Permutation Games for the Weakly Aconjunctive μ-Calculus

Daniel Hausmann(✉), Lutz Schröder(✉), and Hans-Peter Deifel

Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany
{daniel.hausmann,lutz.schroeder}@fau.de

Abstract. We introduce a natural notion of limit-deterministic parity automata and present a method that uses such automata to construct satisfiability games for the weakly aconjunctive fragment of the μ-calculus. To this end we devise a method that determinizes limit-deterministic parity automata of size n with k priorities through limit-deterministic Büchi automata to deterministic parity automata of size $\mathcal{O}((nk)!)$ and with $\mathcal{O}(nk)$ priorities. The construction relies on limit-determinism to avoid the full complexity of the Safra/Piterman-construction by using partial permutations of states in place of Safra-Trees. By showing that limit-deterministic parity automata can be used to recognize unsuccessful branches in pre-tableaux for the weakly aconjunctive μ-calculus, we obtain satisfiability games of size $\mathcal{O}((nk)!)$ with $\mathcal{O}(nk)$ priorities for weakly aconjunctive input formulas of size n and alternation-depth k. A prototypical implementation that employs a tableau-based global caching algorithm to solve these games on-the-fly shows promising initial results.

1 Introduction

The modal μ-calculus [15] is an expressive logic for reasoning about concurrent systems. Its satisfiability problem is ExpTime-complete [5]. Due to nesting of fixpoints, the semantic structure of the μ-calculus is quite involved, which is reflected in the high degree of sophistication of reasoning algorithms for the μ-calculus. One convenient modular approach is the definition of suitable *satisfiability games* (e.g. [10]); solving such games (i.e. computing their winning regions) then amounts to deciding the satisfiability of the input formulas. A standard method for obtaining satisfiability games is to first construct a *tracking automaton* that accepts the *bad branches* in a pre-tableau for the input formula, i.e. those that infinitely defer satisfaction of a least fixpoint; this automaton then is determinized and complemented, and the satisfiability game is built over the carrier set of the resulting automaton. The moves in the game are those transitions from the automaton that correspond to applications of tableau-rules; the existence of a winning strategy in this game ensures the existence of a model,

D. Beyer and M. Huisman (Eds.): TACAS 2018, LNCS 10806, pp. 361–378, 2018.
https://doi.org/10.1007/978-3-319-89963-3_21

i.e. a locally coherent structure that does not contain bad branches. As they typically incur exponential blowup, good determinization procedures for automata on infinite words play a crucial role in standard decision procedures for the satisfiability problem of the μ-calculus and its fragments; in particular, better determinization procedures lead to smaller satisfiability games which are easier to solve.

The *weakly aconjunctive* μ-calculus [15,24] restricts occurrences of recursion variables in conjunctions but is still quite expressive, e.g. can define winning regions in parity games with bounded number of priorities [4]. The key observation for the present paper is that in the weakly aconjunctive case, pre-tableau branches are made 'bad' by a single formula; this implies that the tracking automaton for such formulas is *limit-deterministic*, i.e. that it is sufficient to deterministically track a single formula from some point on. This motivates a notion of *limit-deterministic parity automata* in which all accepting runs are deterministic from some point on. Because the nondeterminism is restricted to finite prefixes of accepting runs in such automata, they can be determinized in a simpler way than unrestricted parity automata. We present a reformulation of a recent determinization method for limit-deterministic *Büchi* automata [6]. The method is inspired by, but significantly less involved than the more general Safra/Piterman construction [19,20], essentially due to the fact that the tree structure of Safra trees collapses, leaving only the permutation structure. The resulting parity automaton can thus be described as a *permutation automaton*. The method yields deterministic parity automata with $\mathcal{O}(n!)$ states, compared to $\mathcal{O}((n!)^2)$ in the Safra/Piterman construction. Crucially, we show that we obtain a similarly simplified determinization for limit-deterministic *parity* automata by translating into Büchi automata.

As indicated above, limit-deterministic parity automata are able to recognize bad branches in pre-tableaux for weakly aconjunctive μ-calculus formulas. Employing them in the standard construction of satisfiability games, we obtain *permutation games* in which nodes from the pre-tableau are annotated with a partial permutation (i.e. a non-repetitive list) of (levelled) formulas. A parity condition is used to detect indices in the permutation that are active infinitely often without ever being removed from the permutation. The resulting parity games are of size $\mathcal{O}((nk)!)$ and have $\mathcal{O}(nk)$ priorities; as a side result, we thus obtain a new bound $\mathcal{O}((nk)!)$ on model size for weakly aconjunctive formulas.

The resulting decision procedure generalizes to the weakly aconjunctive *coalgebraic* μ-calculus, thus covering also, e.g., probabilistic and alternating-time versions of the μ-calculus. The generic algorithm has been implemented as an extension of the *Coalgebraic Ontology Logic Reasoner* (COOL) [11,13]. Our implementation constructs and solves the presented permutation games *on-the-fly*, possibly finishing satisfiability proofs early, and shows promising initial results. The content of the paper is structured as follows: We describe the determinization of limit-deterministic automata in Sect. 2 and the construction of permutation games in Sect. 3, and discuss implementation and evaluation in Sect. 4.

Related Work. Liu and Wang [17] give a tighter estimate $\mathcal{O}((n!)^2)$ for the number of states in Piterman's determinization [19]. Schewe [21] simplifies Piterman's construction (establishing the same bound as Liu and Wang). Tian and Duan [23] further improve Schewe's construction. Fisman and Lustig [7] present a modularization of Büchi determinization that is aimed mainly at easing understanding of the construction. Parity automata can be determinized by first converting them to Büchi automata and then applying Büchi determinization. Schewe and Varghese [22] address the direct determinization of parity automata (via Rabin automata), and prove optimality within a small constant factor, and even absolute optimality for the Büchi subcase. All these constructions and estimates concern unrestricted Büchi or parity automata. Recently, Safra-less determinization of limit-deterministic Büchi automata has been described in the context of controller synthesis for LTL [6]; the determinization method that we present in Sect. 2.2. has been devised independently from [6] but employs a very similar construction (yielding essentially the same results on the complexity of the construction).

The use of games in μ-calculus satisfiability checking goes back to Niwiński and Walukiewicz [18] and has since been extended to the unguarded μ-calculus [10] and the *coalgebraic* μ-calculus [2]. Game-based procedures for the relational μ-calculus have been implemented in MLSolver [9], and for the alternation-free coalgebraic μ-calculus in COOL [13].

2 Determinizing Limit-Deterministic Automata

2.1 Limit-Deterministic Automata

We recall the basics of parity automata: A *parity automaton* is a tuple $\mathcal{A} = (V, \Sigma, \delta, u_0, \alpha)$ where V is a set of *states*, Σ is an *alphabet*, $\delta \subseteq V \times \Sigma \times V$ is a *transition relation*, $u_0 \in V$ is an *initial state*, and $\alpha : \delta \to \mathbb{N}$ is a *priority function* that assigns natural numbers to *transitions* (assigning priorities to transitions rather than states yields a slightly more succinct notion of automata while retaining the computational properties of standard parity automata [22]). For $(v, a) \in V \times \Sigma$, we write $\delta(v, a) = \{u \mid (v, a, u) \in \delta\}$. The *index* $\mathsf{idx}(\mathcal{A}) = \max\{\alpha(t) \mid t \in \delta\}$ of a parity automaton $\mathcal{A}$ is its maximal priority. A *run* $\rho = v_0 v_1 \ldots$ of $\mathcal{A}$ on an infinite word $w = a_0 a_1 \ldots \in \Sigma^\omega$ starting at $v \in V$ is a (possibly infinite) sequence of states v_i such that $v_0 = v$ and for all $i \geq 0$, $v_{i+1} \in \delta(v_i, a_i)$. We see runs ρ or words w as functions from natural numbers to states $\rho(i) = v_i \in V$ or letters $w(i) = a_i \in \Sigma$. For a run ρ on a word w, we define the according sequence $\mathsf{trans}(\rho)$ of transitions by $\mathsf{trans}(\rho)(i) = (\rho(i), w(i), \rho(i+1))$. We denote the set of all runs of $\mathcal{A}$ on a word w starting at v by $\mathsf{run}(\mathcal{A}, v, w)$, or just by $\mathsf{run}(\mathcal{A}, w)$ if $v = u_0$. A run ρ of $\mathcal{A}$ on a word w is *accepting* if the highest priority that occurs infinitely often in it (notation: $\max(\mathsf{Inf}(\alpha \circ \mathsf{trans}(\rho)))$; we generally write $\mathsf{Inf}(s)$ for the set of elements occurring infinitely often in a sequence s) is even. A parity automaton $\mathcal{A}$ *accepts* an infinite word w if $\mathsf{run}(\mathcal{A}, w)$ contains an accepting run, and we denote by $L(\mathcal{A}) \subseteq \Sigma^\omega$ the set of all words that are accepted by $\mathcal{A}$.

Given a state $v \in V$ and a letter $a \in \Sigma$, we define $\delta|_{v,a} = \{(v,a,u) \mid u \in \delta(v,a)\}$. Given a set $\gamma \subseteq \delta$ of transitions, a state $v \in V$, a set of states $U \subseteq V$ and a letter $a \in \Sigma$, we put $\gamma(U,a) = \bigcup\{\gamma(v,a) \mid v \in U\}$; given a finite word $w = a_0 \dots a_n$, we then recursively define $\gamma(v,w) = \gamma(\gamma(v,a_0), a_1 \dots a_n)$, obtaining the set of all states reachable from v when reading w while only using transitions from γ. For $U \subseteq V$, $\gamma \subseteq \delta$ and $w \in \Sigma^*$, we put $\gamma(U,w) = \bigcup\{\gamma(u,w) \mid u \in U\}$. Furthermore, we define the set of states that are *reachable* from a node $v \in V$ using transitions from γ as $\mathsf{reach}_\gamma(v) = \bigcup\{\gamma(v,w) \mid w \in \Sigma^*\}$; we extend this notation to sets of nodes, putting $\mathsf{reach}_\gamma(U) = \bigcup\{\mathsf{reach}_\gamma(u) \mid u \in U\}$ for $U \subseteq V$. If $\gamma = \delta$, then we omit the subscripts. A state $v \in V$ is said to be *deterministic* (in $\gamma \subseteq \delta$) if it has at most one (γ-)successor for each letter $a \in \Sigma$. A set $U \subseteq V$ is deterministic (in $\gamma \subseteq \delta$) if every state $v \in U$ is deterministic (in γ). The automaton $\mathcal{A}$ is said to be *deterministic* if V is deterministic; the transition relation in deterministic automata hence is a partial function (since such automata can be transformed to equivalent automata with total transition function, this definition suffices for purposes of determinization). We put $\alpha(i) = \{t \in \delta \mid \alpha(t) = i\}$ and $\alpha_{\leq}(i) = \{t \in \delta \mid \alpha(t) \leq i\}$.

A *Büchi automaton* is a parity automaton with only the priorities 1 and 2; the set of *accepting transitions* then is $F = \alpha(2)$ and a run is accepting if it passes infinitely many accepting transitions. For Büchi automata, we assume w.l.o.g. that every transition $t \in F$ is part of a cycle. We use the abbreviations (N/D)PA, (N/D)BA to denote the different types of automata.

Our notion of limit-determinism of automata is defined as a semantic property:

Definition 1 (Limit-deterministic parity automata). A PA $\mathcal{A} = (V, \Sigma, \delta, u_0, \alpha)$ is *limit-deterministic* if there is, for each word w and each accepting run $\rho \in \mathsf{run}(\mathcal{A}, w)$, a number i such that for all $j \geq i$, $\delta|_{\rho(j),w(j)} \cap \alpha_{\leq}(l) = \{\mathsf{trans}(\rho)(j)\}$, where $l = \max(\mathsf{Inf}(\alpha \circ \mathsf{trans}(\rho)))$.

If $\mathcal{A}$ is a BA, then we have $\max(\mathsf{Inf}(\alpha \circ \mathsf{trans}(\rho))) = 2$ for every accepting run ρ; as $\alpha_{\leq}(2) = \delta$, the above definition instantiates to requiring the existence of a number i such that for all $j \geq i$, $\delta(\rho(j), w(j)) = \{\rho(j+1)\}$.

Definition 2 (Compartments). Given a PA $\mathcal{A} = (V, \Sigma, \delta, u_0, \alpha)$ with k priorities, and an even number $l \leq k$, the *l-compartment* $C_l(t)$ of a transition $t \in \alpha(l)$ is the set $\mathsf{reach}_{\alpha_{\leq}(l)}(\pi_3(t))$ where π_3 projects transitions $t = (v,a,u)$ to their target nodes u. If l is irrelevant, then we refer to l-compartments just as compartments. The *size* of a compartment C is just $|C|$. A compartment C is *internally deterministic* if for each $v \in C$ and all $a \in \Sigma$, $|\delta(v,a) \cap C| \leq 1$.

Note that the union of all l-compartments is $\mathsf{reach}_{\alpha_{\leq}(l)}(\pi_3[\alpha(l)])$. Compartments allow for a syntactic characterization of limit-determinism:

Lemma 3. *A PA is limit-deterministic if and only if all its compartments are internally deterministic.*

Corollary 4. *It is decidable in polynomial time whether a given automaton is limit-deterministic.*

Lemma 3 specializes to BA as follows: we have $\alpha(0) = \emptyset$, $\alpha_{\leq}(2) = \delta$ and $\alpha(2) = F$, so that the union of all 0-compartments is empty and that of all 2-compartments is $\mathsf{reach}(\pi_3[F])$; thus a BA is limit-deterministic if and only if $\mathsf{reach}(\pi_3[F])$ is deterministic. Such Büchi automata are also called *semi-deterministic* [3].

2.2 Determinizing Limit-Deterministic Büchi Automata

The Safra/Piterman construction [19,20] determinizes Büchi automata by means of so-called Safra trees, i.e. trees whose nodes are labelled with sets of states of the input automaton such that the label of a node is a proper superset of the union of all its children's labels. Additionally, the nodes are ordered by their age and upon each transition between Safra trees, the ages of the oldest nodes that are active and/or removed during this transition determine the priority of the new Safra tree. In its original formulation, the Safra/Piterman construction adds new child nodes to the graph that are labelled with the accepting states in their parent's label. We observe that this step can be modified slightly – without affecting the correctness of the construction – by letting every accepting state from the parent's label receive its own separate child node; then the labels of newly created nodes are always singletons. Limit-determinism of the input automaton then implies that the node labels also *remain* singletons. Since singleton nodes do not have children in Safra trees, this leads to the collapse of their tree structure; the resulting data structure is essentially a partial permutation, i.e. a non-repetitive list, of states (ordered by their age). The arising modified Safra/Piterman construction for the limit-deterministic case boils down to the following method, which (a) has a relatively short presentation and a simpler correctness proof than the full Safra/Piterman construction, and (b) results in asymptotically smaller automata; the underlying idea of the construction has first been described in the context of controller synthesis for LTL [6].

Definition 5 (Partial permutations). Given a set U of states, let $\mathsf{pperm}(U)$ denote the set of *partial permutations* over U, i.e. the set of non-repetitive lists $l = [v_1, \ldots, v_n]$ with $v_i \neq v_j$ for $i \neq j$ and $v_i \in U$, for all $1 \leq i \leq n$. We denote the i-th element in l by $l(i) = v_i$, the empty partial permutation by $[\,]$ and the length of a partial permutation l by $|l|$.

Definition 6 (Determinization of limit-deterministic BA). Fix a limit-deterministic BA $\mathcal{A} = (V, \Sigma, \delta, u_0, F)$, and put $Q = \mathsf{reach}(\pi_3[F])$, $\overline{Q} = V \setminus Q$, $q = |Q|$. Define the DPA $\mathcal{B} = (W, \Sigma, \delta', w_0, \alpha)$ by putting $W = \mathcal{P}(\overline{Q}) \times \mathsf{pperm}(Q)$, $w_0 = (\{u_0\}, [\,])$ if $u_0 \in \overline{Q}$, $w_0 = (\emptyset, [u_0])$ if $u_0 \in Q$ and for $g = (U, l) \in W$ and $a \in \Sigma$, $\delta'(g, a) = h$, where $h = (\delta(U, a) \cap \overline{Q}, l')$ and where l' is constructed from $l = [v_1, \ldots, v_m]$ as follows:

1. Define a list t of length m over $Q \cup \{*\}$ (with $*$ representing undefinedness) in which $t(i) = w$ if $\delta(v_i, a) = \{w\}$, and $t(i) = *$ if $\delta(v_i, a) = \emptyset$.

2. For $j < k$ and $t(j) = t(k)$, put $t(k) = *$.
3. Remove undefined entries in t, formally: for each $1 \leq i \leq |t|$, if $t(i) = *$, then iteratively put $t(j) = t(j+1)$ for each $i \leq j \leq |t|$, starting at i.
4. For any $w \in \delta(U, a) \cap Q$ that does not occur in t, add w to the end of t. If there are several such w, the order in which they are added to t is irrelevant.
5. Put $l' = t$.

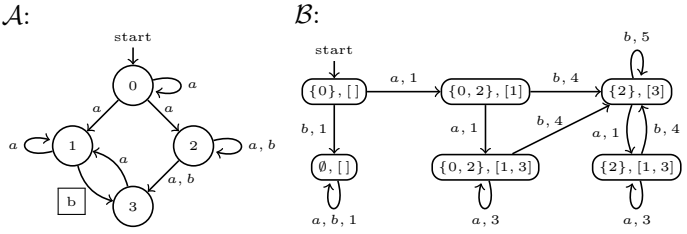
Temporarily, t may contain duplicate or undefined entries, but Steps 2. and 3. ensure that in the end, t is a partial permutation of length at most q. Let r (for 'removed') denote the lowest index i such that $t(i) = *$ after Step 2. Let a (for 'active') denote the lowest index i such that $(l(i), a, l'(i)) \in F$. If $r > |l'|$ and there is no i with $(l(i), a, l'(i)) \in F$, then put $\alpha(g, a, h) = 1$. Otherwise, put

$$\alpha(g, a, h) = \begin{cases} 2(q - r) + 3 & \text{if } r \leq a \\ 2(q - a) + 2 & \text{if } r > a. \end{cases}$$

Theorem 7. *We have $L(\mathcal{A}) = L(\mathcal{B})$, and $\mathcal{B}$ has at most $2n + 1$ priorities; for $n \geq 4$, we have $|W| \leq n!e$.*

Corollary 8. *Limit-deterministic Büchi automata of size n can be determinized to deterministic parity automata of size $\mathcal{O}(n!)$ and with $\mathcal{O}(n)$ priorities.*

Example 9. Consider the limit-deterministic BA $\mathcal{A}$ depicted below and the determinized DPA $\mathcal{B}$ that is constructed from it by applying the method. We see by Lemma 3 that $\mathcal{A}$ is really limit-deterministic: we have $F = \{(1, b, 3)\}$, i.e. the b-transition from state 1 to state 3 (depicted with a boxed transition label) is the only accepting transition; thus we have $Q = \mathsf{reach}(\pi_3[F]) = \{1, 3\}$ (so $\overline{Q} = \{0, 2\}$), and the states 1 and 3 are deterministic. Moreover, $L(\mathcal{A}) = L(\mathcal{B}) = a(a|b)^+(a^+b)^\omega$.



Notice that in $\mathcal{B}$, there is a b-transition with priority 1 from the initial state to the sink state $(\emptyset, [\,])$ and an a-transition to $(\{0, 2\}, [1])$; as $1 \in Q$ but $1 \notin F$, this transition has priority 1. A further b-transition leads from 1 to 3 in $\mathcal{A}$; in $\mathcal{B}$, we have a b-transition from $(\{0, 2\}, [1])$ to $(\{2\}, [3])$ and since $(1, b, 3) \in F$, the first position in the permutation component is active during this transition so that the transition has priority 4. Yet another b-transition loops from $(\{2\}, [3])$ to $(\{2\}, [3])$. Since there is no b-transition starting at state 3, the first element in the permutation is removed in Step 1. of the construction. Since there is a b-transition from 2 to 3, it is added to the permutation again in Step 4. of the

construction. Crucially, however, the priority of the transition is 5, since the first item of the permutation has been (temporarily) removed. The intuition is that the trace of 3 ends when the letter b is read; even though a new trace of 3 immediately starts, we do not consider it to be the same trace as the previous one. Thus the transition obtains priority 5 so that it may be used only finitely often in an accepting run of $\mathcal{B}$, i.e. accepting runs contain an uninterrupted trace that visits state 3 infinitely often. Thus two or more consecutive b's can only occur finitely often in any accepted word.

2.3 Determinizing Limit-Deterministic Parity Automata

To determinize limit-deterministic PA, it suffices to transform them to equivalent limit-deterministic BA and determinize the BA. This transformation from PA to BA is achieved by a construction which is inspired by Theorems 2 and 3 in [14]; we add the observation that the construction preserves limit-determinism.

Definition 10. Given a limit-deterministic PA $\mathcal{C} = (V, \Sigma, \delta, u_0, \alpha)$ with $n = |V|$ and $k > 2$ priorities, we define the limit-deterministic BA $\mathcal{D} = (W, \Sigma, \delta', u_0, F)$ by putting $W = V \cup (V \times \{0, \ldots, \lceil \frac{k-1}{2} \rceil\})$, and for $w \in W$ and $a \in \Sigma$,

$$\delta'(v,a) = \begin{cases} \{(w,m) \mid (v,a,w) \in \alpha(2m)\} \cup \delta(v,a) & \text{if } v \in V \\ \{(w,l) \mid (v',a,w) \in \alpha_{\leq}(2l)\} & \text{if } v = (v',l) \notin V \end{cases}$$

Finally, we put $F = \{((v,l),a,(w,l)) \in \delta' \mid \alpha(v,a,w) = 2l\}$. To see that $\mathcal{D}$ is limit-deterministic, it suffices by Lemma 3 to show that $\mathsf{reach}(\pi_3[F])$ is deterministic. We observe that for each state $(w,l) \in \mathsf{reach}(\pi_3[F])$, (w,l) is deterministic by definition of δ' since w is contained in a (by Lemma 3, internally deterministic) $2l$-compartment of $\mathcal{C}$.

Lemma 11. *We have $L(\mathcal{C}) = L(\mathcal{D})$ and $|W| \leq n(\lceil \frac{k}{2} \rceil + 1) \leq nk$.*

By Theorem 7, $\mathcal{D}$ can be determinized to a DPA $\mathcal{E}$ of size at most $(nk)!e$, with at most $nk + 2$ priorities and with $L(\mathcal{D}) = L(\mathcal{E})$.

Corollary 12. *Limit-deterministic parity automata of size n with k priorities can be determinized to deterministic parity automata of size $\mathcal{O}((nk)!)$ and with $\mathcal{O}(nk)$ priorities.*

3 Permutation Games for the Aconjunctive μ-Calculus

3.1 The μ-Calculus

We briefly recall the definition of the μ-calculus. We fix a set P of *propositions*, a set A of *actions*, and a set $\mathfrak{V}$ of fixpoint variables. The set L_μ of μ-calculus formulas is the set of all formulas ϕ, ψ that can be constructed by the grammar

$$\psi, \phi ::= \bot \mid \top \mid p \mid \neg p \mid X \mid \psi \wedge \phi \mid \psi \vee \phi \mid \langle a \rangle \psi \mid [a]\psi \mid \mu X.\, \psi \mid \nu X.\, \psi$$

where $p \in P$, $a \in A$, and $X \in \mathfrak{V}$; we write $|\psi|$ for the size of a formula ψ. Throughout the paper, we use η to denote one of the fixpoint operators μ or ν. We refer to formulas of the form $\eta X.\,\psi$ as *fixpoint literals*, to formulas of the form $\langle a\rangle\psi$ or $[a]\psi$ as *modal literals*, and to p, $\neg p$ as *propositional literals*. The operators μ and ν *bind* their variables, inducing a standard notion of *free variables* in formulas. We denote the set of free variables of a formula ψ by $\mathsf{FV}(\psi)$. A formula ψ is *closed* if $\mathsf{FV}(\psi) = \emptyset$, and *open* otherwise. We write $\psi \leq \phi$ ($\psi < \phi$) to indicate that ψ is a (proper) subformula of ϕ. We say that ϕ *occurs free* in ψ if ϕ occurs in ψ as a subformula that is not in the scope of any fixpoint operator. Throughout, we *restrict to formulas that are guarded*, i.e. have at least one modal operator between any occurrence of a variable X and an enclosing binder ηX. (This is standard although possibly not without loss of generality [10].) Moreover we assume w.l.o.g. that input formulas are *clean*, i.e. all fixpoint variables are mutually distinct and distinct from all free variables, and *irredundant*, i.e. $X \in \mathsf{FV}(\psi)$ for all subformulas $\eta X.\,\psi$. We refer to a variable X that is bound by a least (greatest) fixpoint operator $\mu X.\chi$ ($\nu X.\chi$) in a formula ϕ as a *μ-variable* (*ν-variable*) of ϕ, and to the process of substituting such an X with its binding fixpoint literal ($\mu X.\chi$ or $\nu X.\chi$, respectively) as *unfolding*. An occurrence of a subformula ψ of a formula ϕ *contains an active μ-variable* [15] if ψ can be converted into a formula containing a free occurrence of a μ-variable of ϕ by repeatedly unfolding ν-variables of ϕ.

Formulas are evaluated over *Kripke structures* $\mathcal{K} = (W, (R_a)_{a\in A}, \pi)$, consisting of a set W of *states*, a family $(R_a)_{a\in A}$ of relations $R_a \subseteq W \times W$, and a valuation $\pi : P \to \mathcal{P}(W)$ of the propositions. Given an *interpretation* $i : \mathfrak{V} \to \mathcal{P}(W)$ of the fixpoint variables, define $[\![\psi]\!]_i \subseteq W$ by the obvious clauses for Boolean operators and propositions, $[\![X]\!]_i = i(X)$, $[\![\langle a\rangle\psi]\!]_i = \{v \in W \mid \exists w \in R_a(v).w \in [\![\psi]\!]_i\}$, $[\![[a]\psi]\!]_i = \{v \in W \mid \forall w \in R_a(v).w \in [\![\psi]\!]_i\}$, $[\![\mu X.\,\psi]\!]_i = \mu[\![\psi]\!]_i^X$ and $[\![\nu X.\,\psi]\!]_i = \nu[\![\psi]\!]_i^X$, where $R_a(v) = \{w \in W \mid (v, w) \in R_a\}$, $[\![\psi]\!]_i^X(G) = [\![\psi]\!]_{i[X\mapsto G]}$, and μ, ν take least and greatest fixpoints of monotone functions, respectively. If ψ is closed, then $[\![\psi]\!]_i$ does not depend on i, so we just write $[\![\psi]\!]$. We denote the *Fischer-Ladner closure* [16] of a formula ϕ by $\mathbf{F}(\phi)$, or just by $\mathbf{F}$, if no confusion arises; intuitively, $\mathbf{F}$ is the set of formulas that can arise as subformulas when unfolding each fixpoint operator in ϕ at most once. We note $\mathbf{F} \leq |\phi|$ [16].

The *aconjunctive fragment* [15] of the μ-calculus is obtained by requiring that for all conjunctions that occur as a subformula, at most one of the conjuncts contains an active μ-variable. In the *weakly aconjunctive fragment* [24], this requirement is loosened to the constraint that all conjunctions that occur as a subformula and contain an active μ-variable are of the shape $\psi \wedge \Diamond\psi_1 \wedge \ldots \wedge \Diamond\psi_n \wedge \Box(\psi_1 \vee \ldots \vee \psi_n)$, where ψ does not contain active μ-variables. For instance, for all n, the formula $\eta X_n \ldots \mu X_1.\nu X_0.\bigvee_{0\leq i\leq n}(q_i \wedge \Diamond X_i)$ is aconjunctive (and equivalent to the weakly aconjunctive formula obtained by replacing $\Diamond X_i$ with $\Diamond X_i \wedge \Diamond\top \wedge \Box(X_i \vee \top)$). The permutation satisfiability games that we introduce work for the more expressive weakly aconjunctive fragment.

We will make use of the standard *tableau rules* [10] (each consisting of one *premise* and a possibly empty set of *conclusions*):

$$(\bot)\ \frac{\Gamma, \bot}{} \qquad (\not\downarrow)\ \frac{\Gamma, p, \neg p}{} \qquad (\wedge)\ \frac{\Gamma, \psi \wedge \phi}{\Gamma, \psi, \phi}$$

$$(\vee)\ \frac{\Gamma, \psi \vee \phi}{\Gamma, \psi \quad \Gamma, \phi} \qquad (\langle a \rangle)\ \frac{\Gamma, [a]\psi_1, \ldots, [a]\psi_n, \langle a \rangle \phi}{\psi_1, \ldots, \psi_n, \phi} \qquad (\eta)\ \frac{\Gamma, \eta X.\, \psi}{\Gamma, \psi[X \mapsto \eta X.\, \psi]}$$

(for $a \in A$, $p \in P$); we refer to the tableau rules by $\mathcal{R}$ and usually write rule applications with premise Γ and conclusion $\Sigma = \Gamma_1, \ldots, \Gamma_n$ sequentially: (Γ / Σ).

To track fixpoint formulas through pre-tableaux, we will use deferrals, that is, the decomposed form of formulas that are obtained by unfolding fixpoint literals.

Definition 13 (Deferrals). Given fixpoint literals $\chi_i = \eta X_i.\, \psi_i$, $i = 1, \ldots, n$, we say that a substitution $\sigma = [X_1 \mapsto \chi_1]; \ldots; [X_n \mapsto \chi_n]$ *sequentially unfolds* χ_n if $\chi_i <_f \chi_{i+1}$ for all $1 \leq i < n$, where we write $\psi <_f \eta X.\, \phi$ if $\psi \leq \phi$ and ψ is open and occurs free in ϕ (i.e. σ unfolds a nested sequence of fixpoints in χ_n innermost-first). We say that a formula χ is *irreducible* if for every substitution $[X_1 \mapsto \chi_1]; \ldots; [X_n \mapsto \chi_n]$ that sequentially unfolds χ_n, we have that $\chi = \chi_1([X_2 \mapsto \chi_2]; \ldots; [X_n \mapsto \chi_n])$ implies $n = 1$ (i.e. $\chi = \chi_1$). A formula ψ *belongs* to an irreducible closed fixpoint literal θ_n, or is a θ_n*-deferral*, if $\psi = \alpha\sigma$ for some substitution $\sigma = [X_1 \mapsto \theta_1]; \ldots; [X_n \mapsto \theta_n]$ that sequentially unfolds θ_n and some $\alpha <_f \theta_1$. We denote the set of θ_n-deferrals by $\mathsf{dfr}(\theta_n)$.

E.g. the substitution $\sigma = [Y \mapsto \mu Y.\, (\Box X \wedge \Diamond\Diamond Y)]; [X \mapsto \theta]$ sequentially unfolds the irreducible closed formula $\theta = \nu X.\, \mu Y.\, (\Box X \wedge \Diamond\Diamond Y)$, and $(\Diamond Y)\sigma = \Diamond \mu Y.\, (\Box\theta \wedge \Diamond\Diamond Y)$ is a θ-deferral. A fixpoint literal is irreducible if it is not an unfolding $\psi[X \mapsto \eta X.\, \psi]$ of a fixpoint literal $\eta X.\, \psi$; in particular, every clean irredundant fixpoint literal is irreducible.

As a technical tool, we define a measure for the depth of alternation at which a deferral resides inside the fixpoint to which it belongs:

Definition 14 (Alternation level and alternation depth). The *alternation level* $\mathsf{al}(\phi\sigma) := \mathsf{al}(\sigma)$ of a deferral $\phi\sigma$ is defined inductively over $|\sigma|$, where $\mathsf{al}(\epsilon) = \mathsf{al}(\epsilon)_\mu = \mathsf{al}(\epsilon)_\nu = 0$, for the empty substitution ϵ, $\mathsf{al}(\sigma; [X \mapsto \eta X.\, \psi]) = \mathsf{al}(\sigma)_\mu + 1$ if $\eta = \mu$ and $\mathsf{al}(\sigma; [X \mapsto \eta X.\, \psi]) = \mathsf{al}(\sigma)_\nu$ otherwise, and

$$\mathsf{al}(\sigma; [X \mapsto \eta X.\, \psi])_\mu = \begin{cases} \mathsf{al}(\sigma)_\mu & \text{if } \eta = \mu \\ \mathsf{al}(\sigma)_\nu + 1 & \text{otherwise} \end{cases}$$

$$\mathsf{al}(\sigma; [X \mapsto \eta X.\, \psi])_\nu = \begin{cases} \mathsf{al}(\sigma)_\nu & \text{if } \eta = \nu \\ \mathsf{al}(\sigma)_\mu + 1 & \text{otherwise} \end{cases}$$

This definition assigns greater numbers to inner fixpoint literals, i.e. to deferrals which occur at higher nesting depth, i.e. with more alternation inside their sequence σ. Given a formula ψ, its *alternation depth* $\mathsf{ad}(\phi)$ is defined as $\mathsf{ad}(\phi) = \max\{\mathsf{al}(\delta) \mid \delta \in \mathbf{F}, \exists \theta.\delta \in \mathsf{dfr}(\theta)\}$.

3.2 Limit-Deterministic Tracking Automata

As a first step towards deciding the satisfiability of a weakly aconjunctive μ-calculus formula ϕ, we now construct a tracking automaton that takes branches of (that is, infinite paths through) standard pre-tableaux for ϕ as input and accepts a branch if and only if it contains a least fixpoint formula whose satisfaction is deferred indefinitely on that branch. To this end, we import the following notions of threads and tableaux from [10]:

Definition 15. A *pre-tableau* for a formula ϕ is a graph the nodes of which are labelled with subsets of the Fischer-Ladner closure $\mathbf{F}$; the graph structure L of a pre-tableau is constructed by applying tableau rules from $\mathcal{R}$ to the labels of nodes with the requirement that for each rule application (Γ / Σ) to the label Γ of a node v, there is a w with $(v, w) \in L$ such that the label of w is contained in Σ. Nodes whose labels are *saturated* (i.e. do not contain propositional or fixpoint operators) are called *states*. Formulas are tracked through rule applications by the *connectedness relation* $\rightsquigarrow \subseteq (\mathcal{P}(\mathbf{F}) \times \mathbf{F})^2$ that is defined by putting $\Phi, \phi \rightsquigarrow \Psi, \psi$ if and only if Ψ is a conclusion of an application of a rule from $\mathcal{R}$ to Φ such that $\phi \in \Phi$, $\psi \in \Psi$, and the rule application transforms ϕ to ψ; if the rule application does not change ϕ, then $\phi = \psi$. E.g. we have $\Phi, \psi_1 \wedge \psi_2 \rightsquigarrow \Psi, \psi_i$, where $i \in \{1, 2\}$ and Ψ is obtained from Φ by applying the rule $(\wedge)$ to $\psi_1 \wedge \psi_2$. A *branch* $\Psi_0, \Psi_1 \ldots$ in a pre-tableau is a sequence of labels such that for all $i > 0$, Ψ_{i+1} is an L-successor of Ψ_i. A *thread* on an infinite branch $\Psi_0, \Psi_1, \ldots$ is an infinite sequence $t = \psi_0, \psi_1 \ldots$ of formulas with $\Psi_0, \psi_0 \rightsquigarrow \Psi_1, \psi_1 \rightsquigarrow \ldots$. A *$\mu$-thread* is a thread t such that $\min(\mathsf{Inf}(\mathsf{al} \circ t))$ is odd, i.e. the outermost fixpoint literal that is unfolded infinitely often in t is a least fixpoint literal. A *bad branch* is an infinite branch that contains a μ-thread. A *tableau* for ϕ is a pre-tableau for ϕ that does not contain bad branches.

We import from [10] the well-known fact that the existence of tableaux in the sense defined above characterizes satisfiability. In [10], the result is shown for the more general *unguarded* μ-calculus; we note that the restriction to guarded formulas does not invalidate the theorem.

Theorem 16 ([10]). *A μ-calculus formula ψ is satisfiable if and only if there is a tableau for ψ.*

Given a formula ϕ, we define the alphabet Σ_ϕ to consist of letters that each identify a rule $R \in \mathcal{R}$, a principal formula from $\mathbf{F}$ and one of the conclusions of R. E.g. the letter $((\vee), 0, p \vee \Diamond q)$ identifies the application of the disjunction rule to a principal formula $p \vee \Diamond q$ and the choice of the left conclusion; thus this letter identifies the transition from $p \vee \Diamond q$ to p by use of rule $(\vee)$. We note $|\Sigma_\phi| \in \mathcal{O}(|\phi|)$. Further, we denote the set of all words that encode some branch and some bad branch in some pre-tableau for ϕ by $\mathsf{Branch}(\phi)$ and $\mathsf{BadBranch}(\phi)$, respectively.

As a crucial result, we now show that limit-deterministic automata are expressive enough to exactly recognize the bad branches in pre-tableaux for weakly aconjunctive formulas.

Lemma 17. *Let ϕ be a weakly aconjunctive formula. Then there is a* limit-deterministic *PA $\mathcal{A} = (V, \Sigma_\phi, \delta, \phi, \alpha)$ with $|V| \leq |\phi|$ and $\mathsf{idx}(\mathcal{A}) \leq \mathsf{ad}(\phi)+1$ such that $L(\mathcal{A}) \cap \mathsf{Branch}(\phi) = \mathsf{BadBranch}(\phi)$.*

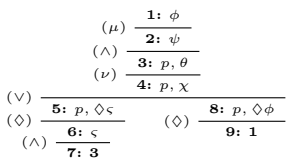
Proof (Sketch). The automaton nondeterministically guesses formulas to be tracked, one at a time; the set of states of the automaton is the Fischer-Ladner closure of ϕ. The priorities of the transitions in the automaton are derived from the alternation level of the target formula of the respective transition; then every word $w \in L(\mathcal{A})$ that encodes some branch encodes a bad branch. Once a deferral is tracked, weak aconjunctivity implies that all compartments to which the tracked formula belongs are internally deterministic; this is the case since for conjunctions $\psi = \psi_0 \wedge \Diamond\psi_1 \wedge \ldots \wedge \Diamond\psi_n \wedge \Box(\psi_1 \vee \ldots \vee \psi_n)$ – the only case that can introduce nondeterminism – each next modal step determines just one of the formulas ψ_i that has to be tracked; the conjunct ψ_0 does not contain active μ-variables, so tracking it causes the automaton to leave all compartments to which ψ belongs. Thus the automaton is limit-deterministic. □

Example 18. We consider the aconjunctive formula

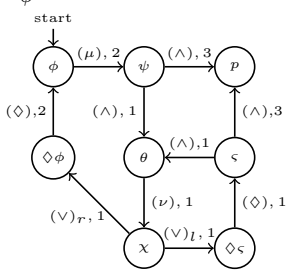
$$\phi = \mu X.(\, p \wedge \nu Y.\, (\Diamond(Y \wedge p) \vee \Diamond X))$$

which expresses the existence of a finite or infinite path on which p holds everywhere. We have the ϕ-deferrals $\phi\epsilon$, $\psi := (p \wedge \nu Y.\, (\Diamond(Y \wedge p) \vee \Diamond X))\sigma_1$, $\theta := (\nu Y.\, (\Diamond(Y \wedge p) \vee \Diamond X))\sigma_1$, $\chi := (\Diamond(Y \wedge p) \vee \Diamond X)\sigma_2$, $(\Diamond(Y \wedge p))\sigma_2$, $\tau := (Y \wedge p)\sigma_2$, $Y\sigma_2$, $\Diamond X\sigma_2$ and $X\sigma_2$, where $\sigma_1 = [X \mapsto \phi]$ and $\sigma_2 = [Y \mapsto \psi]; \sigma_1$. We consider a pre-tableau P_ϕ for ϕ and like in the proof of Lemma 17, we construct the limit-deterministic tracking automaton $\mathcal{A}_\phi$, depicted below:

P_ϕ:



$\mathcal{A}_\phi$:



The priorities in $\mathcal{A}_\phi$ are derived as follows: As $\mathsf{ad}(\phi) = 2$ is even, we put $k = \mathsf{ad}(\phi)+1 = 3$; since $\mathsf{al}(\phi) = \mathsf{al}(\psi) = 1$, $\alpha(\phi, (\mu), \psi) = \alpha(\Diamond\phi, (\Diamond), \phi) = k - \mathsf{al}(\phi) = 2$ and since $\mathsf{al}(p) = 0$, $\alpha(\psi, (\wedge), p) = \alpha(\varsigma, (\wedge), p) = k - \mathsf{al}(\phi) = 3$. All other formulas have alternation level 2 and transitions to them obtain priority 1. The tracking automaton accepts exactly those branches in P_ϕ that start at node **1** and take the loop through node **9** infinitely often; in these branches, ϕ can be tracked forever and evolves to ϕ infinitely often, i.e. their dominating formula is the least fixpoint formula ϕ. All other branches loop through node **7** without passing node **9** from some point on; their dominating fixpoint formula is θ, a greatest fixpoint formula. We observe that due to the aconjunctivity of ϕ, $\mathcal{A}_\phi$ is

limit-deterministic since the only two nondeterministic states ψ and ς each have only one outgoing ($\wedge$)-transition with priority less than $k = 3$.

Given a weakly aconjunctive formula ϕ, we use Lemma 17 to construct a limit-deterministic tracking automaton $\mathcal{A}_\phi$ with $L(\mathcal{A}_\phi) \cap \mathsf{Branch}(\phi) = \mathsf{BadBranch}(\phi)$. Then we put Lemma 11 to use to obtain an equivalent BA in which all states from $Q = \mathsf{reach}(\pi_3[F])$ are *levelled deferrals*, i.e. pairs (ψ, q) consisting of a deferral ψ and a number $q \leq \lceil \frac{k}{2} \rceil$, the *level* of the pair (ψ, q); the level q encodes the odd alternation level $2q - 1$. A levelled deferral (ψ, q) is *active* if $\mathsf{al}(\psi) = 2q - 1$ and the automaton accepts branches which contain a levelled deferral that is active infinitely often without being finished. The set $\overline{Q}$ is just a subset of $\mathbf{F}$. Next we use Theorem 7 to transform this BA to a DPA $\mathcal{B}_\phi$ with $L(\mathcal{A}_\phi) = L(\mathcal{B}_\phi)$. We complement $\mathcal{B}_\phi$ to a DPA $\mathcal{C}_\phi = (W, \Sigma_\phi, \delta, \phi, \alpha)$ by decreasing the priority of each state in $\mathcal{B}_\phi$ by one; we have $L(\mathcal{C}_\phi) = \overline{L(\mathcal{B}_\phi)}$, that is, $\mathcal{C}_\phi$ accepts exactly those words that encode only 'good' branches, if they encode some branch in some pre-tableau for ϕ. By construction, $|W| \in \mathcal{O}((nk)!)$ and $\mathcal{C}_\phi$ has at most $nk + 1$ priorities, and (recalling Definitions 6 and 10) the states in the carrier W of $\mathcal{C}_\phi$ are of the shape (U, l), where U is a subset of $\mathbf{F}$ and l is a partial permutation of levelled deferrals. For a transition $t = ((U, l), r, (V, l'))$ with $(U, l), (V, l') \in W$, $r \in \Sigma_\phi$, if $\alpha(t) = 2(n-a)+1$, then a is the lowest number such that $\mathsf{al}(\phi) = 2q-1$, where $l'(a) = (\phi, q)$ and the a-th element of l is not removed by the transition t (i.e. $\alpha(t)$ references the oldest levelled deferral in l' that is active but not removed by the transition t) and if $\alpha(t) = 2(n-r)+2$, then $\alpha(t)$ is the index of the oldest levelled deferral $(\phi, 2q-1)$ that is finished (i.e. removed from l) in the transition t of the automaton $\mathcal{C}_\phi$, which means that the according r-transition in $\mathcal{A}_\phi$ makes ϕ leave its $2q - 1$-compartment. For a state $v = (U, l)$, we define the *label* $\Gamma(v)$ of v as $\Gamma(v) = U$.

3.3 Permutation Games

The deterministic parity automaton $\mathcal{C}_\phi$ can now be combined with applications of tableau rules from $\mathcal{R}$ to form a satisfiability game for ϕ. We proceed to recall the definition of parity games and some ensuing basic notions. A *parity game* is a graph $\mathcal{G} = (V, E, \alpha)$ that consists of a set of nodes V, a set of edges $E \subseteq V \times V$ and a priority function $\alpha : E \to \mathbb{N}$, assigning priorities to *edges*. We assume $V = V_\exists \uplus V_\forall$, that is, every node in V either belongs to player Eloise ($V_\exists$) or to player Abelard ($V_\forall$). A *play* ρ of $\mathcal{G}$ is a (possibly infinite) sequence $v_0 v_1 \ldots$ such that for all $i \geq 0$, $v_i \in V$ and $(v_i, v_{i+1}) \in E$. A play ρ of $\mathcal{G}$ is won by Eloise if and only if ρ is finite and ends in a node that belongs to Abelard or ρ is infinite and $\max(\mathsf{Inf}(\alpha \circ \mathsf{trans}(\rho)))$ is even (where $\mathsf{trans}(\rho)$ is defined by $\mathsf{trans}(\rho)(i) = (\rho(i), \rho(i+1))$); Abelard wins a play ρ if and only if Eloise does not win ρ. A (memoryless) strategy $s : V \nrightarrow V$ assigns moves to states. A play ρ *conforms* to a strategy s if for all $\rho(i) \in \mathsf{dom}(s)$, $\rho(i+1) = s(\rho(i))$. Eloise has a winning strategy for a node v if there is a strategy $s : V_\exists \to V$ such that every play of $\mathcal{G}$ that starts at v and conforms to s is won by Eloise; we have a dual notion of winning strategies for Abelard. The winning regions $\mathsf{win}_\exists(\mathcal{G})$ and

$\mathsf{win}_\forall(\mathcal{G})$ are the sets of those nodes for which Eloise and Abelard have winning strategies, respectively. *Solving* a parity game $\mathcal{G}$ (locally) for a particular node $v \in V$ amounts to computing the winner of v.

Now we are ready to define permutation games for weakly aconjunctive formulas ϕ, using the DPA $\mathcal{C}_\phi = (W, \Sigma_\phi, \delta, \phi, \alpha)$ from the previous section.

Definition 19 (Permutation games). Let ϕ be a weakly aconjunctive formula. We define the *permutation game* $\mathcal{G}(\phi) = (W, E, \beta)$ to be a parity game that has the carrier of $\mathcal{C}_\phi$ as set of nodes. For every node $v \in W$ for which $\Gamma(v)$ is not a state, we fix a single rule that is to be applied to $\Gamma(v)$ and a single principal formula $\psi_v \in \Gamma(v)$ to which the rule is to be applied. If $(\vee)$ is to be applied to $\Gamma(v)$, then we put $v \in W_\exists$; otherwise, $v \in W_\forall$. In particular, all state nodes are contained in $W_\forall$. For $v \in W$, we put $E(v) = \bigcup\{\delta(v, a) \mid a \in \Sigma_v\}$, where $\Sigma_v \subseteq \Sigma_\phi$ consists of all letters a that encode the application of some rule to $\Gamma(v)$ with the condition that the principal formula of the rule application must be ψ_v if v is not a state node. Finally, we put $\beta(v, w) = \alpha(v, a, w)$ for $(v, w) \in E$, where $a \in \Sigma_v$ encodes the rule application that leads from v to w.

Theorem 20. *Let ϕ be a closed, irreducible and weakly aconjunctive formula. Then we have $(\{\phi\}, [\,]) \in \mathsf{win}_\exists(\mathcal{G}(\phi))$ if and only if ϕ is satisfiable.*

Proof. By construction, Eloise wins $(\{\phi\}, [\,])$ if and only if there is a tableau for ϕ (labelled by the labelling function Γ); we are done by Theorem 16. □

Due to the relatively simple structure and the asymptotically smaller size of the determinized automata $\mathcal{C}_\phi$, the resulting permutation games are somewhat easier to construct and can be solved asymptotically faster than the structures created by standard satisfiability decision procedures for the full μ-calculus (e.g. [5,10]) which employ the full Safra/Piterman-construction; note however, that our method is restricted to the weakly aconjunctive fragment.

Corollary 21. *The satisfiability of weakly aconjunctive μ-calculus formulas can be decided by solving parity games of size $\mathcal{O}((nk)!)$ and $\mathcal{O}(nk)$ priorities.*

The winning strategies for Eloise or Abelard in these games define models for or refutations of the respective formulas, so that we have

Corollary 22. *Satisfiable weakly aconjunctive μ-calculus formulas have models of size $\mathcal{O}((nk)!)$.*

4 Implementation and Benchmarking

We have implemented the permutation satisfiability games as an extension of the *Coalgebraic Ontology Logic Reasoner* (COOL) [11], a generic reasoner for coalgebraic modal logics[1]. COOL achieves its genericity by instantiating an abstract reasoner that works for all coalgebraic logics to concrete instances of logics.

[1] Available at https://www8.cs.fau.de/research:software:cool.

To incorporate support for the aconjunctive coalgebraic μ-calculus, we have extended the global caching algorithm that forms the core of COOL to generate and solve the corresponding permutation games, with optional *on-the-fly* solving; games are solved using either our own implementation of the fixpoint iteration algorithm for parity games (as in [1]) or PGSolver [8], which supports a range of game solving algorithms. Instance logics implemented in COOL currently include linear-time, relational, monotone, and alternating-time logics, as well as any logics that arise as combinations thereof. In particular, this makes COOL, to our knowledge, the only implemented reasoner for the aconjunctive fragments of the alternating-time μ-calculus and Parikh's game logic.

Although our tool supports the aconjunctive coalgebraic μ-calculus, we concentrate on the standard relational aconjunctive μ-calculus for experiments, as this allows us to compare our implementation with the reasoner MLSolver [9], which constructs satisfiability games using the Safra/Piterman-construction and hence supports the full relational μ-calculus; MLSolver uses PGSolver for game solving.

To test the implementations, we devise two series of hard aconjunctive formulas with deep alternating nesting of fixpoints. The following formulas encode that each reachable state in a Kripke structure has one of n priorities (encoded by atoms q_i for $1 \leq i \leq n$) and belongs to either Eloise (q_e) or $\mathsf{Abelard}$ (q_a):

$$\phi_{\mathsf{aut}}(n) = \mathsf{AG}(\bigvee_{1\leq i\leq n}(q_i \wedge \bigwedge_{j\neq i}\neg q_j)) \quad \phi_{\mathsf{game}}(n) = \phi_{\mathsf{aut}}(n) \wedge \mathsf{AG}((q_e \wedge \neg q_a) \vee (\neg q_e \wedge q_a))$$

Here we use $\mathsf{AG}\ \psi$ to abbreviate $\nu X.(\psi \wedge \Box X)$. Then the non-emptiness regions in parity automata and Eloise's winning region in parity games can be specified by the following *aconjunctive* formulas (where $\heartsuit \in \{\Diamond, \Box\}$):

$$\phi_{\mathsf{ne}}(n) = \eta X_n. \ldots . \nu X_2.\mu X_1.\psi_\Diamond \qquad \psi_\heartsuit = \bigvee\nolimits_{1\leq i\leq n}(q_i \wedge \heartsuit X_i)$$
$$\phi_{\mathsf{win}}(n) = \eta X_n. \ldots . \nu X_2.\mu X_1.\phi_{\mathsf{strat}}(\psi_\heartsuit) \quad \phi_{\mathsf{strat}}(\psi_\heartsuit) = (q_e \wedge \psi_\Diamond) \vee (q_a \wedge \psi_\Box)$$

Furthermore, we define (for $\heartsuit \in \{\Diamond, \Box\}$)

$$\theta_\heartsuit(i) = (q_i \wedge \heartsuit Y) \vee \bigvee\nolimits_{i<j\leq n}(q_j \wedge \heartsuit X) \vee \bigvee\nolimits_{1\leq j\leq i}(q_j \wedge \heartsuit Z)$$

The following series of valid formulas states that parity automata with n priorities can be transformed to nondeterministic parity automata with three priorities without affecting the non-emptiness region:

$$\theta_1(n) := \phi_{\mathsf{aut}}(n) \rightarrow (\phi_{\mathsf{ne}}(n) \leftrightarrow \bigvee\nolimits_{i \text{ even}} \mu X.\nu Y.\mu Z.\ \theta_\Diamond(i))$$

Similarly, if Eloise wins a parity game with n priorities, then she can ensure that in each play, each odd priority $1 \leq i \leq n$ is visited only finitely often, unless a priority greater than i is visited infinitely often (the converse does not hold in general [4]):

$$\theta_2(n) := \phi_{\mathsf{game}}(n) \rightarrow (\phi_{\mathsf{win}}(n) \rightarrow \bigwedge_{i \text{ odd}} \nu X.\mu Y.\nu Z.\ \phi_{\mathsf{strat}}(\theta_\heartsuit(i))\ )$$

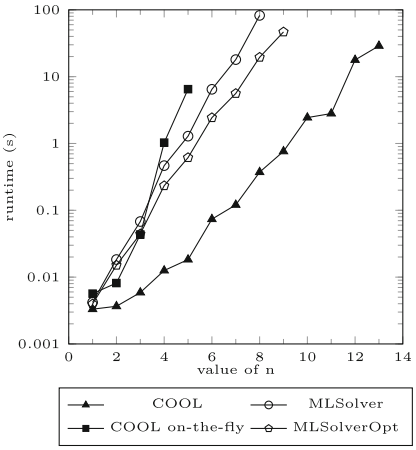


Fig. 1. Times for $\neg\theta_1(n)$ (unsatisfiable)

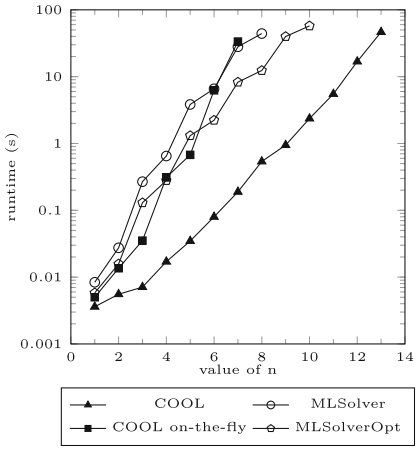


Fig. 2. Times for $\neg\theta_2(n)$ (unsatisfiable)

Additionally, we devise two series of unsatisfiable formulas that exhibit the advantages of COOL's global caching and on-the-fly-solving capabilities. These formulas are inspired by the CTL-formula series $\mathsf{early}(n,j,k)$ and $\mathsf{early}_{\mathsf{gc}}(n,j,k)$ from [13] but contain fixpoint-alternation of depth 2^k inside the subformula θ:

$$\begin{aligned}
\mathsf{early\text{-}ac}(n,j,k) &= \mathit{start}_p \wedge \mathsf{init}(p,n) \wedge \mathsf{init}(r,k) \wedge \mathsf{AG}\,((r \to \mathsf{c}(r,k)) \wedge (p \to \mathsf{c}(p,n))) \wedge \\
&\quad \mathsf{AG}\,((\textstyle\bigwedge_{0\le i\le j} p_i \to \Diamond(\mathit{start}_r \wedge \theta)) \wedge \neg(p \wedge r) \wedge (r \to \Box\, r)) \\
\mathsf{early\text{-}ac}_{\mathsf{gc}}(n,j,k) &= \mathsf{early\text{-}ac}(n,j,k) \wedge b \wedge \mathsf{init}(q,n) \wedge \mathsf{AG}\,(\neg(p \wedge q) \wedge \neg(q \wedge r)) \wedge \\
&\quad \mathsf{AG}\,((q \to c(q,n)) \wedge \mathsf{AF}\; b \wedge (b \to (\Diamond\, p \wedge \Diamond\, \mathit{start}_q \wedge \Box\, \neg b))) \\
\mathsf{init}(x,m) &= \mathsf{AG}\,((\mathit{start}_x \to (x \wedge \textstyle\bigwedge_{0\le i<m} \neg x_i)) \wedge (x \to \Diamond\, x)) \\
\theta &= \eta X_{(2^k)}. \ldots . \nu X_2 . \mu X_1 . \textstyle\bigvee_{1\le i\le 2^k} (\mathsf{bin}(r, i-1) \wedge \Diamond X_i),
\end{aligned}$$

where $\mathsf{c}(x,m)$ encodes an m-bit counter using atoms $x_0,\ldots,x_{m-1}$ and $\mathsf{bin}(r,i)$ denotes the binary encoding of the number i using atoms $r_0,\ldots,r_{k-1}$. The formulas $\mathsf{early\text{-}ac}(n,j,k)$ specify a loop p of length 2^n that branches after j steps to a second loop r of length 2^k on which the highest value of the counter (which counts from 0 to 2^k-1 and then restarts at 0) is required to be an even number. For constant k, the contradiction on loop r yields a small refutation which can be found early, using on-the-fly solving. The formulas $\mathsf{early\text{-}ac}_{\mathsf{gc}}(n,j,k)$ extend this specification by stating that a third loop q of length 2^n is started from loop p infinitely often. Procedures with sufficient caching capabilities will have to (partially) explore this loop at most once.

We compare the runtimes of MLSolver and COOL on the formulas described above; we let COOL and MLSolver solve games using the local strategy improvement algorithm `stratimprloc2` provided by PGSolver. To solve games *on-the-fly* with COOL however, we use our own implementation of the fixpoint iteration

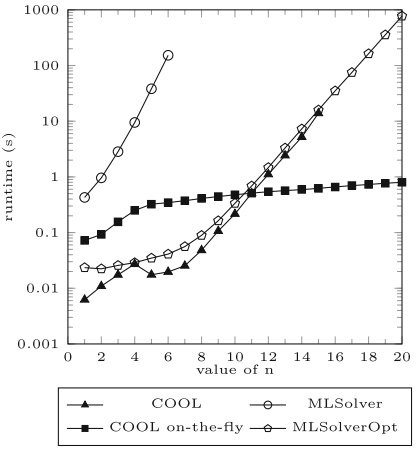


Fig. 3. early-ac$(n, 4, 2)$ (unsatisfiable)

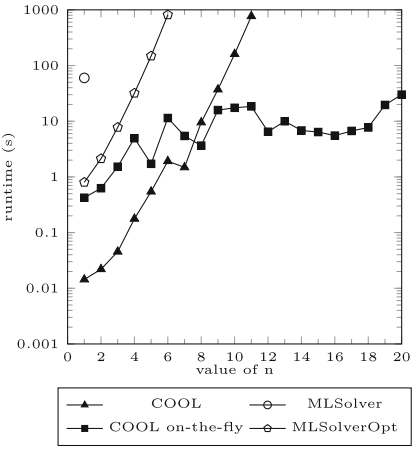


Fig. 4. early-ac$_{\mathsf{gc}}(n, 4, 2)$ (unsatisfiable)

algorithm, which in general is slower than PGSolver but has the advantage that it enables on-the-fly solving. With this option enabled, COOL constructs and solves the satisfiability games step by step and finishes as soon as one of the players has a winning strategy in the partial game. For COOL, we have conducted all experiments with and without on-the-fly solving. For MLSolver, we also enabled the optimizations `-opt litpro` and `-opt comp` (and refer to the resulting prover configuration as MLSolverOpt). Tests have been run on a system with Intel Core i7 3.60 GHz CPU with 16 GB RAM. A more detailed description of the results of the experiments as well as binaries of a formula generator, the prover COOL and scripts that benchmark the various configurations of the provers are available in a figshare repository at [12].

We observe that COOL without on-the-fly solving generally finishes faster than both MLSolver and MLSolverOpt throughout all tested series of formulas (see Figs. 1–4); the reason for this appears to be that the permutation games solved by COOL are of size $\mathcal{O}((nk)!)$, where $n \leq k$, and hence asymptotically smaller than the Safra/Piterman games solved by MLSolver which are of size $\mathcal{O}(((nk)!)^2)$. The size of the refutations for the formulas $\theta_1(n)$ and $\theta_2(n)$ is exponential in n so that on-the-fly solving does in fact *increase* the runtimes of COOL (see Figs. 1 and 2); basically, these formulas cannot be decided early, and therefore any (necessarily unsuccessful) attempt to do so just consumes additional computation time. The formulas early-ac$(n, 4, 2)$ and early-ac$_{\mathsf{gc}}(n, 4, 2)$, on the other hand, have refutations of size polynomial in n, and COOL appears to benefit from on-the-fly solving for these formulas as it is able to decide them early (see Figs. 3 and 4). As mentioned above, COOL uses our own unoptimized implementation of the fixpoint iteration algorithm [1] for on-the-fly solving; while this implementation is slower than PGSolver's `stratimprloc2` algorithm, the on-the-fly abilities of COOL seem to compensate this disadvantage for the early-ac$(n, 4, 2)$ and early-ac$_{\mathsf{gc}}(n, 4, 2)$ formulas from $n = 11$ and $n = 8$ on, respectively.

5 Conclusion

We have presented a method to obtain satisfiability games for the *weakly aconjunctive* μ-calculus. The game construction uses determinization of *limit-deterministic* parity automata, avoiding the full complexity of the Safra/Piterman construction a) in the presentation of the procedure and its correctness proof and b) in the size of the obtained DPA (which comes from $\mathcal{O}((nk)!^2)$ to $\mathcal{O}((nk)!)$). The resulting permutation satisfiability games for the weakly aconjunctive μ-calculus are of size $\mathcal{O}((nk)!)$, have $\mathcal{O}(nk)$ priorities, and yield a new bound of $\mathcal{O}((nk)!)$ on the model size for this fragment. We have implemented this decision procedure in coalgebraic generality and with support for on-the-fly solving as part of the coalgebraic satisfiability solver COOL; initial experiments show favourable results.

The datasets generated and analyzed during the current study are available in the figshare repository: https://doi.org/10.6084/m9.figshare.5919451.v1.

References

1. Bruse, F., Falk, M., Lange, M.: The fixpoint-iteration algorithm for parity games. In: Games, Automata, Logics and Formal Verification, GandALF 2014, EPTCS, vol. 161, pp. 116–130 (2014)
2. Cîrstea, C., Kupke, C., Pattinson, D.: EXPTIME tableaux for the coalgebraic μ-calculus. In: Grädel, E., Kahle, R. (eds.) CSL 2009. LNCS, vol. 5771, pp. 179–193. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04027-6_15
3. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J. ACM **42**(4), 857–907 (1995)
4. Dawar, A., Grädel, E.: The descriptive complexity of parity games. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 354–368. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87531-4_26
5. Emerson, E.A., Jutla, C.: The complexity of tree automata and logics of programs. SIAM J. Comput. **29**(1), 132–158 (1999)
6. Esparza, J., Křetínský, J., Raskin, J.-F., Sickert, S.: From LTL and limit-deterministic Büchi automata to deterministic parity automata. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 426–442. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_25
7. Fisman, D., Lustig, Y.: A modular approach for Büchi determinization. In: Concurrency Theory, CONCUR 2015, LIPIcs, vol. 42, pp. 368–382. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2015)
8. Friedmann, O., Lange, M.: The PGSolver collection of parity game solvers. Technical report, LMU Munich (2009)
9. Friedmann, O., Lange, M.: A solver for modal fixpoint logics. In: Methods for Modalities, M4M-6 2009, ENTCS, vol. 262, pp. 99–111 (2010)
10. Friedmann, O., Lange, M.: Deciding the unguarded modal μ-calculus. J. Appl. Non-Classical Log. **23**, 353–371 (2013)
11. Gorín, D., Pattinson, D., Schröder, L., Widmann, F., Wißmann, T.: Cool – a generic reasoner for coalgebraic hybrid logics (system description). In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS (LNAI), vol. 8562, pp. 396–402. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08587-6_31

12. Hausmann, D., Schröder, L., Deifel, H.-P.: Permutation games for the weakly aconjunctive μ-calculus (artifact). Figshare (2018). https://doi.org/10.6084/m9.figshare.5919451.v1
13. Hausmann, D., Schröder, L., Egger, C.: Global caching for the alternation-free coalgebraic μ-calculus. In: Concurrency Theory, CONCUR 2016, LIPIcs, vol. 59, pp. 34:1–34:15 (2016). Schloss Dagstuhl - Leibniz-Zentrum für Informatik
14. King, V., Kupferman, O., Vardi, M.Y.: On the complexity of parity word automata. In: Honsell, F., Miculan, M. (eds.) FoSSaCS 2001. LNCS, vol. 2030, pp. 276–286. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45315-6_18
15. Kozen, D.: Results on the propositional μ-calculus. Theor. Comput. Sci. **27**, 333–354 (1983)
16. Kozen, D.: A finite model theorem for the propositional μ-calculus. Stud. Log. **47**, 233–241 (1988)
17. Liu, W., Wang, J.: A tighter analysis of Piterman's Büchi determinization. Inf. Process. Lett. **109**, 941–945 (2009)
18. Niwinski, D., Walukiewicz, I.: Games for the μ-calculus. Theor. Comput. Sci. **163**, 99–116 (1996)
19. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. Log. Methods Comput. Sci. **3** (2007)
20. Safra, S.: On the complexity of omega-automata. In: Foundations of Computer Science, FOCS 1988, pp. 319–327. IEEE Computer Society (1988)
21. Schewe, S.: Tighter bounds for the determinisation of Büchi automata. In: de Alfaro, L. (ed.) FoSSaCS 2009. LNCS, vol. 5504, pp. 167–181. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00596-1_13
22. Schewe, S., Varghese, T.: Determinising parity automata. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014. LNCS, vol. 8634, pp. 486–498. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44522-8_41
23. Tian, C., Duan, Z.: Büchi determinization made tighter. CoRR, abs/1404.1436 (2014)
24. Walukiewicz, I.: Completeness of Kozen's axiomatisation of the propositional μ-calculus. Inf. Comput. **157**, 142–182 (2000)