Strategic Dominance: A New Preorder for Nondeterministic Processes

Thomas A. Henzinger
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Nicolas Mazzocchi
Slovak University of Technology in Bratislava, Slovak Republic
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

N. Ege Saraç
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Abstract

We study the following refinement relation between nondeterministic state-transition models: model $\mathcal{B}$ *strategically dominates* model $\mathcal{A}$ iff every deterministic refinement of $\mathcal{A}$ is language contained in some deterministic refinement of $\mathcal{B}$. While language containment is trace inclusion, and the (fair) simulation preorder coincides with tree inclusion, strategic dominance falls strictly between the two and can be characterized as "strategy inclusion" between $\mathcal{A}$ and $\mathcal{B}$: every strategy that resolves the nondeterminism of $\mathcal{A}$ is dominated by a strategy that resolves the nondeterminism of $\mathcal{B}$. Strategic dominance can be checked in 2-ExpTime by a decidable first-order Presburger logic with quantification over words and strategies, called *resolver logic*. We give several other applications of resolver logic, including checking the co-safety, co-liveness, and history-determinism of boolean and quantitative automata, and checking the inclusion between hyperproperties that are specified by nondeterministic boolean and quantitative automata.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Logic and verification; Theory of computation → Program reasoning

Keywords and phrases quantitative automata, refinement relation, resolver, strategy, history-determinism

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.29

Funding This work was supported in part by the ERC-2020-AdG 101020093. N. Mazzocchi was affiliated with ISTA when this work was submitted for publication.

Acknowledgements We thank the anonymous reviewers for their helpful comments.

1 Introduction

Nondeterminism is a powerful mechanism for varying the degree of detail shown in a state-transition model of a system. Intuitively, a nondeterministic model captures the set of possible deterministic implementations. Consider the process model $ab + ac$. This model allows two possible deterministic implementations, ab and ac. Mathematically, each deterministic implementation of a nondeterministic model corresponds to a *strategy* for resolving the nondeterminism, i.e., a function f that maps a finite run h through the model (the "history") and a new letter σ to a successor state $f(h, \sigma)$ allowed by the model. Consider the recursive process model $X = abX + acX$, which repeatedly chooses either the ab branch or the ac branch. There are infinitely many different strategies to resolve the repeated nondeterminism, e.g., the strategy f_{prime} whose nth choice is the ab branch if $n = 1$ or n is prime, and whose nth choice is the ac branch when $n > 1$ and n is composite.

Two models that describe the same system at different levels of detail are related by a preorder. Milner's simulation preorder and the trace-inclusion preorder represent two particularly paradigmatic and widely studied examples of preorders on state-transition

models, but many variations and other refinement preorders can be found in the literature [30, 29, 4, 9, 26]. In its purest form, the *linear-time* view of model refinement postulates that for two models $\mathcal{A}$ and $\mathcal{B}$, for $\mathcal{B}$ to describe the same system as $\mathcal{A}$ at a higher level of abstraction, the less precise model $\mathcal{B}$ must allow all traces that are possible in the more refined model $\mathcal{A}$ while potentially allowing more traces. In contrast, the pure *branching-time* view of model refinement postulates that for two models $\mathcal{A}$ and $\mathcal{B}$, for $\mathcal{B}$ to describe the same system as $\mathcal{A}$ at a higher level of abstraction, the less precise model $\mathcal{B}$ must allow all subtrees of the full computation tree of the more refined model $\mathcal{A}$ (modulo its isomorphic subtrees) while potentially allowing more trees. This tree-inclusion view of branching time corresponds to the simulation preorder and its generalization to fair simulation [20].

In this paper, we introduce and study a third fundamental view of model refinement – *strategy inclusion* – which lies strictly between trace inclusion and tree inclusion. According to the strategic view, for $\mathcal{B}$ to describe the same system as $\mathcal{A}$ at a higher level of abstraction, the less precise model $\mathcal{B}$ must allow all deterministic implementations that are possible for the more refined model $\mathcal{A}$ (but $\mathcal{B}$ may allow strictly more implementations than $\mathcal{A}$). Mathematically, we say that $\mathcal{B}$ (*strategically*) *dominates* $\mathcal{A}$ if for every strategy f resolving the nondeterminism of $\mathcal{A}$, there exists a strategy g for resolving the nondeterminism of $\mathcal{B}$ such that every trace of $\mathcal{A}^f$ (the deterministic result of applying the strategy f to the nondeterministic model $\mathcal{A}$) corresponds to a trace of $\mathcal{B}^g$.

Strategic dominance, like simulation, is a branching-time preorder, but while simulation corresponds to inclusion of all subtrees of the full computation tree of a model, dominance corresponds to the inclusion of all *deterministic* subtrees. To see that dominance is strictly coarser than simulation, recall the model $X = abX + acX$ and compare it with the model $Y = ababY + abacY + acabY + acacY$. Both models X and Y have the same traces, but Y does not simulate X. Nonetheless, for every strategy resolving the nondeterminism of X there exists a strategy resolving the nondeterminism of Y that produces same infinite trace (and vice versa). In particular, the strategy g_{prime} for Y that dominates the strategy f_{prime} for X makes the following choices: initially g_{prime} chooses the $abab$ branch (since 2 is prime) and thereafter, for all $n > 1$, the nth choice of g_{prime} is $abac$ if $2n + 1$ is prime, and $acac$ otherwise (because $2n + 2$ is never prime). To see that strategic dominance is strictly finer than trace inclusion, the standard example of comparing $a(b + c)$ with $ab + ac$ will do; these two process models are trace equivalent, but only the former has a deterministic implementation that includes both traces.

Relation to prior work. Our motivation for studying the strategic view of model refinement originated from the definition of *history-determinism* [22, 8]. A state-transition model $\mathcal{A}$ is history-deterministic if there exists a strategy for resolving the nondeterminism of $\mathcal{A}$ which captures exactly the language of $\mathcal{A}$. In other words, the nondeterminism of a history-deterministic model can be resolved on-the-fly, by looking only at the history, without making guesses about the future.

Strategies (a.k.a. policies) are a central concept of game theory, and our resolvers correspond mathematically to deterministic strategies of games played on graphs. However, we study the *single-player* case – where the player resolves the nondeterminism of a state-transition model – and not the multi-player case that arises in game models [3, 12]. Besides general strategies, we consider the special cases of *finite-state* strategies (which can remember only a finite number of bits about the history), and of *positional* strategies (which have no memory about the history). Positional (a.k.a. memoryless) strategies correspond, in our setting, to the removal of nondeterministic edges from a model ("edge pruning"). To the best of our knowledge, strategic dominance is a novel relation that has not been studied before.

In particular, the seminal works on the linear time-branching time spectrum of sequential processes [30, 29] do not present a comparable relation. In general, these works do not involve a game-theoretic view of nondeterminism, which is a crucial aspect captured through our use of resolvers. In this way, the game-theoretic view adds a new dimension to the spectrum of process preorders.

For generality, we study strategies for state-transition models in a *quantitative* setting, where all states have outgoing transitions and all transitions have numeric weights [11]. In this setting, every finite path through a state-transition model can be extended, every infinite path is assigned a numeric value, and the values of different paths can be compared. The trace preorder between $\mathcal{A}$ and $\mathcal{B}$ requires that for every infinite word w, for every run of $\mathcal{A}$ on w there exists a run of $\mathcal{B}$ on w of equal or greater value. All refinement relations we consider are refinements of the trace preorder. The quantitative setting generalizes the boolean setting (take $\{0, 1\}$ as the value set, and assign the value 1 to a run iff the run is accepted by the model), and it generalizes many common acceptance conditions on finite and infinite runs (Sup and Inf values correspond to reachability and safety acceptance; LimSup and LimInf values to Büchi and coBüchi acceptance). Extensions of simulation and inclusion preorders to this setting have been studied in [11].

Contributions of this paper. Our results are threefold. First, we define *resolver logic* as a first-order logic that is interpreted over a set QA of quantitative finite automata over infinite words. Resolver logic quantifies over infinite words w, strategies f that resolve the nondeterminism of automata $\mathcal{A} \in \mathsf{QA}$ (so-called "resolvers"), and natural numbers. The existentially quantified formulas are built from terms of the form $\mathcal{A}^f(w)$ using Presburger arithmetic. The term $\mathcal{A}^f(w)$ denotes the value of the unique run of the automaton $\mathcal{A}$ over the word w when all nondeterministic choices are resolved by the strategy f. We show that model-checking problem for resolver logic – i.e., the problem of deciding if a closed formula φ is true over a given set $\mathcal{A}$ of automata – can be solved in d-ExpTime, where d is the number of quantifier switches in φ. Our model-checking algorithm uses automata-theoretic constructions over parity tree automata that represent resolvers. A main difference of resolver logic to strategy logics [12, 25], besides the handling of quantitative constraints and Presburger arithmetic, lies in the quantification over infinite words. This quantification is not present in strategy logics, and lets us define strategic dominance and other inclusion-based relations.

Second, we define in resolver logic *eight different resolver relations* between quantitative automata, including the strategic-dominance relation explained above. The eight relations are obtained from each other by reordering quantifiers in resolver logic. We show that six of the eight relations are preorders: one coincides with simulation, four coincide with trace inclusion, and the sixth – strategic dominance – lies strictly between simulation and trace inclusion. The remaining two relations are finer than simulation and transitive, but not reflexive. Since all eight relations are defined in resolver logic, they can be decided using the model-checking algorithm for resolver logic. We also specialize our expressiveness and decidability results to specific value functions (Sup; Inf; LimSup; LimInf) and to restricted classes of strategies (positional; finite-state).

Third, we provide three more applications for resolver logic, in addition to checking strategic dominance. These applications show that resolvers play a central role in many different automata-theoretic problems.

Three more applications. Our first application concerns the *co-safety and co-liveness* of boolean and quantitative automata [21, 5]. In [5], while the authors solved the problems of deciding safety or liveness, they left open the *bottom-value problem* for quantitative automata, which needs to be solved when checking if a quantitative automaton specifies a co-safety or co-liveness property. The bottom value of a quantitative automaton $\mathcal{A}$ can be defined as the infimum over all words w of the supremum over all resolvers f of the value $\mathcal{A}^f(w)$. The bottom value, and all similarly defined values, can therefore be computed using the model-checking algorithm for resolver logic.

Our second application concerns the *history-determinism* of boolean and quantitative automata. For a model $\mathcal{B}$ to strategically dominate $\mathcal{A}$, different resolvers for $\mathcal{A}$ may give rise to different resolvers for $\mathcal{B}$. Alternatively, one may postulate that in refinement, all resolvers for $\mathcal{A}$ should be "dominated" by the same resolver for $\mathcal{B}$, that is, a single deterministic implementation of $\mathcal{B}$ should capture all possible deterministic implementations of $\mathcal{A}$. Such *blind domination* gives rise to a nonreflexive relation on quantitative automata which implies simulation. We show that a quantitative automaton is history-deterministic iff the automaton blindly dominates itself. Consequently, our model-checking algorithm for resolver logic can be used to check history-determinism and generalizes the algorithm provided in [22] for checking the history determinism of boolean automata.

Our third application concerns the *specification of hyperproperties*. A hyperproperty is a set of properties [14]; hyperproperties occur in many different application contexts such as system security. A nondeterministic boolean automaton $\mathcal{A}$ can be viewed as specifying a hyperproperty $[\![A]\!]$ in the following natural way: if $R(\mathcal{A})$ is the set of all possible resolvers for the nondeterminism of $\mathcal{A}$, then $[\![\mathcal{A}]\!] = \{\mathcal{A}^f \mid f \in R(\mathcal{A})\}$, where $\mathcal{A}^f$ is the property (or "language") that is specified by $\mathcal{A}$ if its nondeterminism is resolved by f. In the same way, nondeterministic quantitative automata can be used to specify quantitative hyperproperties as sets of quantitative languages [11]. We show that there are simple automata that specify the boolean hyperproperties that contain all safety (resp. co-safety) properties, which cannot be specified in HyperLTL [13]. However, there are also boolean hyperproperties that can be specified in HyperLTL but not by applying resolvers to nondeterministic automata. Finally, we show how resolver logic can decide the inclusion problem for resolver-specified hyperproperties.

2 Definitional Framework

Let $\Sigma = \{a, b, \ldots\}$ be a finite alphabet of letters. An infinite (resp. finite) word (a.k.a. trace or execution) is an infinite (resp. finite) sequence of letters $w \in \Sigma^\omega$ (resp. $u \in \Sigma^*$). Given $u \in \Sigma^*$ and $w \in \Sigma^* \cup \Sigma^\omega$, we write $u \prec w$ when u is a strict prefix of w. We denote by $|w|$ the length of $w \in \Sigma^* \cup \Sigma^\omega$ and, given $a \in \Sigma$, by $|w|_a$ the number of occurrences of a in w. We assume that the reader is familiar with formal language theory.

▶ **Definition 2.1** (automaton). *A* (quantitative) automaton *is a tuple* $\mathcal{A} = (\Sigma, Q, s, \Delta, \mu, \nu)$ *where* Σ *is a finite alphabet,* Q *is a finite set of states,* $s \in Q$ *is the initial state,* $\Delta \subseteq Q \times \Sigma \times Q$ *is a transition relation,* $\mu \colon \Delta \to \mathbb{Q}$ *is a weight function, and* $\nu \colon \mathbb{Q}^\omega \to \mathbb{R}$ *is a value function. The size of* $\mathcal{A}$ *is defined by* $|Q| + \sum_{\delta \in \Delta} 1 + \log_2(\mu(\delta))$*. A* run π *over a finite (resp. an infinite) word* $w = \sigma_0\sigma_1 \ldots$ *is a finite (resp. an infinite) sequence* $\pi = q_0\sigma_0q_1\sigma_1 \ldots$ *such that* $q_0 = s$ *is the initial state of* $\mathcal{A}$ *and* $(q_i, \sigma_i, q_{i+1}) \in \Delta$ *holds for all* $0 \leq i < |w|$ *(resp.* $i \in \mathbb{N}$*). A* history *is a run over a finite word, and we denote the set of histories of* $\mathcal{A}$ *by* $\Pi_\mathcal{A}$*. The weight sequence* $\mu(\pi)$ *of a run* π *is defined by* $x_0x_1 \ldots$ *where* $x_i = \mu(q_i, \sigma_i, q_{i+1})$ *for all* $i \in \mathbb{N}$*. The value of a run* π *is defined as* $\nu(\mu(\pi))$*. In this paper, we consider automata with* $\nu \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}\}$ *and without loss of generality* $\mu : \Delta \to \mathbb{N}$ *since a finite set of rational weights can be scaled and shifted to naturals and back.*

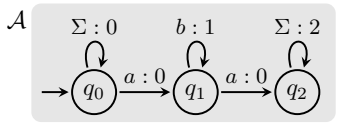

Figure 1 A LimSup-automaton $\mathcal{A}$ over $\Sigma = \{a, b\}$ with $R^{\text{pos}}(\mathcal{A}) \subsetneq R^{\text{fin}}(\mathcal{A}) \subsetneq R(\mathcal{A})$.

In general, automata resolve their nondeterminism by taking the sup over its set of runs on a given word. We take an alternative view and pair every automaton with a *resolver* – an explicit description of how nondeterminism is resolved, which is a central concept in this work. Given a finite prefix of a run and the next input letter, a resolver determines the next state of the automaton.

▶ **Definition 2.2** (resolver). *Let $\mathcal{A} = (\Sigma, Q, s, \Delta, \mu, \nu)$ be an automaton. A* resolver *for $\mathcal{A}$ is a function $f \colon \Pi_{\mathcal{A}} \times \Sigma \to Q$ such that for every history $h = q_0\sigma_0q_1\sigma_1 \dots q_n \in \Pi_{\mathcal{A}}$ and every $\sigma \in \Sigma$ we have $(q_n, \sigma, f(h, \sigma)) \in \Delta$. A resolver f for $\mathcal{A}$ and a word $w = \sigma_1\sigma_2 \dots \in \Sigma^\omega$ produce a unique infinite run $\pi_{f,w} = q_0\sigma_1q_1\sigma_2 \dots$ of $\mathcal{A}$ such that $q_0 = s$ and $f(q_0\sigma_1 \dots q_{i-1}, \sigma_i) = q_i$ for all $i \geq 1$. Given an automaton $\mathcal{A}$, we denote by $R(\mathcal{A})$ the set of its resolvers. Given a resolver $f \in R(\mathcal{A})$, we define the quantitative language $\mathcal{A}^f \colon w \mapsto \nu(\mu(\pi_{f,w}))$ where $w \in \Sigma^\omega$. We also define the quantitative language $\mathcal{A}_{\sup} \colon w \mapsto \sup_{f \in R(\mathcal{A})} \mathcal{A}^f(w)$, which is the standard interpretation of nondeterminism for automata.*

We define finite-memory and positional resolvers the usual way: a resolver f is *finite-memory* iff it can be implemented by a finite-state machine, and it is *positional* iff the output of f only depends on the last state in the input history and the incoming letter (see [17, Sec. 1.5] for the formal definitions). Given an automaton $\mathcal{A}$, we denote by $R^{\text{fin}}(\mathcal{A})$ the set of its finite-memory resolvers, and by $R^{\text{pos}}(\mathcal{A})$ set of its positional resolvers. Let us demonstrate the notion of resolvers.

▶ **Example 2.3.** Let $\mathcal{A}$ be a LimSup-automaton over the alphabet $\Sigma = \{a, b\}$ as in Figure 1 and observe that the only source of nondeterminism is the transitions (q_0, a, q_0) and (q_0, a, q_1).

Consider a resolver f_1 that maps any history of the form $q_0(bq_0)^*$ followed by a to q_1. Intuitively, it is a *positional* resolver because it ignores the transition (q_0, a, q_0) and its output only depends on the current state and the next letter. It denotes the language $\mathcal{A}^{f_1}$ that maps a given word w to 0 if w has no a, to 1 if w has exactly one a, and to 2 otherwise.

Now, consider a resolver f_2 that maps the histories of the form $q_0(bq_0)^*$ followed by a to q_0, and those of the form $q_0(bq_0)^*aq_0(bq_0)^*$ followed by a to q_1. Intuitively, it is a *finite-state* resolver because it only distinguishes between a occurring once or twice or neither. The language $\mathcal{A}^{f_2}$ then maps a word w to 0 if w has at most one a, to 1 if w has exactly two as, and to 2 otherwise.

Finally, consider a resolver f_3 that maps every history that ends at q_0 with the incoming letter a to q_1 if the given history is of the form $q_0(\Sigma q_0)^*(bq_0)^p$ where p is a prime, and to q_0 otherwise. Intuitively, it is an infinite-memory resolver because it needs to store the length of every block of bs, which is not bounded, and check whether it is prime. The language $\mathcal{A}^{f_3}$ maps a word w to 0 if it has no prefix $u = vb^pa$ with $v \in \Sigma^*$, to 1 if it has such a prefix u and $w = ub^\omega$, and to 2 otherwise.

Next, we introduce the notion of *partial resolvers*. In contrast to (nonpartial) resolvers, partial resolvers output a set of successor states for a given history and letter.

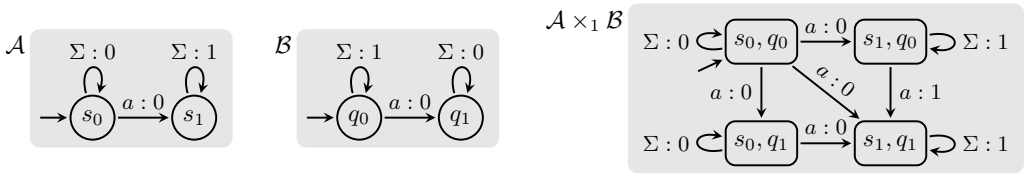

Figure 2 Two LimSup-automata $\mathcal{A}$ and $\mathcal{B}$ and the product $\mathcal{A} \times_1 \mathcal{B}$.

▶ **Definition 2.4** (partial resolver). *Let $\mathcal{A} = (\Sigma, Q, s, \Delta, \mu, \nu)$ be an automaton. A* partial resolver *for $\mathcal{A}$ is a function $f \colon \Pi_{\mathcal{A}} \times \Sigma \to 2^Q$ such that for every history $h = q_0\sigma_0 q_1\sigma_1 \dots q_n \in \Pi_{\mathcal{A}}$ and every $\sigma \in \Sigma$ we have $\{(q_n, \sigma, q) \mid q \in f(h, \sigma)\} \subseteq \Delta$. A collection of partial resolvers $f_1, \dots, f_n$ for $\mathcal{A}$ is said to be* conclusive *when $|\bigcap_{i=1}^n f_i(h, \sigma)| = 1$ for all $h \in \Pi_{\mathcal{A}}$ and all $\sigma \in \Sigma$. Given a conclusive collection of resolvers $f_1, \dots, f_n$, we denote by $\mathcal{A}^{\{f_1,\dots,f_n\}}$ the quantitative language $\mathcal{A}^f$ where f is a resolver defined by $f(h, \sigma) = \bigcap_{i=1}^n f_i(h, \sigma)$ for all $h \in \Pi_{\mathcal{A}}$ and all $\sigma \in \Sigma$.*

Partial resolvers are particularly useful when we consider products of automata. In particular, we will use these objects to capture simulation-like relations in Section 3.

▶ **Definition 2.5** (synchronized product). *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two automata such that $\mathcal{A}_i = (\Sigma, Q_i, s_i, \Delta_i, \mu_i, \nu_i)$ for $i \in \{1, 2\}$. For $k \in \{1, 2\}$, the* (synchronized) product $\mathcal{A}_1 \times_k \mathcal{A}_2$ *corresponds to the input-synchronization of $\mathcal{A}_1$ and $\mathcal{A}_2$ where the transition weights are taken from $\mathcal{A}_k$. Formally, $\mathcal{A}_1 \times_k \mathcal{A}_2 = (\Sigma, Q_1 \times Q_2, (s_1, s_2), \Delta, \mu, \nu_k)$ where the transition relation is such that $((q_1, q_2), \sigma, (q_1', q_2')) \in \Delta$ if and only if $(q_i, \sigma, q_i') \in \Delta_k$ for $i \in \{1, 2\}$, and the weight function is such that $\mu((q_1, q_2), \sigma, (q_1', q_2')) = \mu_k(q_k, \sigma, q_k')$.*

Given $i \in \{1, 2\}$, we denote by $R_i^S(\mathcal{A}_1, \mathcal{A}_2)$ the set of partial resolvers operating non-partially on the ith component of any product between $\mathcal{A}_1$ and $\mathcal{A}_2$. Formally, every $f \in R_i(\mathcal{A}_1, \mathcal{A}_2)$ is a partial resolver that satisfies the following: for all $h \in \Pi_{\mathcal{A}_1 \times_i \mathcal{A}_2}$, all $(q_1, q_2) \in Q_1 \times Q_2$ and all $\sigma \in \Sigma$, there exists $(q_i, \sigma, q_i') \in \Delta_i$ such that $f(h \cdot (q_1, q_2), \sigma)) = \{(q_1', q_2') \mid (q_{3-i}, \sigma, q_{3-i}') \in \Delta_{3-i}\}$.

Let us demonstrate the notions of partial resolvers and synchronized products.

▶ **Example 2.6.** Let $\mathcal{A}$ and $\mathcal{B}$ be as in Figure 2. Because of the loop in their initial states, as in Example 2.3, both automata have infinitely many resolvers, Their product $\mathcal{A} \times_1 \mathcal{B}$ is also shown in Figure 2. Intuitively, the product is similar to the boolean construction with difference of handling the transition weights instead of the accepting states. For $\mathcal{A} \times_1 \mathcal{B}$, the transition weights are obtained from the corresponding ones in $\mathcal{A}$.

Consider the positional resolver f of $\mathcal{A}$ that moves from s_0 to s_1 as soon as a occurs. Observe that there is a partial resolver f' of $\mathcal{A} \times_1 \mathcal{B}$, namely $f' \in R_1(\mathcal{A}, \mathcal{B})$, that imitates f. In particular, $f'((s_0, q_0), a) = \{(s_1, q_0), (s_1, q_1)\}$ since f moves $\mathcal{A}$ from s_0 to s_1 with a, but it is not specified how the $\mathcal{B}$-component of the product resolves this nondeterministic transition. Now, suppose we have $g' \in R_2(\mathcal{A}, \mathcal{B})$ such that $g'((s_0, q_0), a) = \{(s_0, q_1), (s_1, q_1)\}$. The two partial resolvers f' and g' together are conclusive for $\mathcal{A} \times_1 \mathcal{B}$, as witnessed by $f'((s_0, q_0), a) \cap g'((s_0, q_0), a) = \{(s_1, q_1)\}$.

▶ Remark 2.7. As demonstrated in Example 2.6, when we consider partial resolvers over product automata, the partial resolvers operating on the distinct components are conclusive for the product. Formally, let $\mathcal{A}$ and $\mathcal{B}$ be two automata and consider one of their products. Every pair of partial resolvers $f \in R_1(\mathcal{A}, \mathcal{B})$ and $g \in R_2(\mathcal{A}, \mathcal{B})$ is conclusive for the product automaton by definition, and thus corresponds to a (non-partial) resolver over the product.

3 Strategic Dominance and Other Resolver-Based Relations

Given two automata $\mathcal{A}$ and $\mathcal{B}$, we investigate the relations between the problems defined in Figure 3. We denote the *strategic dominance* relation by $\trianglelefteq$: the automaton $\mathcal{A}$ is strategically dominated by $\mathcal{B}$, denoted $\mathcal{A} \trianglelefteq \mathcal{B}$, iff for all resolvers $f \in R(\mathcal{A})$ there exists a resolver $g \in R(\mathcal{B})$ such that $\mathcal{A}^f(w) \leq \mathcal{B}^g(w)$ for all words $w \in \Sigma^\omega$. Intuitively, this holds when each deterministic implementation of $\mathcal{A}$ can be countered by some deterministic implementation of $\mathcal{B}$ that provides a value at least as large for each word.

We define and study other resolver-based relations and compare their expressiveness. In $\sqsubseteq$, the automaton $\mathcal{B}$ has the freedom to choose a resolver per input word (unlike in $\trianglelefteq$). The relations $\blacktriangleleft$ and $\blacksquare$ can be seen as variants of $\trianglelefteq$ and $\sqsubseteq$ where the resolvers of $\mathcal{B}$ are *blind*, meaning that they cannot depend on the resolvers of $\mathcal{A}$.

To capture simulation-like relations, we additionally define the relations $\trianglelefteq_\times, \blacktriangleleft_\times, \sqsubseteq_\times, \blacksquare_\times$ that relate product automata through their partial resolvers. In particular, we show that $\trianglelefteq_\times$, the product-based strategic dominance, coincides with simulation. Thanks to Remark 2.7, we are able to reason about these relations the same way we do for non-partial resolvers.

In addition to these resolver relations, we denote the trace-inclusion preorder by $\subseteq$ and the simulation by $\preceq$. To define simulation formally, let us recall quantitative simulation games [11]: Let $\mathcal{A} = (\Sigma, Q_\mathcal{A}, s_\mathcal{A}, \Delta_\mathcal{A}, \mu_\mathcal{A}, \nu_\mathcal{A})$ and $\mathcal{B} = (\Sigma, Q_\mathcal{B}, s_\mathcal{B}, \Delta_\mathcal{B}, \mu_\mathcal{B}, \nu_\mathcal{B})$. A strategy τ for Challenger is a function from $(Q_\mathcal{A} \times Q_\mathcal{B})^*$ to $\Sigma \times Q_\mathcal{A}$ satisfying for all $\pi = (q_1, p_1) \dots (q_n, p_n) \in (Q_\mathcal{A} \times Q_\mathcal{B})^*$, if $\tau(\pi) = (\sigma, q)$ then $(q_n, \sigma, q) \in \Delta_\mathcal{A}$. Given a strategy τ for Challenger, the set of outcomes is the set of pairs $(q_0\sigma_1 q_1 \sigma_2 q_2 \dots, p_0\sigma_1 p_1 \sigma_2 p_2 \dots)$ of runs such that $q_0 = s_\mathcal{A}$, $p_0 = s_\mathcal{B}$, and for all $i \geq 0$ we have $(\sigma_{i+1}, q_{i+1}) = \tau((q_0, p_0) \dots (q_i, p_i))$ and $(p_i, \sigma_i, p_{i+1}) \in \Delta_\mathcal{B}$. A strategy τ for Challenger is winning iff $\nu_\mathcal{A}(\mu_\mathcal{A}(r_1)) > \nu_\mathcal{B}(\mu_\mathcal{B}(r_2))$ for all outcomes (r_1, r_2) of τ.

Given a problem instance $\mathcal{A} \sim \mathcal{B}$ where $\sim$ is one of the relations in Figure 3, we write $\mathcal{A} \sim^{\text{fin}} \mathcal{B}$ (resp. $\mathcal{A} \sim^{\text{pos}} \mathcal{B}$) for the restriction of the corresponding problem statement to finite-memory (resp. positional) resolvers. For example, $\mathcal{A} \trianglelefteq^{\text{fin}} \mathcal{B}$ iff for all $f \in R^{\text{fin}}(\mathcal{A})$ there exists $g \in R^{\text{fin}}(\mathcal{B})$ such that $\mathcal{A}^f(w) \leq \mathcal{B}^g(w)$ for all $w \in \Sigma^\omega$.

▶ Remark 3.1. The results in this paper hold for quantitative Inf-, Sup-, LimSup-, and LimInf-automata as well as boolean safety, reachability, Büchi, and coBüchi automata. Moreover, they also hold when restricted to finite-state or positional resolvers.

We start with a short lemma showing that the supremum over the values of any word w is attainable by some run over w, which follows from the proof of [11, Thm. 3].

▶ **Lemma 3.2.** *Let $\nu \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimSup}, \mathsf{LimInf}\}$ and $\mathcal{A}$ be a ν-automaton. For every $w \in \Sigma^\omega$ there exist $f \in R(\mathcal{A})$ such that $\mathcal{A}^f(w) = \mathcal{A}_{\text{sup}}(w)$.*

3.1 Implications Between Resolver Relations

We now prove the implications in Figure 3 without any memory constraints on the resolvers. We start with showing that some of these relations coincide with inclusion.

▶ **Proposition 3.3.** *Let $\nu_1, \nu_2 \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimSup}, \mathsf{LimInf}\}$. For all ν_1-automata $\mathcal{A}$ and all ν_2-automata $\mathcal{B}$, we have $\mathcal{A} \sqsubseteq \mathcal{B}$ iff $\mathcal{A} \blacksquare \mathcal{B}$ iff $\mathcal{A} \sqsubseteq_\times \mathcal{B}$ iff $\mathcal{A} \blacksquare_\times \mathcal{B}$ iff $\mathcal{A} \subseteq \mathcal{B}$. This is equally true for boolean safety, reachability, Büchi, and coBüchi automata.*

Next, we show an equivalent formulation for simulation.

▶ **Proposition 3.4.** *Let $\nu_1, \nu_2 \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimSup}, \mathsf{LimInf}\}$. For all ν_1-automata $\mathcal{A}$ and all ν_2-automata $\mathcal{B}$, we have $\mathcal{A} \trianglelefteq_\times \mathcal{B}$ iff $\mathcal{A} \preceq \mathcal{B}$. This is equally true for boolean safety, reachability, Büchi, and coBüchi automata.*

Notation	Problem statement
$\mathcal{A} \subseteq \mathcal{B}$	$\forall w \in \Sigma^\omega : \mathcal{A}_{\sup}(w) \leq \mathcal{B}_{\sup}(w)$
$\mathcal{A} \preceq \mathcal{B}$	no winning strategy for Challenger in the simulation game for $\mathcal{A}$ and $\mathcal{B}$
$\mathcal{A} \trianglelefteq \mathcal{B}$	$\forall f \in R(\mathcal{A}) : \exists g \in R(\mathcal{B}) : \forall w \in \Sigma^\omega : \mathcal{A}^f(w) \leq \mathcal{B}^g(w)$
$\mathcal{A} \trianglelefteq \mathcal{B}$	$\exists g \in R(\mathcal{B}) : \forall f \in R(\mathcal{A}) : \forall w \in \Sigma^\omega : \mathcal{A}^f(w) \leq \mathcal{B}^g(w)$
$\mathcal{A} \sqsubseteq \mathcal{B}$	$\forall w \in \Sigma^\omega : \forall f \in R(\mathcal{A}) : \exists g \in R(\mathcal{B}) : \mathcal{A}^f(w) \leq \mathcal{B}^g(w)$
$\mathcal{A} \trianglelefteq \mathcal{B}$	$\forall w \in \Sigma^\omega : \exists g \in R(\mathcal{B}) : \forall f \in R(\mathcal{A}) : \mathcal{A}^f(w) \leq \mathcal{B}^g(w)$
$\mathcal{A} \trianglelefteq_\times \mathcal{B}$	$\forall f \in R_1(\mathcal{A},\mathcal{B}) : \exists g \in R_2(\mathcal{A},\mathcal{B}) : \forall w \in \Sigma^\omega$ $(\mathcal{A} \times_1 \mathcal{B})^{\{f,g\}}(w) \leq (\mathcal{A} \times_2 \mathcal{B})^{\{f,g\}}(w)$
$\mathcal{A} \trianglelefteq_\times \mathcal{B}$	$\exists g \in R_2(\mathcal{A},\mathcal{B}) : \forall f \in R_1(\mathcal{A},\mathcal{B}) : \forall w \in \Sigma^\omega$ $(\mathcal{A} \times_1 \mathcal{B})^{\{f,g\}}(w) \leq (\mathcal{A} \times_2 \mathcal{B})^{\{f,g\}}(w)$
$\mathcal{A} \sqsubseteq_\times \mathcal{B}$	$\forall w \in \Sigma^\omega : \forall f \in R_1(\mathcal{A},\mathcal{B}) : \exists g \in R_2(\mathcal{A},\mathcal{B})$ $(\mathcal{A} \times_1 \mathcal{B})^{\{f,g\}}(w) \leq (\mathcal{A} \times_2 \mathcal{B})^{\{f,g\}}(w)$
$\mathcal{A} \trianglelefteq_\times \mathcal{B}$	$\forall w \in \Sigma^\omega : \exists g \in R_2(\mathcal{A},\mathcal{B}) : \forall f \in R_1(\mathcal{A},\mathcal{B})$ $(\mathcal{A} \times_1 \mathcal{B})^{\{f,g\}}(w) \leq (\mathcal{A} \times_2 \mathcal{B})^{\{f,g\}}(w)$

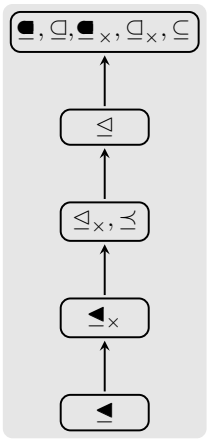

Figure 3 Left: The definitions of inclusion (denoted $\subseteq$), simulation (denoted $\preceq$), and the resolver relations we study in Section 3 including strategic dominance (denoted $\trianglelefteq$). Right: The implications between these relations as proved in Propositions 3.3–3.7 and Corollary 3.8.

We proceed to show the implications posed in Figure 3.

▶ **Proposition 3.5.** *Let $\nu_1, \nu_2 \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimSup}, \mathsf{LimInf}\}$. For all ν_1-automata $\mathcal{A}$ and all ν_2-automata $\mathcal{B}$, the following statements hold. Moreover, they are equally true for boolean safety, reachability, Büchi, and coBüchi automata.*

1. $\mathcal{A} \trianglelefteq \mathcal{B} \Rightarrow \mathcal{A} \subseteq \mathcal{B}$
2. $\mathcal{A} \preceq \mathcal{B} \Rightarrow \mathcal{A} \trianglelefteq \mathcal{B}$
3. $\mathcal{A} \trianglelefteq_\times \mathcal{B} \Rightarrow \mathcal{A} \preceq \mathcal{B}$
4. $\mathcal{A} \trianglelefteq \mathcal{B} \Rightarrow \mathcal{A} \trianglelefteq_\times \mathcal{B}$

The implications given in Figure 3 (and proved in Propositions 3.3–3.5) also hold when the problem statements are restricted to only finite-memory resolvers or positional resolvers.

▶ **Proposition 3.6.** *Let $\nu_1, \nu_2 \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimSup}, \mathsf{LimInf}\}$. For all ν_1-automata $\mathcal{A}$ and all ν_2-automata $\mathcal{B}$ and each $r \in \{fin, pos\}$ the following statements hold. Moreover, they are equally true for boolean safety, reachability, Büchi, and coBüchi automata.*

1. $\mathcal{A} \sqsubseteq^r \mathcal{B} \Leftrightarrow \mathcal{A} \trianglelefteq^r \mathcal{B} \Leftrightarrow \mathcal{A} \sqsubseteq^r_\times \mathcal{B} \Leftrightarrow \mathcal{A} \trianglelefteq^r_\times \mathcal{B} \Leftrightarrow \mathcal{A} \subseteq^r \mathcal{B}$
2. $\mathcal{A} \trianglelefteq^r_\times \mathcal{B} \Leftrightarrow \mathcal{A} \preceq^r \mathcal{B}$
3. $\mathcal{A} \trianglelefteq^r \mathcal{B} \Rightarrow \mathcal{A} \subseteq^r \mathcal{B}$
4. $\mathcal{A} \preceq^r \mathcal{B} \Rightarrow \mathcal{A} \trianglelefteq^r \mathcal{B}$
5. $\mathcal{A} \trianglelefteq^r_\times \mathcal{B} \Rightarrow \mathcal{A} \preceq^r \mathcal{B}$
6. $\mathcal{A} \trianglelefteq^r \mathcal{B} \Rightarrow \mathcal{A} \trianglelefteq^r_\times \mathcal{B}$

3.2 Separating Examples for Resolver Relations

In this part, we provide separating examples for the implications we proved above, establishing a hierarchy of relations given in Figure 3. The counter-examples we used in the proofs below are displayed in Figures 4 and 5.

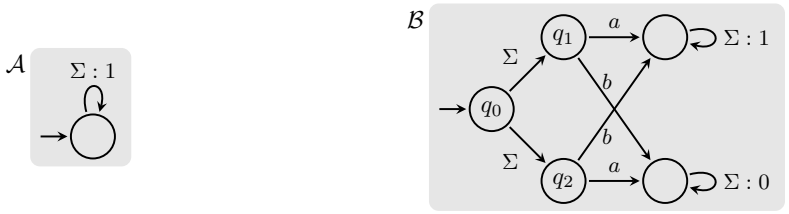

Figure 4 Two automata $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \subseteq^{\text{pos}} \mathcal{B}$ but $\mathcal{A} \not\trianglelefteq^{\text{pos}} \mathcal{B}$. Note that $\mathcal{A}$ and $\mathcal{B}$ can be Inf-, Sup-, LimSup-, or LimInf-automata as well as safety, reachability, Büchi, or coBüchi automata. The exact values of the omitted weights depend on the considered value function. For example, we can take as weight 1 for Inf and 0 for Sup while both choices work for LimSup and LimInf.

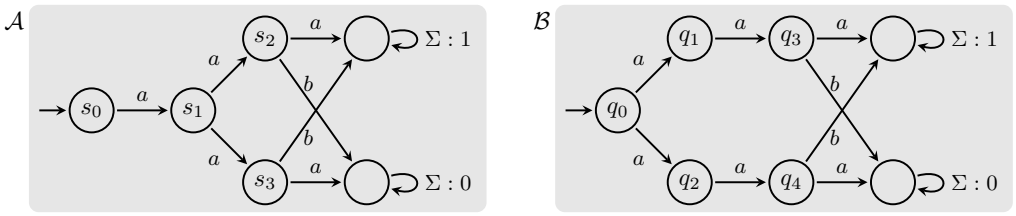

Figure 5 Two automata $\mathcal{A}$ and $\mathcal{B}$ such that (i) $\mathcal{A} \trianglelefteq^{\text{pos}} \mathcal{B}$ but $\mathcal{A} \not\trianglelefteq^{\text{pos}} \mathcal{B}$, (ii) $\mathcal{A} \preceq^{\text{pos}} \mathcal{A}$ but $\mathcal{A} \not\trianglelefteq^{\text{pos}}_{\times} \mathcal{A}$, and (iii) $\mathcal{B} \trianglelefteq^{\text{pos}}_{\times} \mathcal{A}$ but $\mathcal{B} \not\trianglelefteq^{\text{pos}} \mathcal{A}$. The transitions that are not shown lead to a sink state with a self-loop on every letter with weight 0. Note that $\mathcal{A}$ and $\mathcal{B}$ can be Inf-, Sup-, LimSup-, or LimInf-automata as well as safety, reachability, Büchi, or coBüchi automata. The exact values of the omitted weights depend on the considered value function. For example, we can take as weight 1 for Inf and 0 for Sup while both choices work for LimSup and LimInf.

▶ **Proposition 3.7.** *Let $\nu_1, \nu_2 \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimSup}, \mathsf{LimInf}\}$. For each statement below, there exist a ν_1-automaton $\mathcal{A}$ and a ν_2-automaton $\mathcal{B}$ to satisfy it. Moreover, they are equally true for boolean safety, reachability, Büchi, and coBüchi automata.*

1. $\mathcal{A} \subseteq^{pos} \mathcal{B} \wedge \mathcal{A} \not\trianglelefteq^{pos} \mathcal{B}$
2. $\mathcal{A} \trianglelefteq^{pos} \mathcal{B} \wedge \mathcal{A} \not\preceq^{pos} \mathcal{B}$
3. $\mathcal{A} \preceq^{pos} \mathcal{B} \wedge \mathcal{A} \not\trianglelefteq^{pos}_{\times} \mathcal{B}$
4. $\mathcal{A} \trianglelefteq^{pos}_{\times} \mathcal{B} \wedge \mathcal{A} \not\trianglelefteq^{pos} \mathcal{B}$

Since $R^{\text{pos}}(\mathcal{A}) = R^{\text{fin}}(\mathcal{A}) = R(\mathcal{A})$ for each automaton $\mathcal{A}$ we consider in this section (see Figures 4 and 5), the statements above also hold for the finite-memory and the general cases.

▶ **Corollary 3.8.** *Let $\nu_1, \nu_2 \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimSup}, \mathsf{LimInf}\}$. For each statement below, there exist a ν_1-automaton $\mathcal{A}$ and a ν_2-automaton $\mathcal{B}$ to satisfy it. Moreover, they are equally true for boolean safety, reachability, Büchi, and coBüchi automata.*

1. $\mathcal{A} \subseteq^{fin} \mathcal{B} \wedge \mathcal{A} \not\trianglelefteq^{fin} \mathcal{B}$
2. $\mathcal{A} \trianglelefteq^{fin} \mathcal{B} \wedge \mathcal{A} \not\preceq^{fin} \mathcal{B}$
3. $\mathcal{A} \preceq^{fin} \mathcal{B} \wedge \mathcal{A} \not\trianglelefteq^{fin}_{\times} \mathcal{B}$
4. $\mathcal{A} \trianglelefteq^{fin}_{\times} \mathcal{B} \wedge \mathcal{A} \not\trianglelefteq^{fin} \mathcal{B}$
5. $\mathcal{A} \subseteq \mathcal{B} \wedge \mathcal{A} \not\trianglelefteq \mathcal{B}$
6. $\mathcal{A} \trianglelefteq \mathcal{B} \wedge \mathcal{A} \not\preceq \mathcal{B}$
7. $\mathcal{A} \preceq \mathcal{B} \wedge \mathcal{A} \not\trianglelefteq_{\times} \mathcal{B}$
8. $\mathcal{A} \trianglelefteq_{\times} \mathcal{B} \wedge \mathcal{A} \not\trianglelefteq \mathcal{B}$

▶ Remark 3.9. The relations $\trianglelefteq$ and $\trianglelefteq_{\times}$ are not reflexive (and thus not a preorder). For the automaton $\mathcal{A}$ given in Figure 5, we have $\mathcal{A} \not\trianglelefteq^{\text{pos}}_{\times} \mathcal{A}$ as shown in the proof of Item (3) of Proposition 3.7, and thus $\mathcal{A} \not\trianglelefteq_{\times} \mathcal{A}$ since all of its resolvers are positional. Then, by Item (4) of Proposition 3.5, we also have $\mathcal{A} \not\trianglelefteq \mathcal{A}$.

4 Resolver Logic

We dedicate this section to describing resolver logic, which intuitively extends Presburger arithmetic by introducing variables evaluated by automata parameterized by quantified words and resolvers. We formally define resolver logic and show that the model-checking of a resolver logic formula is decidable.

▶ **Definition 4.1** (resolver logic). *Let $\mathsf{QA} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ be a finite set of automata over the same alphabet Σ. For all $k \in \{1, \ldots, n\}$, let F_k be a set of resolver variables ranging over $R(\mathcal{A}_k)$, and let $W \subseteq \Sigma^\omega$ be a set of word variables ranging over Σ^ω. We define the set $V = \{v_{(x,y)} \mid x \in W, y \in \bigcup_{k=1}^{n} F_k\}$ of variables ranging over non-negative integers. A* resolver logic formula *on the automata domain QA is a term generated by the grammar $\Psi ::= \exists x : \Psi \mid \forall x : \Psi \mid \varphi$, where $x \in W \cup \bigcup_{k=1}^{n} F_k$ and $\varphi \in \exists FO(\mathbb{N}, =, +, 1)$ is an existential Presburger formula whose set of free variables is V.*

We write $|\Psi|$ to denote the size of Ψ defined as $|\Psi| = n + m + |\varphi|$ where $n = |\mathsf{QA}|$ is the cardinality of the automata domain, $m = |W| + \sum_{i=1}^{n} |F_i|$ is the number of word and resolver variables in Ψ, and $|\varphi|$ is the number of (existential) quantifiers in φ. Note that the strategic dominance and the other relations defined in Section 3 are examples of resolver logic formulas. An assignment α maps variables of W to words in Σ^ω, variables of F_k to resolvers in $R(\mathcal{A}_k)$ for all $k \in \{1, \ldots, n\}$, and variables of $v \in V$ to values in $\mathbb{N}$. In particular, $\alpha(v_{(x,x')}) = \mathcal{A}_k^f(w)$ where $x \in W$ and $x' \in F_k$ such that $\alpha(x) = w \in \Sigma^\omega$ and $\alpha(x') = f \in R(\mathcal{A}_k)$. The semantics of Ψ is defined as follows.

$(\mathsf{QA}, \alpha) \models \varphi$ iff $\varphi[\forall v \in V : v \leftarrow \alpha(v)]$ holds
$(\mathsf{QA}, \alpha) \models \exists x \in W : \Psi$ iff for some $w \in \Sigma^\omega$ we have $\alpha[x \leftarrow w] \models \Psi$
$(\mathsf{QA}, \alpha) \models \forall x \in W : \Psi$ iff for all $w \in \Sigma^\omega$ we have $\alpha[x \leftarrow w] \models \Psi$
$(\mathsf{QA}, \alpha) \models \exists y \in F_k : \Psi$ iff for some $f \in R(\mathcal{A}_k)$ we have $\alpha[y \leftarrow f] \models \Psi$
$(\mathsf{QA}, \alpha) \models \forall y \in F_k : \Psi$ iff for all $f \in R(\mathcal{A}_k)$ we have $\alpha[y \leftarrow f] \models \Psi$

▶ **Theorem 4.2.** *The model-checking of a given resolver logic formula Ψ is decidable. When $|\Psi|$ is fixed, model-checking is in d-ExpTime if there are $d > 0$ quantifier alternations and PTime if there is no quantifier alternation.*

The decision procedure relies on the following: (1) Each variable assignment can be represented by a single tree. (2) Each resolver logic formula admits a parity tree automata that accepts all tree-encoded assignments that satisfy its inner Presburger formula. (3) The model-checking of a given resolver formula can be decided based on nested complementations and projections over the above parity tree automaton. Below we provide an overview of the corresponding constructions (the full proof is deferred to the appendix due to space constraints). We start by defining trees and describing to use them to encode assignments.

Trees. Let Σ be an alphabet for the structure of trees. We view the set Σ^* of finite words as the domain of an infinite $|\Sigma|$-ary tree. The root is the empty word ε, and for a node $u \in \Sigma^*$ together with some letter $\sigma \in \Sigma$ we call $u\sigma$ the σ-successor of u. Let Λ be an alphabet for the labeling of nodes. An infinite Λ-labeled Σ-structured tree is a function $t\colon \Sigma^* \to \Lambda$. We denote by Λ_Σ^ω the set of all such trees. In a tree $t \in \Lambda_\Sigma^\omega$, the word $w = \sigma_0\sigma_1 \cdots \in \Sigma^\omega$ induces the branch $t(w)$ defined as the infinite sequence $t(\varepsilon)\sigma_0 t(\sigma_0)\sigma_1 t(\sigma_0\sigma_1) \cdots \in (\Lambda \times \Sigma)^\omega$.

Let $\mathsf{QA} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ be a set of automata over the same alphabet Σ. Consider the resolver logic formula over QA of the form $\Psi = \nabla_1 x_1 : \cdots : \nabla_m x_m : \varphi$, where $\nabla_i \in \{\exists, \forall\}$. The decision procedure encodes assignments for resolver and word variables of Ψ into Λ-labeled Σ-structured trees, where $\Lambda = \left(\{0,1\} \cup \bigcup_{i=1}^{n} Q_n\right)^m$. All dimensions of the tree that correspond to a word must have exactly one branch labeled by 1 (which encodes the word), and all other nodes are labeled by 0. Formally, the assignment α_t encoded by a tree $t \in \Lambda_\Sigma^\omega$ maps the word variable $x_j \in W$ to the unique word $\alpha_t(x_j) = \sigma_1\sigma_2 \cdots \in \Sigma^\omega$ for which the jth dimension of the branch $t(\alpha_t(x_j))$ is $t_j(\alpha_t(x_j)) = 1\sigma_1 1\sigma_2 \cdots \in (\{1\} \times \Sigma)^\omega$. All dimensions of the tree that correspond to a resolver of $\mathcal{A}_k$ must respect its transition relation. Formally, the assignment α_t encoded by a tree $t \in \Lambda_\Sigma^\omega$ maps the resolver variable $x_i \in F_k$ to the unique resolver $\alpha_t(x_i) \in R(\mathcal{A}_k)$ defined by $\alpha_t(x_i)(\pi_k, \sigma) = t_i(u\sigma)$ where π_k is the finite run of $\mathcal{A}_k^{\alpha_t(x_i)}$ over $u \in \Sigma^*$. Consequently, for all $x_i \in F_k$ and all $x_j \in W$, the assignment α_t encoded by a tree $t \in \Lambda_\Sigma^\omega$ maps the variable $v_{(x_i,x_j)} \in V$ to the unique non-negative integer $\mathcal{A}_k^{\alpha_t(x_i)}(\alpha_t(x_j))$. Next, we describe the parity tree automaton constructed in the decision procedure.

Parity Tree Automata. A (nondeterministic) parity tree automaton $\mathcal{T}$ over Λ_Σ^ω is a tuple $(\Lambda, \Sigma, Q, I, \Delta, \theta)$ where Λ is a finite labeling alphabet, Σ is a finite structure alphabet, Q is a finite set of states, $I \subseteq Q$ is a set of initial states, $\Delta \subseteq Q \times \Lambda \times (\Sigma \to Q)$ is a transition relation, and $\theta \colon Q \to \mathbb{N}$ is the priority function. Note that the arity of the trees is $|\Sigma|$ and is statically encoded in the transition relation. A run of $\mathcal{T}$ over $t \in \Lambda_\Sigma^\omega$ is a Q-labeled Σ-structured tree $\pi \in Q_\Sigma^\omega$ such that $\pi(\varepsilon) \in I$ and for each $u \in \Sigma^*$ we have $(\pi(u), t(u), \sigma \mapsto \pi(u\sigma)) \in \Delta$. The set of runs of $\mathcal{A}$ over t is denoted $\Pi_t(\mathcal{T})$. A run π is accepting if, for all $w \in \Sigma^\omega$, the maximal priority that appears infinitely often along the branch $t(w)$, namely $\limsup_{i\to\infty} \theta(\pi(\sigma_0 \ldots \sigma_i))$, is even. The language of $\mathcal{A}$ is $T(\mathcal{A}) = \{t \in \Lambda_\Sigma^\omega \mid \pi \in \Pi_t(\mathcal{T}), \limsup_{i\to\infty} \theta(\pi(\sigma_0 \ldots \sigma_i)) \equiv 0 \mod 2\}$, i.e., the set of all trees that admit an accepting run.

Parity tree automata are expressive enough to recognize the language of tree-encoded assignments for a given resolver logic formula. We describe an automaton with three computational phases. In first phase, the automaton guesses its initial state. All states hold a vector $\vec{z}$ of m^2 weights appearing in $\mathcal{A}_1, \ldots, \mathcal{A}_n$ and satisfying φ, i.e., such that $\varphi[\forall v_{(x,x')} \in V : v_{(x,x')} \leftarrow \vec{z}[x][x']]$ is true. Such a vector is guessed at the root of the run tree and carried in all nodes thanks to the states. Since there are finitely many weights and free variables in φ, there are also finitely many vectors $\vec{z}$. In the second phase, the automaton "waits" for finitely many transitions. This is important for LimInf and LimSup automata, because the run of $\mathcal{A}_k^{\alpha_t(x_i)}$ over $\alpha_t(x_j)$ may visit finitely many times some weights independent of the long-run value. In the third phase, the automaton checks whether the guessed vector $\vec{z}$ is coherent with the tree-encoded assignment that it reads. Given a run π over the tree $t \in \Lambda_\Sigma^\omega$ that carries the vector $\vec{z}$, for all word variable $x_j \in W$ and all resolver variable $x_i \in F_k$, the value $\vec{z}[x_i][x_j]$ is a coherent assignment for the variable $v_{(x_i,x_j)} \in V$ when $\vec{z}[x_i][x_j] = \mathcal{A}_k^{\alpha_t(x_i)}(\alpha_t(x_j))$. Consider the assignment α_t given as an input tree $t \in \Lambda_\Sigma^\omega$. The run produced by the resolver $\alpha_t(x_i) \in R(\mathcal{A}_k)$ over the word $\alpha_t(x_j) \in \Sigma^\omega$ corresponds, by construction, to the branch $t_i(\alpha_t(x_j))$. To check $\vec{z}[x_i][x_j] = \mathcal{A}_k^{\alpha_t(x_i)}(\alpha_t(x_j))$ at runtime, the automaton ensures that: (1) the weight $\vec{z}[x_i][x_j]$ is visited infinitely often along the run $t_i(\alpha_t(x_j))$, and (2) it is never dismissed by another weight (e.g., for LimInf value function, the guessed weight should be the smallest visited after the waiting phase). Hence, the accepting condition of the automaton is such that, for all $x_j \in W$ and all $x_i \in F_k$, if the automaton accepts $t \in \Lambda_\Sigma^\omega$ then $\mathcal{A}_k^{\alpha_t(x_i)}(\alpha_t(x_j)) = \vec{z}[x_i][x_j]$. Next, we describe how the quantifiers of a resolver logic formula are handled base on the automaton corresponding to its inner Presburger formula.

Handling Quantifiers. To decide the model-checking of a resolver logic formula of fixed size, we first construct a parity tree automata as presented above. Its size is at most $\mathcal{O}(|\max_{1\leq i\leq n}|\mathcal{A}_i|)^{\mathcal{O}(m^2+n)}$. Since, the satisfiability of an existential Presburger formula with a fixed number of quantifiers is in PTime [27], the automaton can be constructed in polynomial time when $|\Psi|$ is fixed (i.e., n, m and $|\varphi|$ are fixed). Then, we release the existentially quantified variables through projections, i.e., leaving the automaton a non-deterministic choice while relaxing a dimension of the input tree. Universal quantifiers $\forall x : \Psi'$ are treated as $\neg\exists x : \neg\Psi'$, where each negation $\neg$ requires the complementation of the current tree automaton, and then induce an exponential blow up of the computation time [24]. Ultimately, we obtain a tree automaton that does not read labels and the model-checking of the resolver logic formula reduces to its language non-emptiness. It is worth emphasizing that, when a universal quantifier appears at the edge of the quantifier sequence, some complementations can be avoided (e.g., when all quantifiers are universal). When the innermost quantifier is universal, the automaton is constructed over $\neg\varphi$ instead of φ. When the outermost quantifier is universal, we leverage the parity acceptance condition of the tree automata to perform a final complementation in PTime. Formally, we increase all state priority by 1 and we check the language emptiness instead of non-emptiness.

▶ Remark 4.3. Resolver logic, as presented above, quantifies over non-partial resolvers (called *full* here to improve clarity). We can extend it to handle partial resolvers over product automata. The key observation is that, as long as a collection of partial resolvers is conclusive, they collectively define a full resolver. Moreover, partial resolvers over the components of product automata are conclusive by definition (see Remark 2.7). Hence, the parity tree automaton in the proof of Theorem 4.2 is constructed similarly. Some modifications are necessary to reason about the full resolver defined by the conclusive collection of partial resolvers, but the size of the parity tree automaton constructed above and the overall complexity will not change. This is because the partial resolvers are defined over a product of automata, which is already taken into account for full resolvers in the construction above.

To conclude, let us note that our construction allows checking inclusion or simulation at a high cost. For example, although checking the inclusion of two LimSup-automata is PSpace-complete, using an equivalent formulation from Proposition 3.3, we obtain a 2-ExpTime algorithm using the construction presented in this section. Similarly, while checking simulation for LimSup-automata can be done in NP∩co-NP, we obtain a 3-ExpTime algorithm using Proposition 3.4.

5 Applications of Resolver Logic

In this section, we explore various applications of resolver logic, highlighting its versatility in addressing problems in automata theory and system verification. We begin by presenting how resolver logic can be used for checking strategic dominance and other relations we studied in Section 3. Next, we examine its role in checking the bottom value of automata, which is a crucial problem for deciding co-safety and co-liveness of automata and was left open in [5]. Then, we explore its application for checking history-determinism of automata, and finally discuss its relevance in checking hyperproperty inclusion. As in the previous sections, we note that the results of this section also hold when we only consider boolean safety, reachability, Büchi, and coBüchi automata.

5.1 Checking Strategic Dominance and Other Resolver-Based Relations

Let $\mathcal{A}$ and $\mathcal{B}$ be two automata and recall that $\mathcal{B}$ strategically dominates $\mathcal{A}$, denoted $\mathcal{A} \trianglelefteq \mathcal{B}$, iff for all resolvers $f \in R(\mathcal{A})$ there exists a resolver $g \in R(\mathcal{B})$ such that $\mathcal{A}^f(w) \leq \mathcal{B}^g(w)$ for all words $w \in \Sigma^\omega$. Formulating this condition as a resolver logic formula, we obtain a 2-ExpTime algorithm for checking strategic dominance thanks to Theorem 4.2.

▶ **Corollary 5.1.** *Let $\nu_1, \nu_2 \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimSup}, \mathsf{LimInf}\}$. For all ν_1-automata $\mathcal{A}$ and all ν_2-automata $\mathcal{B}$, checking whether $\mathcal{A} \trianglelefteq \mathcal{B}$ can be done in 2-ExpTime.*

Note that other relations introduced in Section 3 can be also checked similarly.

5.2 Checking the Bottom Value of Automata

Safety and liveness [1, 2], as well as co-safety and co-liveness, are fundamental concepts in specification of system properties and their verification. These concepts have been recently extended to quantitative properties [21], and safety and liveness have been studied in the context of quantitative automata [5]. Note that quantitative automata resolve nondeterminism by sup, i.e., given an automaton $\mathcal{A}$ and a word w, we have $\mathcal{A}(w) = \mathcal{A}_{\sup}(w)$.

For deciding the safety or liveness of a given automaton $\mathcal{A}$, computing its *top value*, namely the value of $\top_{\mathcal{A}} = \sup_{w \in \Sigma^\omega} \sup_{f \in R(\mathcal{A})} \mathcal{A}^f(w)$, is shown to be a central step. The PTime algorithm provided in [5] is used as a subroutine for computing the safety closure of a given automaton $\mathcal{A}$, which is used for checking both safety and liveness of $\mathcal{A}$. In particular, $\mathcal{A}$ is safe iff the safety closure of $\mathcal{A}$ maps every word w to the same value as $\mathcal{A}$, and $\mathcal{A}$ is live iff the safety closure of $\mathcal{A}$ maps every word w to $\top_{\mathcal{A}}$.

Given an automaton $\mathcal{A}$, we can solve the top-value problem by simply iterating over its weights in decreasing order and checking for each weight k whether there exist $f \in R(\mathcal{A})$ and $w \in \Sigma^\omega$ with $\mathcal{A}^f(w) \geq k$. The largest k for which this holds is the top value of $\mathcal{A}$. Note that thanks to Theorem 4.2 we can achieve this in PTime as it is only an existential formula, which gives us a new algorithm for this problem.

However, the problems of deciding the co-safety and co-liveness of automata were left open. For these, one needs to compute *bottom value* of a given automaton $\mathcal{A}$, namely $\bot_{\mathcal{A}} = \inf_{w \in \Sigma^\omega} \sup_{f \in R(\mathcal{A})} \mathcal{A}^f(w)$. Similarly as above, using the computation of the bottom value of $\mathcal{A}$ as a subroutine, we can decide its co-safety and co-liveness: $\mathcal{A}$ is co-safe iff the co-safety closure of $\mathcal{A}$ maps every word w to the same value as $\mathcal{A}$, and $\mathcal{A}$ is co-live iff the co-safety closure of $\mathcal{A}$ maps every word w to $\bot_{\mathcal{A}}$.

For the classes of automata we consider, we can compute the bottom value in PSpace by repeated universality checks over its finite set of weights: the largest weight k for which the automaton is universal is its bottom value. We remark that the bottom value of limit-average automata is uncomputable since their universality is undecidable [16, 10].

Together with Theorem 4.2, the theorem below provides us with a 2-ExpTime algorithm for computing the bottom value of Inf-, Sup-, LimInf-, and LimSup-automata.

▶ **Theorem 5.2.** *Let $\nu \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimSup}, \mathsf{LimInf}\}$. Let $\mathcal{A}$ be a ν-automaton and x be an integer. Then, the bottom value of $\mathcal{A}$ is x iff $\exists w_1 \in \Sigma^\omega : \exists f_1 \in R(\mathcal{A}) : \forall w_2 \in \Sigma^\omega : \forall f_2 \in R(\mathcal{A}) : \exists f_3 \in R(\mathcal{A}) : \mathcal{A}^{f_1}(w_1) = x \wedge \mathcal{A}^{f_1}(w_1) \geq \mathcal{A}^{f_2}(w_1) \wedge \mathcal{A}^{f_1}(w_1) \leq \mathcal{A}^{f_3}(w_2)$. Moreover, given $\mathcal{A}$ and x, this can be checked in 2-ExpTime.*

5.3 Checking History-Determinism of Automata

History-determinism [22, 15] lies between determinism and nondeterminism. Intuitively, an automaton is history-deterministic if there exists a way of resolving its nondeterminism based on the current execution prefix (i.e., only the past) while ensuring that the value of the

resulting run equals the value assigned to the word by resolving its nondeterminism by sup. Although the concept of history-determinism first appeared as "good-for-gameness" in [22], following the distinction made in [6], we use the definition of history-determinism in [15].

▶ **Definition 5.3** (history-determinism [15]). *Let $\mathcal{A} = (\Sigma, Q, s, \Delta, \mu, \nu)$ be an automaton. Then, $\mathcal{A}$ is* history-deterministic *iff Player-2 wins the letter game defined below.*

The letter game *on $\mathcal{A}$ is played as follows: The game begins on the initial state $q_0 = s$. At each turn $i \geq 0$ that starts in a state q_i, Player-1 first chooses a letter $\sigma_i \in \Sigma$, then Player-2 chooses a transition $d_i = (q_i, \sigma_i, q_{i+1}) \in \Delta$, and the game proceeds to state q_{i+1}. The corresponding infinite play is an infinite run π over the word $w = \sigma_0\sigma_1 \ldots$, and Player-2 wins the game iff $\mathcal{A}_{\sup}(w) \leq \nu(\mu(\pi))$.*

▶ Remark 5.4. An automaton $\mathcal{A}$ is history-deterministic iff there exists a resolver $f \in R(\mathcal{A})$ such that $\mathcal{A}_{\sup}(w) \leq \mathcal{A}^f(w)$ for all $w \in \Sigma^\omega$. One can verify that the resolver f is exactly the winning strategy for Player-2 in the letter game on $\mathcal{A}$.

History-deterministic automata offer a balance between deterministic and nondeterministic counterparts, with notable advantages. For instance, history-deterministic LimInf-automata are exponentially more concise than deterministic ones [23], and history-deterministic pushdown automata exhibit both increased expressiveness and at least exponential succinctness compared to their deterministic counterparts [19]. Further exploration is detailed in [8, 7].

In [7], algorithms are presented to determine whether an automaton is history-deterministic. The approach involves solving a token game that characterizes history-determinism for the given automata type. The procedure is in PTime for Inf- and Sup-automata, quasipolynomial time for LimSup, and ExpTime for LimInf. Combined with Theorem 4.2, the theorem below presents a new ExpTime algorithm for checking history-determinism across all these automata types, providing competitive complexity with [7] for LimInf-automata.

▶ **Theorem 5.5.** *Let $\nu \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimSup}, \mathsf{LimInf}\}$ and $\mathcal{A}$ be a ν-automaton. Then, $\mathcal{A}$ is history-deterministic iff $\mathcal{A} \blacktriangleleft \mathcal{A}$. Moreover, given $\mathcal{A}$, this can be checked in ExpTime.*

5.4 Checking Hyperproperty Inclusion

We have focused on trace properties – functions mapping words to values, either 0 or 1 in the boolean setting. While adept at representing temporal event orderings, trace properties lack the capacity to capture dependencies among multiple system executions, such as noninterference in security policies [18] or fairness conditions for learning-based systems [28].

This limitation is addressed by hyperproperties [14]. Unlike trace properties, hyperproperties encompass global characteristics applicable to sets of traces. This enables the specification of intricate relationships and constraints beyond temporal sequencing. Formally, while a trace property is a set of traces, a hyperproperty is a set of trace properties.

In this subsection, we use nondeterministic automata as a specification language for hyperproperties. A deterministic automaton defines a trace property where each word has a single run, yielding a unique value. In contrast, a nondeterministic automaton specifies a trace property only when equipped with a resolver, representing a function from its resolvers to trace properties. Formally, a nondeterministic automaton $\mathcal{A}$ specifies the hyperproperty $H_\mathcal{A} = \{\mathcal{A}^f \mid f \in R(\mathcal{A})\}$. An illustrative example is presented in Figure 6 and Proposition 5.6.

▶ **Proposition 5.6.** *The nondeterministic automata $\mathcal{A}$ and $\mathcal{B}$ in Figure 6 respectively specify the hyperproperties $\mathsf{SP} = \{P \subseteq \Sigma^\omega \mid P \text{ is safe}\}$ and $\mathsf{CP} = \{P \subseteq \Sigma^\omega \mid P \text{ is co-safe}\}$.*

Figure 6 Two nondeterministic automata $\mathcal{A}$ and $\mathcal{B}$ over a finite alphabet Σ that respectively specify the hyperproperties $\mathsf{SP} = \{P \subseteq \Sigma^\omega \mid P \text{ is safe}\}$ and $\mathsf{CP} = \{P \subseteq \Sigma^\omega \mid P \text{ is co-safe}\}$.

HyperLTL [13] extends linear temporal logic (LTL) only with quantification over traces, and therefore cannot express the hyperproperty SP specifying the set of all safety trace properties. However, using HyperLTL over the alphabet $\Sigma = \{i, s, o, x\}$, one can express the noninterference between a secret input s and a public output o as follows: $\forall \pi, \pi' : \square(i_\pi \leftrightarrow i_{\pi'}) \rightarrow \square(o_\pi \leftrightarrow o_{\pi'})$, i.e., for every pair π, π' of traces, if the positions of the public input i coincide in π and π', then so do the positions of the public output o. We show below that a simpler variant of this property cannot be specified by nondeterministic automata, separating them as a specification language for hyperproperties from HyperLTL.

▶ **Proposition 5.7.** *Let $\Sigma = \{a, b, c\}$ and let $\phi = \forall \pi, \pi' : \square(b_\pi \leftrightarrow b_{\pi'})$ be a HyperLTL formula. Neither $H_1 = \{P \subseteq \Sigma^\omega \mid P \text{ satisfies } \phi\}$ nor $H_2 = \{P \subseteq \Sigma^\omega \mid P \text{ satisfies } \neg\phi\}$ is expressible by an automaton.*

Together with Theorem 4.2, the following theorem gives us a 2-ExpTime algorithm for checking if the hyperproperty specified by an automaton is included in another one.

▶ **Theorem 5.8.** *Let $\mathcal{A}$ and $\mathcal{B}$ be two nondeterministic automata respectively denoting the hyperproperties $H_\mathcal{A}$ and $H_\mathcal{B}$. Then, $H_\mathcal{A} \subseteq H_\mathcal{B}$ iff $\forall f \in R(\mathcal{A}) : \exists g \in R(\mathcal{B}) : \forall w \in \Sigma^\omega : \mathcal{A}^f(w) = \mathcal{B}^g(w)$. Moreover, given $\mathcal{A}$ and $\mathcal{B}$, this can be checked in 2-ExpTime.*

Using $\mathcal{A}$ for a deterministic automaton representing a system and $\mathcal{B}$ for a nondeterministic automaton defining a hyperproperty, we solve the model checking problem by determining if there exists $g \in R(\mathcal{B})$ such that $\mathcal{A}(w) = \mathcal{B}^g(w)$ for all $w \in \Sigma^\omega$. This is solvable in ExpTime (Theorem 4.2). Notably, for HyperLTL, which is incomparable to nondeterministic automata as a specification language, the complexity is PSpace-hard in the system's size [13].

6 Conclusion

We introduced a novel perspective on model refinement, termed *strategic dominance*. This view, which falls between trace inclusion and tree inclusion, captures the relationship between two nondeterministic state-transition models by emphasizing the ability of the less precise model to accommodate all deterministic implementations of the more refined one. We formally defined strategic dominance and showed that it can be checked in 2-ExpTime using *resolver logic* – a decidable extension of Presburger logic we developed in this work. Resolver logic is a powerful tool for reasoning about nondeterministic boolean and quantitative finite automata over infinite words. We provided a model-checking algorithm for resolver logic which, besides the verification of resolver-based refinement relations such as strategic dominance, allows the checking of co-safety, co-liveness, and history-determinism of quantitative automata, and the inclusion of hyperproperties specified by nondeterministic automata. There are some problems we have left open, including the study of resolver logic for other value functions as well as lower bounds for the model-checking problem of resolver logic and its fragments. Future research should also extend resolver logic and its model-checking algorithm to handle the settings of partial information, of multiple agents, and of probabilistic strategies.

References

1 Bowen Alpern and Fred B. Schneider. Defining liveness. *Inf. Process. Lett.*, 21(4):181–185, 1985. doi:10.1016/0020-0190(85)90056-0.

2 Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Comput.*, 2(3):117–126, 1987. doi:10.1007/BF01782772.

3 Rajeev Alur, Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. Alternating refinement relations. In Davide Sangiorgi and Robert de Simone, editors, *CONCUR '98: Concurrency Theory, 9th International Conference, Nice, France, September 8-11, 1998, Proceedings*, volume 1466 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 1998. doi:10.1007/BFB0055622.

4 Ralph-Johan Back and Joakim von Wright. *Refinement Calculus - A Systematic Introduction*. Graduate Texts in Computer Science. Springer, 1998. doi:10.1007/978-1-4612-1674-2.

5 Udi Boker, Thomas A. Henzinger, Nicolas Mazzocchi, and N. Ege Saraç. Safety and liveness of quantitative automata. In Guillermo A. Pérez and Jean-François Raskin, editors, *34th International Conference on Concurrency Theory, CONCUR 2023, September 18-23, 2023, Antwerp, Belgium*, volume 279 of *LIPIcs*, pages 17:1–17:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CONCUR.2023.17.

6 Udi Boker and Karoliina Lehtinen. History determinism vs. good for gameness in quantitative automata. In Mikolaj Bojanczyk and Chandra Chekuri, editors, *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15-17, 2021, Virtual Conference*, volume 213 of *LIPIcs*, pages 38:1–38:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.FSTTCS.2021.38.

7 Udi Boker and Karoliina Lehtinen. Token games and history-deterministic quantitative-automata. *Log. Methods Comput. Sci.*, 19(4), 2023. doi:10.46298/LMCS-19(4:8)2023.

8 Udi Boker and Karoliina Lehtinen. When a little nondeterminism goes a long way: An introduction to history-determinism. *ACM SIGLOG News*, 10(1):24–51, 2023. doi:10.1145/3584676.3584682.

9 Stephen Brookes. A semantics for concurrent separation logic. *Theor. Comput. Sci.*, 375(1-3):227–270, 2007. doi:10.1016/J.TCS.2006.12.034.

10 Krishnendu Chatterjee, Laurent Doyen, Herbert Edelsbrunner, Thomas A. Henzinger, and Philippe Rannou. Mean-payoff automaton expressions. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, volume 6269 of *Lecture Notes in Computer Science*, pages 269–283. Springer, 2010. doi:10.1007/978-3-642-15375-4_19.

11 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4):23:1–23:38, 2010. doi:10.1145/1805950.1805953.

12 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy logic. *Inf. Comput.*, 208(6):677–693, 2010. doi:10.1016/J.IC.2009.07.004.

13 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8414 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2014. doi:10.1007/978-3-642-54792-8_15.

14 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.

15 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th Internatilonal Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 139–150. Springer, 2009. doi:10.1007/978-3-642-02930-1_12.

16 Aldric Degorre, Laurent Doyen, Raffaella Gentilini, Jean-François Raskin, and Szymon Torunczyk. Energy and mean-payoff games with imperfect information. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2010. doi:10.1007/978-3-642-15205-4_22.

17 Nathanaël Fijalkow, Nathalie Bertrand, Patricia Bouyer-Decitre, Romain Brenguier, Arnaud Carayol, John Fearnley, Hugo Gimbert, Florian Horn, Rasmus Ibsen-Jensen, Nicolas Markey, Benjamin Monmege, Petr Novotný, Mickael Randour, Ocan Sankur, Sylvain Schmitz, Olivier Serre, and Mateusz Skomra. Games on graphs. *CoRR*, abs/2305.10546, 2023. doi:10.48550/arXiv.2305.10546.

18 Joseph A. Goguen and José Meseguer. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy, Oakland, CA, USA, April 26-28, 1982*, pages 11–20. IEEE Computer Society, 1982. doi:10.1109/SP.1982.10014.

19 Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. A bit of nondeterminism makes pushdown automata expressive and succinct. *Log. Methods Comput. Sci.*, 20(1), 2024. doi:10.46298/LMCS-20(1:3)2024.

20 Thomas A. Henzinger, Orna Kupferman, and Sriram K. Rajamani. Fair simulation. *Inf. Comput.*, 173(1):64–81, 2002. doi:10.1006/INCO.2001.3085.

21 Thomas A. Henzinger, Nicolas Mazzocchi, and N. Ege Saraç. Quantitative safety and liveness. In Orna Kupferman and Pawel Sobocinski, editors, *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13992 of *Lecture Notes in Computer Science*, pages 349–370. Springer, 2023. doi:10.1007/978-3-031-30829-1_17.

22 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, volume 4207 of *Lecture Notes in Computer Science*, pages 395–410. Springer, 2006. doi:10.1007/11874683_26.

23 Denis Kuperberg and Michal Skrzypczak. On determinisation of good-for-games automata. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2015. doi:10.1007/978-3-662-47666-6_24.

24 Christof Löding. Automata on infinite trees. In Jean-Éric Pin, editor, *Handbook of Automata Theory*, pages 265–302. European Mathematical Society Publishing House, Zürich, Switzerland, 2021. doi:10.4171/AUTOMATA-1/8.

25 Fabio Mogavero, Aniello Murano, and Moshe Y. Vardi. Reasoning about strategies. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPIcs*, pages 133–144. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPICS.FSTTCS.2010.133.

26 Peter W. O'Hearn. Resources, concurrency, and local reasoning. *Theor. Comput. Sci.*, 375(1-3):271–307, 2007. doi:10.1016/J.TCS.2006.12.035.

27 Bruno Scarpellini. Complexity of subcases of presburger arithmetic. *Transactions of the American Mathematical Society*, 284(1):203–218, 1984. URL: http://www.jstor.org/stable/1999283.

28 Sanjit A. Seshia, Ankush Desai, Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Sumukh Shivakumar, Marcell Vazquez-Chanlatte, and Xiangyu Yue. Formal specification for deep neural networks. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 20–34. Springer, 2018. doi:10.1007/978-3-030-01090-4_2.

29 Rob J. van Glabbeek. The linear time - branching time spectrum II. In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1993. doi:10.1007/3-540-57208-2_6.

30 Rob J. van Glabbeek. The linear time - branching time spectrum I. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. North-Holland / Elsevier, 2001. doi:10.1016/B978-044482830-9/50019-9.

A Appendix

Proof of Theorem 4.2. We assume that each automata $\mathcal{A}_k = (Q_k, s_k, \Delta_k, \mu_k, \nu_k)$ is either a LimSup-automaton or a LimInf-automaton. This is without loss of generality since Sup-automata and Inf-automata can be converted in PTime into LimInf-automata [5, Proposition 2.1]. The proof goes as follows. First, we construct in polynomial time a parity tree automaton $\mathcal{C}$ which read an assignment α for Ψ as input, such that its language is empty if and only if $\alpha \models \Psi$. Then we handle the quantifiers of Ψ based on nested complementations and projections applied on $\mathcal{C}$. Finally, we construct a parity game such that the even player wins iff Ψ is satisfiable over $\mathcal{A}_1, \ldots, \mathcal{A}_n$.

Construction of $\mathcal{C}$. Let Ψ be of the form $\nabla_1 x_1 : \cdots : \nabla_m x_m : \varphi$, where $\nabla_i \in \{\exists, \forall\}$. In this construction, we encode assignments for resolver and word variables of Ψ into single Σ-structured trees. The labeling alphabet is defined from the sets $Q_1, \ldots, Q_n$ and $\{0, 1\}$ in order to manipulate branches as runs. For all $i \in \{1, \ldots, m\}$, we define Λ_i and $\ell_i \in \Lambda_i$ such that if $x_i \in W$ then $\Lambda_i = \{0, 1\}$ and $\ell_i = 1$; otherwise, $x_i \in F_k$ for some $k \in \{1, \ldots, n\}$, and so $\Lambda_i = Q_k$ and $\ell_i = s_k$ where s_k is the initial state of $\mathcal{A}_k$. Let the labeling alphabet be $\Lambda = \Lambda_1 \times \cdots \times \Lambda_m$ and let the label of roots be $\ell = \ell_1 \times \cdots \times \ell_m$. For all $\lambda \in \Lambda$ and $1 \leq i \leq m$ we write $\lambda[i]$ to denote the dimension of λ corresponding to Λ_i. In the same way, we construct the value domains of the variables of V from the sets of weights of $\mathcal{A}_1, \ldots, \mathcal{A}_k$. For all $x_i \in F_k$ and all $x_j \in W$, we define the value domain of the variable $v_{(x_i,x_j)} \in V$ as $Z_{(x_i,x_j)} = \{\mu_k(\delta) \in \mathbb{N} \mid \delta \in \Delta_k\}$. Let $Z = \Pi_{k=1}^{n} \Pi_{f \in F_k} \Pi_{w \in W} Z_{(f,w)}$ be the set of assignment of the variable of V. For all $z \in Z$, all $x_i \in F_k$ and all $x_j \in W$, we write $z[x_i][x_j]$ to denote the dimension of z corresponding to $Z_{(f,w)}$.

We now construct the parity tree automaton $\mathcal{C} = (\Lambda, \Sigma, Q, I, \Delta, \theta)$. The set of $\heartsuit$-states is $Q_\heartsuit = \{(\heartsuit_{(y_1,y_2)}, z, \lambda) \mid y_1, y_2 \in \{1, \ldots, m\}, z \in Z, \lambda \in \Lambda\}$, the set of $\spadesuit$-states is $Q_\spadesuit = \{(\spadesuit_{(y_1,y_2)}, z, \lambda) \mid y_1, y_2 \in \{1, \ldots, m\}, z \in Z, \lambda \in \Lambda\}$, and the set of $\bot$-states is $Q_\bot = \{(\bot, z, \lambda) \mid z \in Z, \lambda \in \Lambda\}$. The set of states is $Q = Q_\bot \cup Q_\heartsuit \cup Q_\spadesuit$ and the set of initial states is $I = \{(\bot, z, \ell) \mid z \in Z, \varphi[\forall v_{(x,x')} \in V : v_{(x,x')} \leftarrow z[x][x']]\}$. The priority function $\theta \colon Q \to \{1, 2\}$ maps $\heartsuit$-states to 2 and all the other states to 1. The transition relation Δ is defined as follows.

- $((\bot, z, \lambda), \lambda, \sigma \mapsto (S_\sigma, z, \lambda_\sigma)) \in \Delta$ where $S_\sigma \in \{\bot, \spadesuit_{(1,1)}\}$ iff

$$\bigwedge \begin{cases} \bigwedge_{j=1}^{m} \Big(x_j \in W \Rightarrow \sum_{\sigma \in \Sigma} \lambda_\sigma[j] = \lambda[j] \Big) \\ \bigwedge_{i=1}^{m} \bigwedge_{k=1}^{n} \bigwedge_{\sigma \in \Sigma} \Big(x_i \in F_k \Rightarrow (\lambda[i], \sigma, \lambda_\sigma[i]) \in \Delta_k \Big) \end{cases}$$

For all transitions, $\mathcal{C}$ ensures that the encoding of its assignment for $x_1, \ldots, x_m$ is a coherent Λ-labeled Σ-structured tree. Above, the first constraint guarantees that all dimensions encoding a word have exactly one branch labeled by 1 (which encodes the word), and all other nodes are labeled by 0. Formally, each tree $t \in T(\mathcal{C})$ assigns the variable $x_j \in W$ to the unique word $\alpha_t(x_j) = \sigma_1 \sigma_2 \cdots \in \Sigma^\omega$ for which the jth dimension of the branch $t(\alpha_t(x_j))$ equals $1\sigma_1 1\sigma_2 \cdots \in (\{1\} \times \Sigma)^\omega$. The second constraint guarantees that all dimensions

encoding a resolver of $\mathcal{A}_k$ respect its transition relation, i.e., a node labeled by λ and its σ-child labeled by λ_σ must encode a transition of $\mathcal{A}_k$ in these dimensions. Formally, each tree $t \in T(\mathcal{C})$ assigns the variable $x_i \in F_k$ to the unique resolver $\alpha_t(x_i) \in R(\mathcal{A}_k)$ defined by $\alpha_t(x_i)(\pi_k, \sigma) = t(u\sigma)$ where π_k is the finite run of $\mathcal{A}_k^{\alpha_t(x_i)}$ over $u \in \Sigma^*$. In particular, for all $t \in T(\mathcal{C})$, all $x_i \in F_k$ and $x_j \in W$, the value $\mathcal{A}_k^{\alpha_t(x_i)}(\alpha_t(x_j))$ is the correct assignment for the free variable $v_{(x_i,x_j)} \in V$ of φ.

- $((\spadesuit_{(y_1,y_2)}, z, \lambda), \lambda, \sigma \mapsto (\spadesuit_{(y_1,y_2)}, z, \lambda_\sigma)) \in \Delta$ iff

$$\bigwedge \begin{cases} \bigwedge_{j=1}^{m} \Big(x_j \in W \Rightarrow \sum_{\sigma\in\Sigma} \lambda_\sigma[j] = \lambda[j] \Big) \\ \bigwedge_{i=1}^{m} \bigwedge_{k=1}^{n} \bigwedge_{\sigma\in\Sigma} \Big(x_i \in F_k \Rightarrow (\lambda[i], \sigma, \lambda_\sigma[i]) \in \Delta_k \Big) \\ \bigwedge_{i=1}^{m} \bigwedge_{k=1}^{n} \bigwedge_{j=1}^{m} \bigwedge_{\sigma\in\Sigma} \Big(\big(x_i \in F_k \wedge \lambda[j] = 1\big) \Rightarrow h_k(z[x_i][x_j], \mu_k(\lambda[i], \sigma, \lambda_\sigma[i])) = z[x_i][x_j] \Big) \end{cases}$$

where $h_k = \max$ if $\mathcal{A}_k$ is a LimSup-automaton and $h_k = \min$ if it is a LimInf-automaton.

Observe that $\perp$-states are reachable only from $\perp$-states and cannot lead to acceptance as their priority is odd. Once a $\spadesuit_{(1,1)}$-state is reached, $\mathcal{C}$ checks through the rest of the run tree whether z provides a correct assignment for the variable of V. By construction, z is guessed at the root of the run tree and carried in all its nodes. Given a run π of $\mathcal{C}$ over $t \in \Lambda_\Sigma^\omega$ that carries $z \in Z$, for all $x_j \in W$, all $x_i \in F_k$, the value $z[x_i][x_j]$ is a correct assignment for $v_{(x_i,x_j)}$ when $z[x_i][x_j] = \mathcal{A}_k^{\alpha_t(x_i)}(\alpha_t(x_j))$. Intuitively, $v_{(x_i,x_j)}$ requires $\mathcal{C}$ to ensure that the weight $z[x_i][x_j]$ is (†) visited infinitely often, (‡) never dismissed by another weight, and so along the branch induced by the word $\alpha_t(x_j)$. The condition (†) is handled by $\mathcal{C}$ thanks to its acceptance condition that we explain below. Above, the last constraint guarantees the condition (‡), i.e., assuming that $x_j \in W$, $x_i \in R(\mathcal{A}_k)$ and that the current node belongs to the branch induced by $\alpha_t(x_j)$, if $\mathcal{A}_k$ is a LimSup-automaton then the weight of the transition $(\lambda[i], \sigma, \lambda_\sigma[i]) \in \Delta_k$ from the node to its σ-child is at most $z[x_i][x_j]$, otherwise $\mathcal{A}_k$ is a LimInf-automaton and the weight of this transition is at least $z[x_i][x_j]$. This constrain appears in all transitions outgoing from the $\spadesuit$-states and the $\heartsuit$-states.

- $((\spadesuit_{(y_1,y_2)}, z, \lambda), \lambda, \sigma \mapsto (S_{\sigma(y_1,y_2)}, z, \lambda_\sigma)) \in \Delta$ where $S_{\sigma(y_1,y_2)} \in \{\heartsuit_{(y_1,y_2)}, \spadesuit_{(y_1,y_2)}\}$ iff

$$\bigwedge \begin{cases} \bigwedge_{j=1}^{m} \Big(x_j \in W \Rightarrow \sum_{\sigma\in\Sigma} \lambda_\sigma[j] = \lambda[j] \Big) \\ \bigwedge_{i=1}^{m} \bigwedge_{k=1}^{n} \bigwedge_{\sigma\in\Sigma} \Big(x_i \in F_k \Rightarrow (\lambda[i], \sigma, \lambda_\sigma[i]) \in \Delta_k \Big) \\ \bigwedge_{i=1}^{m} \bigwedge_{k=1}^{n} \bigwedge_{j=1}^{m} \bigwedge_{\sigma\in\Sigma} \Big(\big(x_i \in F_k \wedge \lambda[j] = 1\big) \Rightarrow h_k(z[x_i][x_j], \mu_k(\lambda[i], \sigma, \lambda_\sigma[i])) = z[x_i][x_j] \Big) \\ \bigwedge_{k=1}^{n} \bigwedge_{\sigma\in\Sigma} \Big(\big(x_{y_1} \in F_k \wedge \lambda[y_2] = 1 \wedge S_{\sigma(y_1,y_2)} = \heartsuit_{(y_1,y_2)}\big) \Rightarrow \mu_k(\lambda[y_1], \sigma, \lambda_\sigma[y_1]) = z[x_{y_1}][x_{y_2}] \Big) \end{cases}$$

Given a run π of $\mathcal{C}$ over $t \in \Lambda_\Sigma^\omega$ that carries $z \in Z$, assuming that $x_{y_2} \in W$ and $x_{y_1} \in F_k$, the condition (†) asks $\mathcal{C}$ to check whether the guessed value $z[x_{y_1}][x_{y_2}]$ is among the values visited infinitely many times along the branch induced by $\alpha_t(x_{y_2}) \in \Sigma^\omega$. Above, the last constraint guarantees that $\mathcal{C}$ allows to move from a $\spadesuit_{(y_1,y_2)}$-state to a $\heartsuit_{(y_1,y_2)}$-state only if either $x_{y_1} \notin R(\mathcal{A}_k)$, or the current node does not belong to the branch induced by $\alpha_t(x_{y_2}) \in \Sigma^\omega$, or the guessed weight $z[x_{y_1}][x_{y_2}]$ is visited in the corresponding dimension. Observe that $\heartsuit$-states have priority 2 in $\mathcal{C}$, while $\spadesuit$-states have priority 1. The condition (†) on $z[x_{y_1}][x_{y_2}]$ holds for all accepting runs because $\mathcal{C}$ ensures that a $\heartsuit_{(y_1,y_2)}$-state is visited infinitely many times on all branches of its accepting runs, as we explain below.

- $((\heartsuit_{(y_1,y_2)}, z, \lambda), \lambda, \sigma \mapsto (\spadesuit_{(y'_1,y'_2)}, z, \lambda_\sigma)) \in \Delta$ iff

$$\bigwedge \begin{cases} \bigwedge_{j=1}^{m} \Big(x_j \in W \Rightarrow \sum_{\sigma\in\Sigma} \lambda_\sigma[j] = \lambda[j] \Big) \\ \bigwedge_{i=1}^{m} \bigwedge_{k=1}^{n} \bigwedge_{\sigma\in\Sigma} \Big(x_i \in F_k \Rightarrow (\lambda[i], \sigma, \lambda_\sigma[i]) \in \Delta_k \Big) \\ \bigwedge_{i=1}^{m} \bigwedge_{k=1}^{n} \bigwedge_{j=1}^{m} \bigwedge_{\sigma\in\Sigma} \Big(\big(x_i \in F_k \wedge \lambda[j] = 1 \big) \Rightarrow h_k(z[x_i][x_j], \mu_k(\lambda[i], \sigma, \lambda_\sigma[i])) = z[x_i][x_j] \Big) \\ (y'_1 = y_1 \wedge y'_2 = y_2 + 1) \vee (y'_1 = y_1 + 1 \wedge y_2 = m \wedge y'_2 = 1) \vee (y_1 = y_2 = m \wedge y'_1 = y'_2 = 1) \end{cases}$$

We recall that $\mathcal{O}(m^2)$ values are checked by $\mathcal{C}$ through its runs. To ensure that condition (†) holds for all dimensions of z, the transitions of $\mathcal{C}$ enforces to visit cyclically all $\heartsuit_{(y_1,y_2)}$-states in order to get a run of even priority. Above, the last constraint guarantees that $\mathcal{C}$ allows to leave a $\heartsuit_{(y_1,y_2)}$-state only toward a $\spadesuit$-state that regulates the next pair of index. Since there is no transition from a $\heartsuit$-state to a $\heartsuit$-state, a run of $\mathcal{C}$ is accepting if and only if all branches visit a $\heartsuit_{(y_1,y_2)}$-state infinitely often for all $y_1, y_2 \in \{1, \ldots, m\}$. As final observation, we point out that, since z carried in all nodes of the run tree, the consistency of (†) and (‡) through branches is guaranteed. Hence, for all $t \in T(\mathcal{C})$, if $x_j \in W$ and $x_i \in F_k$ then the value $\mathcal{A}_k^{\alpha_t(x_i)}(\alpha_t(x_j))$ equals $z[x_i][x_j]$ thanks to the conditions (†) and (‡).

Construction of the parity game. Note that the size of $\mathcal{C}$ is at most $\mathcal{O}(|\max_{1\leq i\leq n} |\mathcal{A}_i|)^{\mathcal{O}(m^2+n)}$. In particular, when $|\Psi|$ is fixed (i.e., n, m and $|\varphi|$ are fixed), $\mathcal{C}$ can be constructed in polynomial time since the satisfiability of an existential Presburger formula with a fixed number of quantifiers is in PTime [27]. To handle the quantifiers of Ψ, we construct a parity tree automaton $\mathcal{C}'$ that do not take inputs. Essentially, $\mathcal{C}'$ is constructed from $\mathcal{C}$ by releasing the existentially quantified variables through projections, i.e., leaving the tree automaton a non-deterministic choice while relaxing a dimension of the input tree. Universal quantifiers $\forall x : \Psi'$ are treated as $\neg\exists x : \neg\Psi'$, where each negation $\neg$ requires the computation of the complement of the current tree automaton, and then induce an exponential blow up of the computation time [24]. The parity game is constructed in PTime from a tree automaton $\mathcal{C}'$. The game proceeds with the even player first choosing a transition in the tree automaton, and then the odd player choosing a subtree. The even player wins iff the language of the tree automaton is not empty. Naturally, a resolver logic formula with only existential quantifiers do not require tree automata complementations. However, with a naive approach, a formula with only universal quantifiers may requires two complementations while none are necessary. This is because if the innermost quantifier is universal then the first complementation can be avoided by using $\neg\varphi$ instead of φ to construct $\mathcal{C}$. Additionally, if the outermost quantifier is universal then the last complementation can be avoided by constructing a parity game that is winning for the player with even objective if and only if the current tree automaton is empty. This is doable in PTime as before, but the players are swapped and the priority are increased by one. ◀