteq \mathcal{B}^\beta_\downarrow$, then $\mathcal{A} \subseteq \mathcal{B}$, and that if $\mathcal{A}^\alpha_\downarrow \not\subseteq \mathcal{B}^\beta_\uparrow$, then $\mathcal{A} \not\subseteq \mathcal{B}$. We show that the above is valid not just of containment but also for our weighted-simulation relation. This gives rise to the following heuristics. We start by checking $\mathcal{A}^\alpha_\uparrow \leq \mathcal{B}^\beta_\downarrow$ and $\mathcal{A}^\alpha_\downarrow \not\leq \mathcal{B}^\beta_\uparrow$, for some (typically coarse) initial abstraction functions α and β. As we prove in the paper, if we are lucky and one of them holds, we are done. Otherwise, the winning strategies of the antagonist in the first case and the protagonist in the second case suggest a way to refine α and β, and we repeat the process with the refined abstractions. While refinement in the Boolean case only splits abstract states in order to close the gap between may and must transitions, here we also have refinement steps that tighten the weights along transitions. Note that while repeated refinement can only get us to a solution to the $\mathcal{A} \leq \mathcal{B}$ problem, they also make it more likely that one of our checks returns an answer that would imply a definite solution to the undecidable $\mathcal{A} \subseteq \mathcal{B}$ problem.

Note that our abstraction-refinement procedure combines two games. The first, which corresponds to $\mathcal{A}^\alpha_\uparrow \leq \mathcal{B}^\beta_\downarrow$, approximates the simulation question $\mathcal{A} \leq \mathcal{B}$ from below. The second, which corresponds to $\mathcal{A}^\alpha_\downarrow \leq \mathcal{B}^\beta_\uparrow$, approximates it from above. Such dual approximations have proven useful also in the Boolean setting [6,16,18,21,22], where games combine may and must transitions, and also in settings in which games that are determined are approximated by means other than abstraction. For example, when LTL realizability is done by checking the realizability of approximations of both the specification and its negation [7].

Due to the lack of space, some proofs are only sketched. The full proofs can be found in the full version in the authors' homepages.

2 Weighted Automata and their Abstraction

A nondeterministic finite weighted automaton on finite words (WFA, for short) is a tuple $\mathcal{A} = \langle \Sigma, Q, \Delta, Q_0, \tau, F \rangle$, where Σ is an alphabet, Q is a set of states, $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation, $Q_0 \subseteq Q$ is a set of initial states, $\tau : \Delta \to \mathbb{R}$ is a function that maps each transition to a real value in $\mathbb{R}$, and $F \subseteq Q$ is a set of accepting states. We assume that there are no *dead-end* states in $\mathcal{A}$. That is, for every $q \in Q$ there is a letter $\sigma \in \Sigma$ and state $q' \in Q$ such that $\Delta(q, \sigma, q')$.

A run of $\mathcal{A}$ on a word $u = u_1, \ldots, u_n \in \Sigma^*$ is a sequence of states $r = r_0, r_1, \ldots, r_n$ such that $r_0 \in Q_0$ and for every $0 \leq i < n$ we have $\Delta(r_i, u_{i+1}, r_{i+1})$. The run r is accepting iff $r_n \in F$. The value of the run, denoted $val(r, u)$, is the sum of transitions it traverses. That is, $val(r, u) = \sum_{0 \leq i < n} \tau(\langle r_i, u_{i+1}, r_{i+1} \rangle)$. Since $\mathcal{A}$ is nondeterministic, there can be more than one run on a single word. We define the value that $\mathcal{A}$ assigns to $u \in \Sigma^*$, denoted $val(\mathcal{A}, u)$, as the value of the maximal-valued accepting run of $\mathcal{A}$ on u. That is, $val(\mathcal{A}, u) = max\{val(r, u) : r \text{ is an accepting run of } \mathcal{A} \text{ on } u\}$. As in NFAs, the language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words in Σ^* that $\mathcal{A}$ accepts.

We say that $\mathcal{A}$ is *deterministic* if $|Q_0| = 1$ and for every $q \in Q$ and $\sigma \in \Sigma$, there is at most one state $q' \in Q$ such that $\Delta(q, \sigma, q')$.

An *abstraction function* for a WFA $\mathcal{A}$ is a function $\alpha : Q \to A$, for a set A, which we assume to be smaller than Q. We refer to the members of Q as the *concrete states* and to these of A as the *abstract states*. The function α induces a partition of Q, and we sometimes refer to abstract states as sets of concrete states. In particular, for a concrete state $c \in Q$ and an abstract state $a \in A$, we use the notation $c \in a$ to indicate that $\alpha(c) = a$.

Consider a WFA $\mathcal{A}$ and an abstraction function α. For parameters $\beta \in \{may, must\}$ and $\gamma \in \{max, min\}$, the abstraction of $\mathcal{A}$ according to α, β, and γ is the WFA $\mathcal{A}^{\gamma}_{\beta}[\alpha] = \langle \Sigma, A, \Delta_\beta, A_0, \tau_\gamma, F_\beta \rangle$, where $A_0 = \{\alpha(q_0) : q_0 \in Q_0\}$, and Δ_β, τ_γ, and F_β are defined as follows:

- Consider $a, a' \in A$ and $\sigma \in \Sigma$. We define $\Delta_{must} \subseteq A \times \Sigma \times A$ so that $\Delta_{must}(a, \sigma, a')$ iff for every $c \in a$ there is $c' \in a'$ such that $\Delta(c, \sigma, c')$. We define $\Delta_{may} \subseteq A \times \Sigma \times A$ so that $\Delta_{may}(a, \sigma, a')$ iff there exists $c \in a$ and $c' \in a'$ such that $\Delta(c, \sigma, c')$.
- We define the minimum-value function, denoted τ_{min}, of an abstract transition to be the minimum over the values of concrete transitions that induce it. Formally, for $\langle a, \sigma, a' \rangle \in \Delta_\beta$, we define $\tau_{min}(\langle a, \sigma, a' \rangle) = min\{\tau(\langle c, \sigma, c' \rangle) : c \in a, c' \in a', \text{ and } \Delta(c, \sigma, c')\}$. Similarly, we define the maximal-value function as τ_{max}, with $\tau_{max}((\langle a, \sigma, a' \rangle) = max\{\tau(\langle c, \sigma, c' \rangle) : c \in a, c' \in a', \text{ and } \Delta(c, \sigma, c')\}$.
- We define $F_{may} = \{a \in A : a \cap F \neq \emptyset\}$ and $F_{must} = \{a \in A : a \subseteq F\}$.

Note that without weights, our definition coincides with the standard over- and under-approximations studied in the Boolean case. In the weighted setting, the abstraction approximates, in addition to the transitions, the value that the concrete WFA assigns to words. The two interesting combinations are then the under-approximating WFA $\mathcal{A}^{\alpha}_{\downarrow} = \mathcal{A}^{min}_{must}[\alpha]$ and the over-approximating WFA $\mathcal{A}^{\alpha}_{\uparrow} = \mathcal{A}^{max}_{may}[\alpha]$. When α is not important or clear form the context, we omit it.

We refer to runs of $\mathcal{A}_\downarrow$ as *must-runs*, runs of $\mathcal{A}_\uparrow$ as *may-runs*, and runs of $\mathcal{A}$ as *concrete-runs*. Note that for every must-run $r = r_0, \ldots, r_n$ of $\mathcal{A}_\downarrow$ on some word $u \in \Sigma^*$, there is a matching run $r' = r'_0, \ldots, r'_n$ of $\mathcal{A}$ on u such that, for every $0 \leq i \leq n$, we have $r'_i \in r_i$. Similarly, for every run $r = r_0, \ldots, r_n$ of $\mathcal{A}$ on some word u, the sequence $r' = \alpha(r_0), \ldots, \alpha(r_n)$ is a run of $\mathcal{A}_\uparrow$ on u.

In the Boolean setting, *language containment* refers to words accepted by the automaton. That is, for two NFAs $\mathcal{A}$ and $\mathcal{B}$, we say that $\mathcal{A} \subseteq \mathcal{B}$ iff $L(\mathcal{A}) \subseteq L(\mathcal{B})$. In the weighted setting, language containment is more involved, as we also have a requirement on the values the automata assign to words. For two WFAs, we say that $\mathcal{A} \subseteq \mathcal{B}$ iff $L(\mathcal{A}) \subseteq L(\mathcal{B})$ and for every $w \in L(\mathcal{A})$ we have $val(\mathcal{A}, w) \leq val(\mathcal{B}, w)$.

The *containment problem* gets as input two automata $\mathcal{A}$ and $\mathcal{B}$, and decides whether $\mathcal{A} \subseteq \mathcal{B}$. The problem is known to be PSPACE-complete in the Boolean setting [33] and undecidable in the weighted setting [1,24].

Since, in practice, WFAs are typically very large, we would like to reason on their abstractions. As Theorem 1 below shows, $\mathcal{A}_\downarrow$ and $\mathcal{A}_\uparrow$ under- and over-approximates $\mathcal{A}$, making such a reasoning possible.

Theorem 1. *Consider a WFA $\mathcal{A}$ and an abstraction function α. Then, $\mathcal{A}^\alpha_\downarrow \subseteq \mathcal{A} \subseteq \mathcal{A}^\alpha_\uparrow$.*

Proof: We start by proving that $\mathcal{A}_\downarrow \subseteq \mathcal{A}$. Consider a word $u = u_1, \ldots, u_n \in L(\mathcal{A}_\downarrow)$. We prove that $u \in L(\mathcal{A})$ and $val(\mathcal{A}_\downarrow, u) \leq val(\mathcal{A}, u)$. Let $r = a_0, \ldots, a_n \in A^*$ be an accepting run of $\mathcal{A}_\downarrow$ on u. Since $a_0 \in A_0$, there is a concrete state $c_0 \in (a_0 \cap Q_0)$. Since r is a must-run, there is a concrete run $r' = c_0, \ldots, c_n$ of $\mathcal{A}$ on u such that, for $1 \leq i \leq n$, we have $c_i \in a_i$. Since r is accepting, $r_n \subseteq F$, implying that $c_n \in F$. Thus, r' is accepting and $u \in L(\mathcal{A})$.

It is left to prove that $val(\mathcal{A}_\downarrow, u) \leq val(\mathcal{A}, u)$. We show that for every accepting run r of $\mathcal{A}_\downarrow$ on u and every concrete run r' that corresponds to it, $val(r, u) \leq val(r', u)$. Indeed, by the definition of τ_{min}, we have $val(r, u) = \sum_{0 \leq i \leq n} \tau_{min}(r_i, u_{i+1}, r_{i+1}) \leq \sum_{0 \leq i \leq n} \tau(r_i, u_{i+1}, r_{i+1}) = val(r', u)$, so we are done.

The proof of the second claim is similar and is presented in the full version. □

3 Weighted Simulation

As discussed in Section 1, the pre-order of simulation [28] is used in the Boolean setting as a heuristic for checking containment. In this section we define weighted simulation and show that it enjoys the appealing properties of simulation in the Boolean setting. In Section 4, we show that weighted simulation can be checked by reasoning about abstractions of the WFAs in question.

3.1 Defining the Weighted Simulation Relation

Given two WFAs $\mathcal{A}$ and $\mathcal{B}$, deciding whether $\mathcal{A} \subseteq \mathcal{B}$ can be thought of as a two-player game of one round: Player 1, the Player whose goal it is to show that there is no containment, chooses a word w and a run r_1 of $\mathcal{A}$ on w. Player 2 then replies by choosing a run r_2 of $\mathcal{B}$ on w. Player 1 wins if r_1 is accepting and r_2 is not or if $val(r_1, w) > val(r_2, w)$. While this game clearly captures containment, it does not

lead to interesting insights or algoritmic ideas about checking containment. A useful way to view simulation is as a "step-wise" version of the above game in which in each round the players proceed according to a single transition of the WFAs.

We continue to describe the simulation game formally. A game between Player 1 and Player 2 is a pair $\langle G, \Gamma \rangle$, for an arena G and an objective Γ for Player 1. Consider two WFAs $\mathcal{A}$ and $\mathcal{B}$, where for $\gamma \in \{\mathcal{A}, \mathcal{B}\}$, let $\gamma = \langle \Sigma, Q_\gamma, \Delta_\gamma, q_0^\gamma, F_\gamma, \tau_\gamma \rangle$. For simplicity, we assume that the WFAs are full, in the sense that each state and letter have at least one successor.

The arena of the game that corresponds to $\mathcal{A} \leq \mathcal{B}$ is $G = \langle V, E, v_0, \tau \rangle$. The set V of vertices is partitioned into two disjoint sets: $V_1 = Q_\mathcal{A} \times Q_\mathcal{B}$ are vertices from which Player 1 proceeds, and $V_2 = Q_\mathcal{A} \times \Sigma \times Q_\mathcal{B}$ are vertices from which Player 2 proceeds. The players alternate moves, thus $E \subseteq (V_1 \times V_2) \cup (V_2 \times V_1)$. Each play starts in the initial vertex $v_0 = \langle q_0^\mathcal{A}, q_0^\mathcal{B} \rangle \in V_1$, and $\tau : E \to \mathbb{R}$ is the weight function. We define $E = E_1 \cup E_2$ and τ as follows:

- $E_1 = \{\langle \langle p, q \rangle, \langle p', \sigma, q \rangle \rangle : \langle p, \sigma, p' \rangle \in \Delta_\mathcal{A}$ and $q \in Q_\mathcal{B}\}$.
- $E_2 = \{\langle \langle p, \sigma, q \rangle, \langle p, q' \rangle \rangle : \langle q, \sigma, q' \rangle \in \Delta_\mathcal{B}$ and $p \in Q_\mathcal{A}\}$.
- For $e_1 = \langle \langle p, q \rangle, \langle p', \sigma, q \rangle \rangle \in E_1$, we define $\tau(e_1) = \tau_\mathcal{A}(\langle p, \sigma, p' \rangle)$.
- For $e_2 = \langle \langle p, \sigma, q \rangle, \langle p, q' \rangle \rangle \in E_2$, we define $\tau(e_2) = -\tau_\mathcal{B}(\langle q, \sigma, q' \rangle)$.

Thus, edges in E_1 leave vertices in V_1 and correspond to Player 1 choosing a letter and a transition in $\mathcal{A}$. Edges in E_2 leave vertices in V_2 and correspond to Player 2 choosing a transition in $\mathcal{B}$.

A play of the game is a (possibly infinite) sequence of vertices $\pi = \pi_0, \pi_1, \ldots$, where $\pi_0 = v_0$, and for every $i \geq 0$ we have $E(v_i, v_{i+1})$. Every finite play has a value, denoted $val(\pi)$, which is the sum of the edges that are traversed along it: i.e., $val(\pi) = \sum_{0 \leq i < |\pi|} \tau(\langle \pi_i, \pi_{i+1} \rangle)$. We use $\pi[i : j]$, for $i < j$, to refer to the sub-play $\pi_i, \ldots, \pi_j$.

A strategy for player $i \in \{1, 2\}$ is a function $\rho_i : V^* \cdot V_i \to V$. Let $\mathcal{S}_i$ be the set of all strategies for player i. Two strategies $\rho_1 \in \mathcal{S}_1$ and $\rho_2 \in \mathcal{S}_2$, induce a single play obtained when both players follow their strategies. Formally, the outcome of ρ_1 and ρ_2, denoted $out(\rho_1, \rho_2)$, is the infinite play $\pi = \pi_1, \pi_2, \ldots$, where for every $i \geq 0$, we have $\pi_{2i+1} = \rho_1(\pi[0 : 2i])$ and $\pi_{2i+2} = \rho_2(\pi[0 : 2i + 1])$. We say that a strategy ρ_i is *memoryless* if it depends only on the current vertex. Formally, $\rho_i(u_1 \cdot v) = \rho_i(u_2 \cdot v)$ for all $u_1, u_2 \in V^*$ and $v \in V_i$.

It is left to define the objective of the game. A finite play π is winning for Player 1 if the last vertex of π is in $F_\mathcal{A} \times (Q_\mathcal{B} \setminus F_\mathcal{B})$ or the last vertex of π is in $F_\mathcal{A} \times F_\mathcal{B}$ and $val(\pi) > 0$. The *objective* $\Gamma \subseteq V^\omega$ of Player 1, namely the set of plays that are winning for Player 1 is defined so that an infinite play π is in Γ iff it has a finite prefix that is winning according to the definition above. Note that for an infinite play π, if $\pi \notin \Gamma$, then it is winning for Player 2. Thus, the objective of $Player\ 2$ is $\overline{\Gamma} = V^\omega \setminus \Gamma$. Also note that once the play has a prefix that is winning for Player 1, there is no actual need for the play to continue. A *winning strategy* for Player 1 is a strategy $\rho_1 \in \mathcal{S}_1$ such that for every strategy $\rho_2 \in \mathcal{S}_2$, the play $out(\rho_1, \rho_2)$ is in Γ. A winning strategy for Player 2 is defined symmetrically. We define the simulation relation so that $\mathcal{A} \leq \mathcal{B}$ iff Player 2 has a winning strategy in G.

Theorem 2. *Simulation is strictly stronger than containment: (1) for all WFAs $\mathcal{A}$ and $\mathcal{B}$, if $\mathcal{A} \leq \mathcal{B}$, then $\mathcal{A} \subseteq \mathcal{B}$. (2) There are WFAs $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \subseteq \mathcal{B}$ and $\mathcal{A} \not\leq \mathcal{B}$.*

Proof: We start with the first claim. Recall that $\mathcal{A} \subseteq \mathcal{B}$ iff $L(\mathcal{A}) \subseteq L(\mathcal{B})$ and for every $u \in L(\mathcal{A})$ we have $val(\mathcal{A}, u) \leq val(\mathcal{B}, u)$. We prove that if $\mathcal{A} \not\subseteq \mathcal{B}$ then Player 1 has a winning strategy. Thus, there is no winning strategy for Player 2 and $\mathcal{A} \not\leq \mathcal{B}$.

Assume that $\mathcal{A} \not\subseteq \mathcal{B}$. That is, there exists a word $u \in \Sigma^*$ such that $u \in L(\mathcal{A}) \setminus L(\mathcal{B})$, or $u \in L(\mathcal{A})$ and $val(\mathcal{A}, u) > val(\mathcal{B}, u)$. Consider the strategy $\rho_1 \in \mathcal{S}_1$ in which Player 1 selects the word u and chooses the run r_1 that maximizes the value of u in $\mathcal{A}$. In the full version, we show that for every strategy $\rho_2 \in \mathcal{S}_2$ of Player 2, the play $out(\rho_1, \rho_2)$ is winning for Player 1. Thus, ρ_1 is a winning strategy of Player 1 and we are done.

The proof of the send claim is described in the full version. While the claim easily follows form the analogous claim in the Boolean setting, the example there is such that $\mathcal{A}$ is simulated by $\mathcal{B}$ in the Boolean sense, and the weights of the WFAs are these that wreck the simulation. □

As in the Boolean setting, simulation and containment do coincide in case the simulating automaton is deterministic. Indeed, then, there is only one Player 2 strategy, so the "step-wise nature" of simulation does not play a role.

Theorem 3. *If $\mathcal{B}$ is a DWFA, then $\mathcal{A} \subseteq \mathcal{B}$ iff $\mathcal{A} \leq \mathcal{B}$.*

Another property of simulation that stays valid in the weighted setting is its transitivity.

Theorem 4. *For WFAs $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, if $\mathcal{A} \leq \mathcal{B}$ and $\mathcal{B} \leq \mathcal{C}$, then $\mathcal{A} \leq \mathcal{C}$.*

Unlike the Boolean case, here Player 1 need not have a memoryless strategy, as we demonstrate below. The WFAs we use in the example are used also in [9] in order to show that Player 2 has no memoryless winning strategy in energy parity games.

Example 1. We show a family of WFAs $\mathcal{A}_1, \mathcal{A}_2, \ldots$ and a WFA $\mathcal{B}$ such that for all $n \geq 1$, Player 1 wins the simulation game corresponding to $\mathcal{A}_n$ and $\mathcal{B}$, but he has no memoryless winning strategy. Moreover, a winning strategy for Player 1 needs memory of size $\Omega(m \cdot W)$, where m is the size of $\mathcal{A}_n \times \mathcal{B}_n$ and W is the maximal weight.

Consider the WFAs $\mathcal{A}_n$ and $\mathcal{B}$ in Figure 1. Since $L(\mathcal{B}) = a^*$, then clearly $L(\mathcal{A}_n) \subseteq L(\mathcal{B})$. However, $\mathcal{A}_n \not\subseteq \mathcal{B}$, since for $w = a^n \cdot a^{2Wn+1} a^n$, we have $cost(\mathcal{A}_n, w) = 1 > 0 = cost(\mathcal{B}, w)$.

We claim that in the simulation game (on bottom) that corresponds to the two WFAs, there is a winning Player 1 strategy (i.e., $\mathcal{A}_n \not\leq \mathcal{B}$) and that every such winning strategy for Player 1 requires $\Omega(m \cdot W)$ memory. Indeed, a winning Player 1 strategy must proceed to the state (q_n, s_0) and loop there for at least $2Wn+1$ rounds before returning to the initial state. Thus, a winning strategy must "count" to $2Wn + 1$.

Before we turn to study a solution to the simulation game, observe that the set of winning plays for Player 1 is open since it is defined by prefixes. By the Gale-Stewart theorem [19], every game that satisfies this property is determined, hence we have the following.

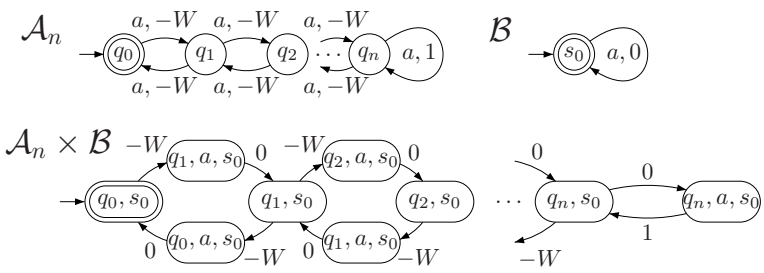

Fig. 1. WFAs $\mathcal{A}_n$ and $\mathcal{B}$ such that $\mathcal{A}_n \not\leq \mathcal{B}$ yet Player 1 does not have a memoryless strategy in the corresponding simulation game.

Theorem 5. *The simulation game is determined. That is, Player 1 or Player 2 has a winning strategy.*

3.2 Solving the Simulation Game

The simulation game stands between the energy games of [4], where the NFAs have no acceptance conditions, and the energy parity games of [9], where the winning condition is richer than the one of WFAs. Both these games are determined and can be decided in NP $\cap$ co-NP. It is thus not surprising that we are going to show that the same holds for our simulation game. Our main challenge is to develop an algorithm that would maintain the simplicity of the algorithm in [4] in the richer setting of WFAs. The setting is indeed richer, and in particular, as in energy parity games, Player 1 need not have a memoryless winning strategy. We do this by performing a preprocessing on the arena of the game, one after which we can perform only local changes in the algorithm of [4]. Our main contribution, however, is not the study of simulation games and their solution – the main ideas here are similar to these in [4,9]. Rather, it is the combination of these ideas in an abstraction-refinement paradigm, to be described in Section 4.

Reducing $\langle G, \Gamma \rangle$ to a simpler game $\langle G', \Gamma' \rangle$ Consider an arena $G = \langle V, E, v_0, \tau \rangle$.

- Let $W_1 \subseteq V$ be the set of vertices from which Player 1 wins the reachability game with objective $F_{\mathcal{A}} \times (Q_{\mathcal{B}} \setminus F_{\mathcal{B}})$. That is, $v \in W_1$ iff Player 1 can force the game that starts in v to a vertex in $F_{\mathcal{A}} \times (Q_{\mathcal{B}} \setminus F_{\mathcal{B}})$.
- Let $W_2 \subseteq V$ be the vertices from which Player 2 wins the safety game with objective $((Q_{\mathcal{A}} \setminus F_{\mathcal{A}}) \times Q_{\mathcal{B}}) \cup V_2$. That is, $v \in W_2$ iff Player 1 can force the game that starts in v to stay in vertices in $((Q_{\mathcal{A}} \setminus F_{\mathcal{A}}) \times Q_{\mathcal{B}}) \cup V_2$.

It is not hard to see that $W_1 \cap W_2 = \emptyset$. We can therefore distinguish between three cases: If $v_0 \in W_1$, then Player 1 wins the game. If $v_0 \in W_2$, Player 2 wins the game. Otherwise, we define a new game, which excludes states from $W_1 \cup W_2$.

We define the new game $\langle G', \Gamma' \rangle$ on the arena $G' = \langle V', E', v_0, \tau' \rangle$. The set V' of vertices are $\{v_{sink}\} \cup (V \setminus (W_1 \cup W_2))$. The set V'_1 of vertices of Player 1 is $V_1 \cap V'$,

and the set V_2' of vertices of Player 2 is $V_2 \cap V'$. We say that a vertex $v \in V'$ is a *dead-end* iff $adj(v) \subseteq (W_1 \cup W_2)$, where $adj(v)$ is defined with respect to E. That is, $adj(v) = \{v' \in V : E(v, v')\}$. The set E' of edges restricts the set E to vertices in V' and includes, in addition, an edge from every dead-end vertex in V' to v_{sink} and an edge from v_{sink} to itself. Recall that we assume that the WFAs on which the simulation game are defined are total, and thus there are no dead-ends in G. Hence, a vertex is a dead-end in G' when all its successors in G are in $W_1 \cup W_2$. Finally, τ' assigns the same value as τ for the edges in E, and assigns 0 to the new edges.

A finite play π is winning for Player 1 in the new game iff the last vertex in π is in $F_{\mathcal{A}} \times F_{\mathcal{B}}$ and $val(\pi) > 0$. As in the original game, Player 2 wins an infinite play iff it does not have a finite prefix that is winning for Player 1. Note that an infinite play π satisfies this condition, i.e., $\pi \in \overline{\Gamma'}$, if either one of two conditions: either π is contained in V and $\pi \in \overline{\Gamma}$, or π has a finite prefix $\pi[0 : i]$ that ends in a dead-end vertex, i.e., for every $k > i$ we have $\pi_k = v_{sink}$, and for every $j \leq i$, we have that $\pi[0 : j]$ is not winning for Player 1.

As we prove in the full version, the characteristics of the vertices in W_1 and W_2, as well as the dead-end states enable us to construct, given a winning strategy for Player 1 in G', a winning strategy for him in G, and similarly for Player 2. Hence, we have the following.

Lemma 1. *Player 1 wins $\langle G, \Gamma \rangle$ iff he wins $\langle G', \Gamma' \rangle$.*

It is thus left to show how to solve the game $\langle G', \Gamma' \rangle$.

Solving the game $\langle G', \Gamma' \rangle$ We say that a strategy of Player 1 is an *almost memoryless* strategy if, intuitively, in every play, when visiting a vertex, Player 1 plays in the same manner except for, possibly, the last visit to the vertex. Formally, consider a Player 1 strategy ρ_1. Consider a vertex $v \in V_1$. We say that ρ_1 is almost memoryless for v iff there are two vertices $v_1, v_2 \in V_2$ such that for every Player 2 strategy ρ_2, if $out(\rho_1, \rho_2)[0 : n] = v$, then either $out(\rho_1, \rho_2)[n + 1] = v_1$, or $out(\rho_1, \rho_2)[n + 1] = v_2$ and for every index $n' > n + 1$ we have $out(\rho_1, \rho_2)[n'] \neq v$. We say that ρ_1 is almost memoryless iff it is almost memoryless for every vertex in V_1.

Lemma 2. *If Player 1 has a winning strategy in $\langle G', \Gamma' \rangle$, then he also has an almost memoryless winning strategy.*

Proof: Assume ρ_1 is a Player 1 winning strategy. Our goal is to construct an almost memoryless winning strategy ρ_1' from ρ_1. Intuitively, we divide the Player 1 strategy into two "phases": in the first phase, which we refer to as the "accumulation phase", Player 1's goal is to force the game into accumulating a high value. In the second phase, which we refer to as the "reachability phase", his goal is to force the game to a winning position, which is a vertex in $F_{\mathcal{A}} \times F_{\mathcal{B}}$. Since all the vertices in V' are not in W_2, Player 1 can force the game to a winning position from every vertex in V'. Also, since reaching a winning position is done in a memoryless manner, it does not involve a play with cycles, and thus, we bound the maximal value that Player 1 needs to accumulate in the first phase. The technical details can be found in the full version. □

For Player 2, our situation is even better as, intuitively, cycles in the game are either good for Player 2, in which case a strategy for him would always proceed to these cycles, or bad for Player 2, in which case a strategy for him would never enter them. Formally, as proven in the full version, we have the following.

Lemma 3. *If Player 2 has a winning strategy in $\langle G', \Gamma' \rangle$, then he also has a winning memoryless strategy.*

Before turning to prove the complexity results, we remind the reader of the Bellman-Ford algorithm. The algorithm gets as input a weighted directed graph $\langle \mathcal{V}, \mathcal{E}, \theta \rangle$, where $\mathcal{V}$ is a set of vertices, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges, and $\theta : \mathcal{E} \to \mathbb{R}$ is a weight function. The algorithm also gets a distinguished source vertex $s \in \mathcal{V}$. It outputs a function $C : \mathcal{V} \to \mathbb{R}$, where for every $v \in \mathcal{V}$, the value $C(v)$ is the value of the shortest path between s and v. If there is a negative cycle connected to s, the algorithm reports that such a cycle exists but it cannot return a correct answer since no shortest path exists.

We continue to prove the complexity that follows.

Theorem 6. *Solving simulation games is in NP $\cap$ co-NP.*

Proof: We first show membership in co-NP. We show that we can check in PTIME, given a memoryless Player 2 strategy, whether it is a winning strategy for Player 2. Given a memoryless Player 2 strategy, we trim every edge that starts from vertices in V_2' and does not agree with the strategy. We run the longest path version of the Bellman-Ford algorithm on the trimmed arena. Given a directed graph and a source vertex, the algorithm returns a function $C : V \to \mathbb{R}$ that assigns to every vertex the longest path from the source vertex, and reports if there is a positive valued cycle in the graph.

In the full version, we prove that the strategy is winning iff there is no positive cycle in the trimmed arena, and if every vertex $v \in F_{\mathcal{A}} \times F_{\mathcal{B}}$ has $C(v) \leq 0$.

We proceed to show membership in NP. For that, we show that we can check in PTIME, given a memoryless Player 1 strategy ρ_1, whether it can serve as the strategy to be used in the "accumulation phase" of the game in a way that induces an almost memoryless winning strategy for Player 1. Intuitively, we check if Player 2 can play against ρ_1 in a way that closes a cycle that is winning for Player 2, or if Player 2 can reach the vertex v_{sink} in a play that is not losing. If he cannot, we show that ρ_1 either forces the game into a vertex in $F_{\mathcal{A}} \times F_{\mathcal{B}}$ after a positive-valued play, or ρ_1 forces the game to close a positive valued cycle.

The algorithm, described in the full version, is similar to the algorithm in [4] with a small adjustment, which in turn is an adjustment of the Bellman-Ford algorithm: we restrict the vertices considered by the algorithm to ones in $F_{\mathcal{A}} \times F_{\mathcal{B}}$, and we take into an account the ability to reach v_{sink}. □

4 An Abstraction-Refinement-based Algorithm for Deciding Simulation

In this section we solve the weighted-simulation problem $\mathcal{A} \leq \mathcal{B}$ by reasoning about abstractions of $\mathcal{A}$ and $\mathcal{B}$. Recall that for every WFA $\mathcal{A}$, we have that $\mathcal{A}_{\downarrow} \subseteq \mathcal{A} \subseteq \mathcal{A}_{\uparrow}$. We first argue that this order is maintained for the simulation relation. We then use this fact in order to check simulation (and hence, also containment) with respect to abstractions.

Theorem 7. *For every WFA $\mathcal{A}$ and abstraction function α, we have $\mathcal{A}^{\alpha}_{\downarrow} \leq \mathcal{A} \leq \mathcal{A}^{\alpha}_{\uparrow}$.*

Proof: We construct the required winning strategies for Player 2. We start with the claim $\mathcal{A}_{\downarrow} \leq \mathcal{A}$ and show that Player 2 has a winning strategy in the game that corresponds to $\mathcal{A}_{\downarrow}$ and $\mathcal{A}$. Intuitively, whenever Player 1 selects a letter and a must-transition to proceed with in $\mathcal{A}_{\downarrow}$, the winning Player 2 strategy selects a matching concrete transition in $\mathcal{A}$ and proceeds with it. Thus, the winning Player 2 strategy maintains the invariant that when the game reaches a vertex $\langle a, c \rangle$, then $c \in a$. Recall that $\mathcal{A}_{\downarrow}$ underapproximates $\mathcal{A}$ in three ways: the transition relation, the weight function, and the definition of the accepting states. Consequently, as we formally prove in the full version by induction on the length of the prefix, all the prefixes of the play are not winning for Player 1. The proof of the second claim is similar. □

We note that beyond the use of Theorem 7 in practice, it provides an additional witness to the appropriateness of our definition of weighted simulation.

Recall that our algorithm solves the weighted-simulation problem $\mathcal{A} \leq \mathcal{B}$ by reasoning about abstractions of $\mathcal{A}$ and $\mathcal{B}$. We first show that indeed we can conclude about the existence of simulation or its nonexistence by reasoning about the abstractions:

Theorem 8. *Consider two WFAs $\mathcal{A}$ and $\mathcal{B}$ and abstraction functions α and β.*

- *If $\mathcal{A}^{\alpha}_{\uparrow} \leq \mathcal{B}^{\beta}_{\downarrow}$, then $\mathcal{A} \leq \mathcal{B}$.*
- *If $\mathcal{A}^{\alpha}_{\downarrow} \not\leq \mathcal{B}^{\beta}_{\uparrow}$, then $\mathcal{A} \not\leq \mathcal{B}$.*

Our algorithm proceeds as follows. We start by checking whether $\mathcal{A}^{\alpha}_{\uparrow} \leq \mathcal{B}^{\beta}_{\downarrow}$ and $\mathcal{A}^{\alpha}_{\downarrow} \not\leq \mathcal{B}^{\beta}_{\uparrow}$, for some (typically coarse) initial abstraction functions α and β. By Theorem /refrefinement-thm, if we are lucky and one of them holds, we are done. Otherwise, the winning strategies of the Player 2 in the first case and Player 1 in the second case suggest a way to refine α and β, and we repeat the process with the refined abstractions. While refinement in the Boolean case only splits abstract states in order to close the gap between may and must transitions, here we also have refinement steps that tighten the weights along transitions. Below is a formal description of the algorithm.

Input: Two WFAs $\mathcal{A}$ and B, with abstraction functions α and β
Output: *yes* if $\mathcal{A} \leq \mathcal{B}$ and *no* otherwise
while true **do**
  if Player 2 wins the game that corresponds to $\mathcal{A}^{\alpha}_{\uparrow}$ and $\mathcal{B}^{\beta}_{\downarrow}$ **then** return *yes*
  else let ρ_1 be a winning Player 1 strategy in the game
  if Player 1 wins the game that corresponds to $\mathcal{A}^{\alpha}_{\downarrow}$ and $\mathcal{B}^{\beta}_{\uparrow}$ **then** return *no*
  else let ρ_2 be a winning Player 2 strategy in the game
    $\alpha', \beta' = \mathit{refine}(\mathcal{A}, \mathcal{B}, \alpha, \beta, \rho_1, \rho_2)$
    set: $\alpha = \alpha'$ and $\beta = \beta'$
end while

By Theorem 8, if the algorithm returns *yes*, then $\mathcal{A}$ simulates $\mathcal{B}$, and if the algorithm returns *no*, then $\mathcal{A}$ does not simulate $\mathcal{B}$. If $\mathcal{A}^{\alpha}_{\uparrow} \not\leq \mathcal{B}^{\beta}_{\downarrow}$ and $\mathcal{A}^{\alpha}_{\downarrow} \leq \mathcal{B}^{\beta}_{\uparrow}$, then the answer is

indefinite and we refine the abstractions. This is done by the procedure $refine$, described below.

Recall that we refine α and β in case both $\mathcal{A}^{\alpha}_{\uparrow} \not\leq \mathcal{B}^{\beta}_{\downarrow}$ and $\mathcal{A}^{\alpha}_{\downarrow} \leq \mathcal{B}^{\beta}_{\uparrow}$. When this happens, the algorithm for checking simulation generates a winning strategy ρ_1 of Player 1 in the game corresponding to $\mathcal{A}^{\alpha}_{\uparrow} \not\leq \mathcal{B}^{\beta}_{\downarrow}$ and a winning strategy ρ_2 of Player 2 in the game corresponding to $\mathcal{A}^{\alpha}_{\downarrow} \leq \mathcal{B}^{\beta}_{\uparrow}$. The procedure $refine$ gets as input the WFAs $\mathcal{A}$ and $\mathcal{B}$, the abstraction functions α and β, and the winning strategies ρ_1 and ρ_2. It returns two new abstraction functions α' and β'.

In order to see the idea behind $refine$, assume that Player 1 wins in the concrete game. Then, ρ_2 is winning in a *spurious* manner in the game that corresponds to $\mathcal{A}^{\alpha}_{\downarrow} \leq \mathcal{B}^{\beta}_{\uparrow}$. Our goal in the refinement process is to remove at least one of the reasons ρ_2 is winning. Also, if Player 1 wins in the concrete game, then refinement in the game corresponding to $\mathcal{A}^{\alpha}_{\uparrow} \leq \mathcal{B}^{\beta}_{\downarrow}$ should reveal the fact that ρ_1 is winning. The situation is dual if Player 2 wins the concrete game. Since during the refinement process we cannot know which of the players wins the concrete game, we perform refinements to the two games simultaneously until we reach a definite answer. Since we assume that the WFAs are finite, we are guaranteed to eventually terminate. Thus, our procedure is complete (it attempts, however, to solve only the simulation, rather than containment, problem).

Observe that the arenas on which the strategies are defined have the same vertices but different edges. Edges that appear in one game but not the other correspond to may transitions that are not must transitions. Our refinement procedure is based on the algorithm for solving the simulation game as described in Section 3.2. Recall that the algorithm first performs a pre-processing stage in which it removes two sets of vertices: vertices from which Player 1 wins (namely the set W_1), and vertices from which Player 2 wins (namely the set W_2). The first set of vertices are the winning vertices in the un-weighted reachability game with objective $F_{\mathcal{A}} \times (Q_{\mathcal{B}} \setminus F_{\mathcal{B}})$. The second set of vertices are the winning vertices in the un-weighted safety game with objective $((Q_{\mathcal{A}} \setminus F_{\mathcal{A}}) \times Q_{\mathcal{B}}) \cup V_2$.

Since the two winning sets depend on the edges of the game, the sets we remove are not the same in the two games. We refer to the vertices after their removal as $V'_{\mathcal{A}_\uparrow,\mathcal{B}_\downarrow}$ and $V'_{\mathcal{A}_\downarrow,\mathcal{B}_\uparrow}$, and we refine them until the initial vertex is in both sets.

We describe the refinement according to the strategy ρ_1. Refinement according to ρ_2 is dual.

Recall that, by Theorem 2, if Player 1 wins, then he has an almost-memoryless winning strategy. Thus, we assume ρ_1 is almost memoryless. Also recall that ρ_1 is winning in the game played on the vertices $V'_{\mathcal{A}_\uparrow,\mathcal{B}_\downarrow}$, and since ρ_2 is winning in the game played on the vertices $V'_{\mathcal{A}_\downarrow,\mathcal{B}_\uparrow}$, the strategy ρ_1 is not winning in this game.

We proceed as in the algorithm for solving simulation games: we "guess" the strategy ρ_1 and check if (actually, how) Player 2 can win against this strategy. Since ρ_1 is not winning in the game played on the vertices $V'_{\mathcal{A}_\uparrow,\mathcal{B}_\downarrow}$, we find at least one path π that is winning for Player 2. As seen in the algorithm, π is either a path that reaches v_{sink} or is a lasso contained in the vertices $V'_{\mathcal{A}_\uparrow,\mathcal{B}_\downarrow} \setminus \{v_{sink}\}$. More formally, π is of the form $\pi_1 \cdot \pi_2^{\omega}$. The path π_1 is a simple path that uses vertices from $V'_{\mathcal{A}_\uparrow,\mathcal{B}_\downarrow} \setminus \{v_{sink}\}$ and it (and every prefix of it) is not losing for Player 2. The path π_2 is either the cycle that is the single vertex v_{sink} or it is a cycle contained in $V'_{\mathcal{A}_\uparrow,\mathcal{B}_\downarrow} \setminus \{v_{sink}\}$. In the

second case, $val(\pi_2) \leq 0$, and for every $0 \leq i \leq |\pi_2|$, if $\pi_2[i] \in F_{\mathcal{A}} \times F_{\mathcal{B}}$, then $val(\pi_1) + val(\pi_2[0:i]) \leq 0$.

Since π is not a path in $V'_{\mathcal{A}_\downarrow,\mathcal{B}_\uparrow}$, at least one of the following three cases hold:

- π uses a vertex in $V'_{\mathcal{A}_\uparrow,\mathcal{B}_\downarrow} \setminus V'_{\mathcal{A}_\downarrow,\mathcal{B}_\uparrow}$,
- π traverses an edge that corresponds to a may but not must transition, or
- the sum of the edges traversed in π is larger in the one game than in the other.

In the first case, we refine the vertices $V'_{\mathcal{A}_\uparrow,\mathcal{B}_\downarrow}$ and $V'_{\mathcal{A}_\downarrow,\mathcal{B}_\uparrow}$, as described above. In the second case, the refinement is similar to the one done in the Boolean setting, where we close the gap between may and must transitions. Finally, in the third case, we split states in order to tighten the weights on the transitions. Recall that these weights are defined by taking the minimum or maximum of the corresponding set of transitions. Therefore, splitting of states indeed tightens the weights.

Example 2. Consider the two simulation games $\mathcal{G}_1$ and $\mathcal{G}_2$ in Figure 2. The game $\mathcal{G}_1$ corresponds to $\mathcal{A}_\uparrow$ and $\mathcal{B}_\downarrow$, and $\mathcal{G}_2$ corresponds to $\mathcal{A}_\downarrow$ and $\mathcal{B}_\uparrow$, for some two WFAs $\mathcal{A}$ and $\mathcal{B}$ with abstraction functions. In the figure, we use circle and boxes in order to denote, respectively, the nodes in which Player 1 and Player 2 proceed. In $\mathcal{G}_1$, we define $F_{\mathcal{A}} \times (Q_{\mathcal{B}} \setminus F_{\mathcal{B}}) = \{s_4\}$, $F_{\mathcal{A}} \times F_{\mathcal{B}} = \{s_5, s_6\}$, and $(Q_{\mathcal{A}} \setminus F_{\mathcal{A}}) \times Q_{\mathcal{B}} = \{s_4\}$, and the definition is similar in $\mathcal{G}_2$. Due to lack of space, we omit the letters from the arenas.

We show that Player 1 wins $\mathcal{G}_1$ in three different ways and Player 2 wins $\mathcal{G}_2$. In the first strategy, in $\mathcal{G}_1$, Player 1 proceeds from s_0 to s_1. Player 2 is then forced to continue to s_4, which is losing for Player 2 since it is a vertex in $F_{\mathcal{A}} \times (Q_{\mathcal{B}} \setminus F_{\mathcal{B}})$. In the second winning strategy, Player 1 proceeds from s_0 to s_2. The game continues by alternating between s_2 and s_6. Since the cycle has a positive value and $s_6 \in F_{\mathcal{A}} \times F_{\mathcal{B}}$, Player 1 wins the prefix $s_0 s_2 s_6 s_2 s_6 s_2 s_6$. Finally, in the third strategy, Player 1 proceeds from s_0 to s_3. Player 2 is then forced to proceed to s_6. Since $s_6 \in F_{\mathcal{A}} \times F_{\mathcal{B}}$ and $val(s_0 s_2 s_6) > 0$, Player 1 wins the prefix. Clearly, in $\mathcal{G}_2$, the Player 2 strategy that proceeds from s_1 to s_5 is winning.

We proceed to describe the refinement of the abstractions using these strategies. First, note that in $\mathcal{G}_1$, by playing the first strategy described above, Player 1 can force the game to a vertex in $F_{\mathcal{A}} \times (Q_{\mathcal{B}} \setminus F_{\mathcal{B}})$ from the initial vertex. Thus, $s_0 \notin V'_{\mathcal{A}_\uparrow,\mathcal{B}_\downarrow}$. We start by refining the set of Player 1 winning vertices in the reachability game with objective $F_{\mathcal{A}} \times (Q_{\mathcal{B}} \setminus F_{\mathcal{B}})$. In this process we refine the vertex s_1.

Next, we apply the third Player 1 winning strategy on G_2 and see how Player 2 can win against it. Player 2 wins because Player 1 uses the edge $\langle s_0, s_3 \rangle$, which is not in $\mathcal{G}_2$. We refine s_0, and after the refinement the strategy is no longer valid for Player 1 in $\mathcal{G}_1$. After these two refinements, Player 1 can still win in $\mathcal{G}_1$ using the second strategy, and we apply it in $\mathcal{G}_2$. The outcome of the game against a winning Player 2 strategy is $s_0(s_2 s_6)^\omega$. In this path, we find the failure vertex s_2 and refine it in order to tighten the values of the edges.

The two resulting games after these three refinements are $\mathcal{G}'_1$ and $\mathcal{G}'_2$ (see the right side of Figure 2). Player 1 wins in $\mathcal{G}'_2$ by proceding from s_0 to s_2, and thus we are done.

Note that since not all the values on the edges are the same in $\mathcal{G}'_1$ and $\mathcal{G}'_2$, the refinement is not exauhsted. That is, the arenas are not $Q_{\mathcal{A}} \times Q_{\mathcal{B}}$. Thus, the abstraction-refinement algorithm successfully decides simulation on a smaller state space than the

concrete one. Since, however, we found that $\mathcal{A} \not\leq \mathcal{B}$, then by Theorem 2, it might still be the case that $\mathcal{A} \subseteq \mathcal{B}$.

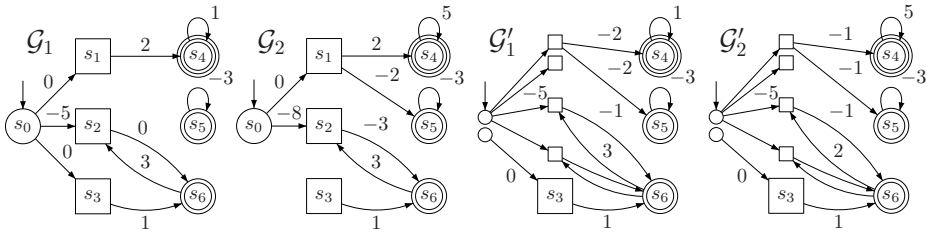

Fig. 2. An example of applying the refinement algorithm on two simulation games.

5 Directions for Future Research

We introduced the notions of abstraction and simulation for weighted automata and argue that they form a useful heuristic for checking containment – a problem of practical interest that is known to be undecidable. In the Boolean setting, researchers have suggested ways for closing the gap between containment and simulation [23,26]. Some, like these that extend the definition of simulation with a look ahead, are easy to extend to the weighted setting. Other ways require special treatment of the accumulated weights and are subject to future research. Finally, the rich weighted setting allows one to measure the differences between systems. For example, we can talk about one WFA t-approximating another WFA, in the sense that the value of a word in the second is at most t times its value in the first [2]. Our weighted simulation corresponds to the special case $t = 1$ and we plan to study approximated weighted simulation.

References

1. S. Almagor, U. Boker, and O. Kupferman. What's decidable about weighted automata? In *9th ATVA*, LNCS 6996, pages 482–491, 2011.
2. B. Aminof, O. Kupferman, and R. Lampert. Formal analysis of online algorithms. In *9th ATVA*, LNCS 6996, page 213-227 Springer, 2011.
3. T. Ball, E. Bounimova, B. Cook, V. Levin, J. Lichtenberg, C. McGarvey, B. Ondrusek, S.K. Rajamani, and A. Ustuner. Thorough static analysis of device drivers. In *EuroSys*, 2006.
4. K. Larsen N. Markey P. Bouyer, U. Fahrenberg and J. Srba. Infinite runs in weighted timed automata with energy constraints. In *FORMATS*, pages 33–47, 2008.
5. G. Bruns and P. Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *Proc. 11th CAV*, pages 274–287, 1999.
6. T. Ball and O. Kupferman. An abstraction-refinement framework for multi-agent systems. In *Proc. 21st LICS*, 2006.
7. U. Boker and O. Kupferman. Co-ing Büchi made tight and helpful. In *Proc. 24th LICS*, pages 245–254, 2009.
8. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for the static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th POPL*, pages 238–252, 1977.

9. K. Chatterjee and L. Doyen. Energy parity games. In *Proc. 37th ICALP*, pages 599–610, 2010.
10. K. Chatterjee, L. Doyen, and T. Henzinger. Quantative languages. In *Proc. 17th CSL*, pages 385–400, 2008.
11. K. Chatterjee, L. Doyen, and T. Henzinger. Probabilistic weighted automata. In *Proc. 20th CONCUR*, pages 224–258, 2009.
12. K. Chatterjee, L. Doyen, and T. A. Henzinger. Expressiveness and closure properties for quantitative languages. *LMCS*, 6(3), 2010.
13. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5):752–794, 2003.
14. P. Černý, T. Henzinger, and A. Radhakrishna. Simulation distances. In *Proc. 21st CONCUR*, pages 253–268, 2010.
15. K. Culik and J. Kari. Digital images and formal languages. *Handbook of formal languages, vol. 3: beyond words*, pages 599–616, 1997.
16. L. de Alfaro and P. Roy. Solving games via three-valued abstraction refinement. In *Proc. 18th CONCUR*, pages 74–89, 2007.
17. M. Droste and P. Gastin. Weighted automata and weighted logics. In *Proc. 32nd ICALP*, pages 513–525, 2005.
18. O. Grumberg, M. Lange, M. Leucker, and S. Shoham. When not losing is better than winning: Abstraction and refinement for the full μ-calculus. *I&C*, 205(8):1130–1148, 2007.
19. D. Gale and F. M. Stewart. Infinite games of perfect information. *Ann. Math. Studies*, 28:245–266, 1953.
20. M.R. Henzinger, T.A. Henzinger, and P.W. Kopke. Computing simulations on finite and infinite graphs. In *Proc. 36th FOCS*, pages 453–462, 1995.
21. T. Henzinger, R. Jhala, and R. Majumdar. Counterexample-guided control. In *Proc. 30th ICALP*, pages 886–902, 2003.
22. 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.
23. Y. Kesten, N. Piterman, and A. Pnueli. Bridging the gap between fair simulation and trace containment. In *Proc. 15th CAV*, LNCS 2725, pages 381–393, 2003.
24. D. Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation*, 4(3):405–425, 1994.
25. D. Kuperberg. Linear temporal logic for regular cost functions. In *Proc. 28th STACS*, pages 627–636, 2011.
26. N. A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. 6th PODC*, pages 137–151, 1987.
27. K.G. Larsen and G.B. Thomsen. A modal process logic. In *Proc. 3rd LICS*, 1988.
28. R. Milner. An algebraic definition of simulation between programs. In *Proc. 2nd IJCAI*, pages 481–489, 1971.
29. M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.
30. M. Mohri, F.C.N. Pereira, and M. Riley. Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, 16(1):69–88, 2002.
31. A.R. Meyer and L.J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proc. 13th SWAT*, pages 125–129, 1972.
32. M.P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961.
33. A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.