Summaries for Context-Free Games*

Lukáš Holík[1], Roland Meyer[2], and Sebastian Muskalla[3]

1 **Brno University of Technology, Brno, Czech Republic**
`holik@fit.vutbr.cz`
2 **TU Braunschweig, Germany; and Aalto University, Finland**
`meyer@cs.uni-kl.de`
3 **TU Braunschweig, Germany**
`muskalla@cs.uni-kl.de`

Abstract

We study two-player games played on the infinite graph of sentential forms induced by a context-free grammar (that comes with an ownership partitioning of the non-terminals). The winning condition is inclusion of the derived terminal word in the language of a finite automaton. Our contribution is a new algorithm to decide the winning player and to compute her strategy. It is based on a novel representation of all plays starting in a non-terminal. The representation uses the domain of Boolean formulas over the transition monoid of the target automaton. The elements of the monoid are essentially procedure summaries, and our approach can be seen as the first summary-based algorithm for the synthesis of recursive programs. We show that our algorithm has optimal (doubly exponential) time complexity, that it is compatible with recent antichain optimizations, and that it admits a lazy evaluation strategy. Our preliminary experiments indeed show encouraging results, indicating a speed up of three orders of magnitude over a competitor.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases summaries, context-free games, Kleene iteration, transition monoid, strategy synthesis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2016.41

1 Introduction

The motivation of our work is to generalize the language-theoretic approach to verification of recursive programs [24, 29] to synthesis. Central to verification are queries $\mathcal{L}(G) \subseteq \mathcal{L}(A)$, where G is a context-free grammar representing the control-flow of a recursive program and A is a finite automaton representing the specification. When moving to synthesis, we replace the inclusion query by a strategy synthesis for an *inclusion game*. This means G comes with an ownership partitioning of the non-terminals. It induces a game arena defined by the sentential forms and the left-derivation relation (replace the leftmost non-terminal, corresponds to executing the recursive program). The winning condition is inclusion in a regular language given by a finite automaton A. To be precise, player *prover* tries to meet the inclusion by deriving terminal words from the language or enforcing infinite derivations. The goal of *refuter* is to disprove the inclusion by deriving a word outside $\mathcal{L}(A)$.

For the verification of recursive programs, the two major paradigms are *summarization* [37, 33] and *saturation* [9, 18]. Procedure summaries compute the effect of a procedure in

* This work was partially supported by the Czech Science Foundation project 16-24707Y, the IT4IXS: IT4Innovations Excellence in Science project LQ1602, and the BUT project FIT-S-14-2486.

the form of an input-output relation. Saturation techniques compute the pre*-image over the configurations of a pushdown system (including the stack). Both were extensively studied, optimized, and implemented [35, 41, 6, 7]. What speaks for summaries is that they seem to be used more often, as witnessed by the vast majority of verification tools participating in the software verification competition [6, 7]. The reason, besides simpler implementability, may be that the stack maintained by the pre*-construction increases the search space.

Saturation has been lifted to games and synthesis in [13, 22], from which closest to our setting is the work of Cachat [13], where the game arena is defined by a pushdown system and the winning condition is given by a regular set of goal configurations, and the work of Muscholl, Schwentick, and Segoufin [30], where a problem similar to ours is solved by a reduction to [13]. In this paper[1], we fill in the empty spot in the picture and propose a solver and synthesis method for context-free inclusion games based on summaries.

Problem \ Method	Saturation	Summarization
Verification	[9, 18]	[37, 33]
Synthesis	[13, 30, 22]	

Overview of Our Method. Our main contribution is a novel representation of inclusion games that combines well with efficient methods from algorithmic verification (see below). The basic data structure are the elements of the *transition monoid* of the automaton A, called boxes. Boxes are relations over the states of A that capture the state changes on A induced by terminal words [12]. As such, they correspond to procedure summaries. The set of all plays starting in a non-terminal yields a (typically infinite) tree. We show how to represent this tree by a (finite) *negation-free Boolean formula over the transition monoid*, where conjunction and disjunction represent the behavior of the players on the inner nodes.

To compute the representation, we employ a fixed-point iteration on a system of equations that reflects closely the rules of the grammar (and hence the shape of the tree). Indeed, we simultaneously compute the formulas for all non-terminals. In the fixed-point computation, a strategy of prover to enforce an infinite play naturally yields a formula equivalent to *false*. For the domain to be finite, we work modulo logical equivalence. The order is implication. Key to the fixed-point computations is the following compositionality: The formula describing the plays from a sentential form $\alpha\beta$ can be obtained by appropriately composing the formulas for α and β. Indeed, since we consider left-derivations, each play starting in $\alpha\beta$ will have a prefix that coincides with a maximal play starting in α, followed by a suffix that essentially is a play from β. Composition is monotonic wrt. implication.

Having a finite representation for the set of plays starting in each non-terminal has several applications. With compositionality, we can construct the formulas for all sentential forms. This allows us to decide whether a sentential form is in the winning region of a player: We compute the formula and check whether it is rejecting in the sense that refuter can enforce the derivation of a word rejected by the automaton. The latter amounts to evaluating the formula under the assignment that sets to *true* the rejecting boxes. When a sentential form is found to belong to the winning region of a player, we show how to compute a winning strategy, explained here for refuter. We transform the formula to conjunctive normal form (CNF). On CNFs, we define so-called *choice functions* that select a box from each clause. We define a strategy such that all conforming plays end in a terminal word represented by a

[1] The full version is available as technical report [25].

chosen box. Instantiating the strategy for a choice function that only picks rejecting boxes (always possible if the initial formula is rejecting) yields a winning strategy for refuter.

Complexity and Efficiency. We show that our algorithm is in 2EXPTIME, which is tight by [30]. Cachat's algorithm is singly exponential and our input instances can be reduced to his with an exponential blow-up, which together also gives a doubly exponential procedure. The complexity of the reduction comes from that it must determinize the automaton A [30].

Our domain is compatible with algorithmic techniques that have proven efficient in a number of applications (see Section 8). We show how to adapt two heuristics to our fixed-point computation over formulas over boxes, namely antichains from [19, 3, 4] and lazy evaluation inspired by [17]. We also discuss the compatibility of our technique with recent algorithms for the analysis of well-structured systems. It is not immediate how to use the same heuristics for Cachat's domain of automata. Moreover, the determinization within the reduction to Cachat's method does not offer much opportunities for optimization, which means there is one level of exponential complexity that is hardly amenable to heuristics.

In preliminary experiments, we have compared an implementation of Cachat's saturation-based algorithm with our new summary-based algorithm. The benchmarks were generated according to the Tabakov-Vardi random automata model [40] that we adapted to grammars. The running times of our algorithm were consistently better by three orders of magnitude (without the aforementioned optimizations). This supports our conjecture that keeping the stack has a negative impact on search procedures, and summaries should be preferable.

2 Inclusion Games on Context-Free Grammars

A context-free grammar (CFG) is a tuple $G = (N, T, P)$, where N is a finite set of non-terminals, T is a finite set of terminals with $N \cap T = \emptyset$, and $P \subseteq N \times \vartheta$ is a finite set of production rules. Here, $\vartheta = (N \cup T)^*$ denotes the set of sentential forms. We write $X \to \eta$ if $(X, \eta) \in P$. We assume that every non-terminal is the left-hand side of some rule. The *left-derivation relation* $\Rightarrow_L$ replaces the leftmost non-terminal X in α by the right-hand side of a rule. Formally, $\alpha \Rightarrow_L \beta$ if $\alpha = wX\gamma$ with $w \in T^*$, $\beta = w\eta\gamma$, and there is a rule $X \to \eta \in P$. We use w to refer to terminal words (so that a following non-terminal is understood to be leftmost). We consider CFGs that come with an *ownership partitioning* $N = N_{\bigcirc} \uplus N_{\square}$ of the set of non-terminals. We say that the non-terminals in $N_{\star}$ are owned by player $\star \in \{\bigcirc, \square\}$. The ownership partitioning is lifted to the sentential forms ($\vartheta = \vartheta_{\bigcirc} \uplus \vartheta_{\square}$) as follows: $\alpha \in \vartheta_{\square}$ if the leftmost non-terminal in α is owned by $\square$, and $\vartheta_{\bigcirc} = \vartheta \setminus \vartheta_{\square}$. In particular, $\bigcirc$ owns all terminal words. Combined with the left-derivation relation, this yields a game arena.

▶ **Definition 1.** Let $G = (N_{\bigcirc} \uplus N_{\square}, T, P)$ be a CFG with ownership partitioning. The *arena induced by* G is the directed graph $(\vartheta_{\bigcirc} \uplus \vartheta_{\square}, \Rightarrow_L)$.

A *play* $p = p_0 p_1 \ldots$ is a finite or infinite path in the arena. Being a path means $p_i \Rightarrow_L p_{i+1}$ for all positions. If it is finite, the path ends in a vertex denoted $p_{last} \in \vartheta$. A path corresponds to a sequence of left-derivations, where for each leftmost non-terminal the owning player selects the rule that should be applied. A play is *maximal* if it has infinite length or if the last position is a terminal word.

The winning condition of the game is defined by inclusion or non-inclusion in a regular language (depending on who is the player) for the terminal words derived in maximal plays. If the maximal play is infinite, it does not derive a terminal word and satisfies inclusion. The regular language is given by a (non-deterministic) finite automaton $A = (T, Q, q_0, Q_F, \to)$.

Here, T is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, $Q_F \subseteq Q$ is the set of final states, and $\rightarrow \subseteq Q \times T \times Q$ is the transition relation. Instead of $(q, a, q') \in \rightarrow$, we write $q \xrightarrow{a} q'$ and extend the relation to words: $q \xrightarrow{w} q'$ means there is a sequence of states starting in q and ending in q' labeled by w. The language $\mathcal{L}(A)$ consists of all words $w \in T^*$ with $q_0 \xrightarrow{w} q_f$ for some $q_f \in Q_F$. We write $\overline{\mathcal{L}(A)} = T^* \setminus \mathcal{L}(A)$ for the complement language.

From now on, we use $A = (T, Q, q_0, Q_F, \rightarrow)$ for finite automata and $G = (N_{\bigcirc} \uplus N_{\square}, T, P)$ for grammars with ownership. Note that both use the terminal symbols T.

▶ **Definition 2.** The *inclusion game* and the *non-inclusion game* wrt. A on the arena induced by G are defined by the following winning conditions. A maximal play p satisfies the *inclusion winning condition* if it is either infinite or we have $p_{last} \in \mathcal{L}(A)$. A maximal play satisfies the *non-inclusion winning condition* if it is finite and $p_{last} \in \overline{\mathcal{L}(A)}$.

The two games are complementary: For every maximal play, exactly one of the winning conditions is satisfied. We will fix player $\bigcirc$ as the *refuter*, the player wanting plays to satisfy non-inclusion, which is a reachability condition. The opponent $\square$ is the *prover*, wanting plays to satisfy inclusion, which is a safety condition. Since refuter has a single goal to achieve and has to enforce termination, we will always explain our constructions from refuter's point of view. To win, prover just has to ensure that she stays in her winning region. She does not need to care about termination.

A *strategy* for player $\bigstar \in \{\bigcirc, \square\}$ is a function that takes a non-maximal play p with $p_{last} \in \vartheta_{\bigstar}$ (it is $\bigstar$'s turn) and returns a successor of this last position. A play *conforms* to a strategy if whenever it is the turn of $\bigstar$, her next move coincides with the position returned by the strategy. A strategy is *winning from a position* p_0 if every play starting in p_0 that is conform to the strategy eventually satisfies the winning condition of the game. The *winning region* for a player is the set of all positions from which the player has a winning strategy.

▶ **Example 3.** Consider the grammar $G_{ex} = (\{X, Y\}, \{a, b\}, \{X \rightarrow aY, X \rightarrow \varepsilon, Y \rightarrow bX\})$. The automaton A_{ex} is given in Figure 1 and accepts $(ab)^*$. If refuter owns X and prover owns Y, then prover has a winning strategy for the inclusion game from position X. Indeed, finite plays only derive words in $(ab)^*$. Moreover, if refuter enforces an infinite derivation, prover wins inclusion as no terminal word is being derived. Refuter can win non-inclusion starting from Y. After prover has chosen $Y \rightarrow bX$, refuter selects $X \rightarrow \varepsilon$ to derive $b \notin (ab)^*$.

Our contribution is an algorithm to compute (a representation of) both, the winning region of the non-inclusion game for $\bigcirc$ and the winning region of the inclusion game for $\square$.

3 From Inclusion Games to Fixed Points

We give a summary-based representation of the set of all plays from each non-terminal and a fixed-point analysis to compute it. We lift the information to the sentential forms.

Domain. The idea of the analysis domain is to use Boolean formulas over words. To obtain a finite set of propositions, we consider words equivalent that induce the same state changes on A, denoted by $\sim_A$. The winning condition is insensitive to the choice of $\sim_A$-equivalent words. This means it is sufficient to take formulas over $\sim_A$-equivalence classes.

To finitely represent the $\sim_A$-equivalence classes, we rely on the *transition monoid of* A, defined as $\mathrm{M}_A = (\mathcal{P}(Q \times Q), \ ; \ , \mathrm{id})$. We refer to the elements $\rho, \tau \in \mathrm{M}_A$ as *boxes*. Since boxes are relations over the states of A, their relational composition is defined as usual,

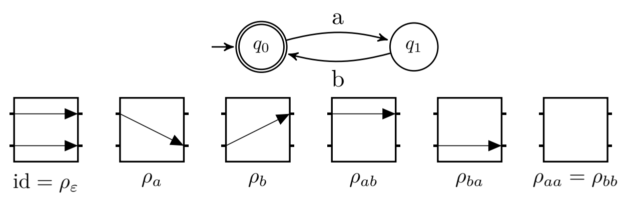

Figure 1 The automaton A_{ex} accepting $(ab)^*$ and all its boxes with non-empty language. The first dash on each side of a box represents state q_0, the second dash represents q_1.

$\rho;\tau = \{(q, q'') \mid \exists q' \in Q : (q, q') \in \rho \text{ and } (q', q'') \in \tau\}$. Relational composition is associative. The identity box id $= \{(q, q) \mid q \in Q\}$ is the neutral element wrt. relational composition.

A box ρ represents the language $\mathcal{L}(\rho) = \{w \in \mathcal{L}(\rho) \mid \forall q, q' \in Q \colon q \xrightarrow{w} q' \text{ iff } (q, q') \in \rho\}$. That is, the words induce the state changes specified by the box. Hence, $\mathcal{L}(\rho)$ is an equivalence class of $\sim_A$, finitely represented by ρ. The function $\rho_- : T^* \to \mathrm{M}_A$ maps w to the unique box ρ_w representing the word, $w \in \mathcal{L}(\rho_w)$. More explicitly, $\rho_\varepsilon =$ id, $\rho_a = \{(q, q') \mid q \xrightarrow{a} q'\}$ for all $a \in T$, and $\rho_{uv} = \rho_u; \rho_v$. The image ρ_{T^*} contains exactly the boxes ρ with $\mathcal{L}(\rho) \neq \emptyset$. Figure 1 illustrates the representation of words as boxes.

The terminal words generated by maximal plays are represented by boxes, disjunction gives the alternatives of refuter, and conjunction expresses the options for prover. The set of plays from a given position is thus represented by a formula F from the set BF_A of *negation-free Boolean formulas over the transition monoid* (propositions are boxes). This set includes the unsatisfiable formula *false*. We use the rules $\mathit{false} \wedge F = F \wedge \mathit{false} = \mathit{false}$ and $\mathit{false} \vee F = F \vee \mathit{false} = F$ to evaluate conjunctions and disjunctions involving *false* on the syntactical level. As a consequence, *false* is the only syntactic representation of the unsatisfiable formula. This will simplify the definition of relational composition. From now on and without further mentioning, F and G will refer to formulas from BF_A.

Our goal is to decide whether refuter can force the plays from an initial position to end in a terminal word rejected by A. To mimic this, we define a formula to be *rejecting* if it is satisfied under the assignment $\nu : \mathrm{M}_A \to \{\mathit{true}, \mathit{false}\}$ such that $\nu(\rho) = \mathit{true}$ if and only if ρ does not contain a pair (q_0, q_f) with $q_f \in Q_F$.

To use formulas in a Kleene iteration, we have to define a partial ordering on them. Intuitively, F should be smaller than G if G makes it easier for refuter to win. Taking the logical perspective, it is easier for refuter to win if F implies G. Implication on BF_A is not antisymmetric. To factor out the symmetries, we reason modulo logical equivalence, $\mathrm{BF}/_\Leftrightarrow$. Every formula is understood as a representative of the class of logically equivalent formulas of BF_A. Extending $\Rightarrow$ to $\mathrm{BF}/_\Leftrightarrow$ by comparing representatives then yields a partial order. The least element of the partial order is the equivalence class of *false*.

Operations. We combine formulas by conjunction, disjunction, and by an operation of relational composition that lifts ; from the transition monoid to formulas over boxes. To explain the definition of relational composition, note that every finite maximal play from $\alpha\beta$ proceeds in two phases. It starts with a maximal play turning α into a terminal word, say w, followed by a play from $w\beta$. Since there are no more derivations for w, the play from $w\beta$ coincides with a play from β, except that all sentential forms have a prefix w.

Let F and G represent all plays starting in α and β, respectively. In F, terminal words like w are represented by boxes ρ. We append the plays from β by replacing ρ with $\rho; G$. To take into account all plays from α, we do this replacement for all boxes in F. It remains to add the prefix w to the sentential forms in the plays from β. In $\rho;G$, every box τ in G is replaced by $\rho;\tau$. The so-defined formula $F;G$ will represent all plays from $\alpha\beta$.

▶ **Example 4.** Let $F = \rho_a \vee \rho_b$ and $G = \rho_c \wedge \rho_d$. We have $(\rho_a \vee \rho_b);(\rho_c \wedge \rho_d) = \rho_a;(\rho_c \wedge \rho_d) \vee \rho_b;(\rho_c \wedge \rho_d) = (\rho_a;\rho_c \wedge \rho_a;\rho_d) \vee (\rho_b;\rho_c \wedge \rho_b;\rho_d)$. The first equality replaces ρ_a and ρ_b by $\rho_a;G$ and $\rho_b;G$, respectively. The second equality prefixes ρ_c and ρ_d in G by the corresponding box ρ_a or ρ_b.

▶ **Definition 5.** *Relational composition* over BF_A is defined by $F;\mathit{false} = \mathit{false};G = \mathit{false}$ and for composite formulas ($\star \in \{\wedge, \vee\}$, $\rho \in \mathrm{M}_A$) by

$$(F_1 \star F_2);G = F_1;G \star F_2;G \qquad \text{and} \qquad \rho;(G_1 \star G_2) = \ \rho;G_1 \star \rho;G_2\ .$$

Note that the composition of two non-*false* formulas is not *false*. Therefore, the result of a relational composition is *false* if and only if at least one of the arguments was *false*.

Relational composition equips the set of formulas with a monoid structure. In particular, relational composition is associative. For a fixed-point iteration, the operations also have to be monotonic wrt. $\Rightarrow$. For conjunction and disjunction, monotonicity obviously holds.

▶ **Lemma 6.** *If $F \Rightarrow F'$ and $G \Rightarrow G'$, then $F;G \Rightarrow F';G'$.*

We lift the three operations to $\Leftrightarrow$-equivalence classes by applying them to arbitrary representatives. Since implication is transitive, monotonicity of the operations ensures well-definedness. Moreover, the operations still behave monotonically on $\mathrm{BF}/_{\Leftrightarrow}$. From now on, we can thus identify formulas with the classes they represent.

System of Equations. We introduce one variable Δ_X for each non-terminal $X \in N$. Terminals $a \in T$ yield boxes, and we write Δ_a for ρ_a. We lift the notation $\Delta_{\_}$ to sentential forms: $\Delta_\varepsilon = \mathrm{id}$ and $\Delta_{\alpha\beta} = \Delta_\alpha;\Delta_\beta$. This means concatenation in rules is replaced by relational composition. All rules for the same non-terminal are combined into one equation using disjunction or conjunction, depending on who is the owner of the non-terminal.

▶ **Definition 7.** The *system of equations (over* $\mathrm{BF}/_{\Leftrightarrow}$*) induced by G and A* has one equation for each non-terminal $X \in N_{\star}$ with $\star \in \{\bigcirc, \Box\}$. If $X \to \eta_1, \dots, X \to \eta_k$ are all rules with X as their left-hand side, the equation is $\Delta_X = \Delta_{\eta_1} \wedge \cdots \wedge \Delta_{\eta_k}$ if $X \in N_\Box$ and $\Delta_X = \Delta_{\eta_1} \vee \cdots \vee \Delta_{\eta_k}$ if $X \in N_\bigcirc$.

With Lemma 6, for each non-terminal X we can understand the right-hand side of the associated equation as a monotonic function $f_X : (\mathrm{BF}/_{\Leftrightarrow})^N \to \mathrm{BF}/_{\Leftrightarrow}$. It takes as input a vector of formulas (one for each non-terminal) and computes a new formula for Δ_X. We combine the functions for each non-terminal to a single function $f : (\mathrm{BF}/_{\Leftrightarrow})^N \to (\mathrm{BF}/_{\Leftrightarrow})^N$. It is monotonic on the product domain wrt. the product order $\Rightarrow^N$.

Since $\mathrm{BF}/_{\Leftrightarrow}$ with $\Rightarrow$ is a finite bottomed partial order, there is a unique least solution σ for the equation $\Delta = f(\Delta)$, namely $\sigma = \bigsqcup_{i \in \mathbb{N}} f^i(\bot)$ [15]. The least element of the product domain is the vector with the $\Leftrightarrow$-equivalence class of *false* in every component. Note that the solution is computed by iteratively applying f until a fixed point is reached. This procedure terminates since the chain $\bot \Rightarrow^N f(\bot) \Rightarrow^N f(f(\bot)) \Rightarrow^N \dots$ stabilizes on a finite domain.

The solution $\sigma : N \to \mathrm{BF}/_{\Leftrightarrow}$ yields a value σ_X for each non-terminal $X \in N$. We lift the notation to sentential forms by $\sigma_\varepsilon = \mathrm{id}$, $\sigma_a = \rho_a$ for all $a \in T$, and $\sigma_{\alpha\beta} = \sigma_\alpha;\sigma_\beta$. From now on, σ will always be the least solution to a system of equations. The system will be clear from the context (either G, A from the development or G_{ex}, A_{ex} from the running example).

▶ **Example 8.** For G_{ex} and A_{ex} from Example 3, the system of equations consists of $\Delta_X = \Delta_a; \Delta_Y \vee \Delta_\varepsilon = \rho_a; \Delta_Y \vee \mathrm{id}$ and $\Delta_Y = \Delta_b; \Delta_X = \rho_b; \Delta_X$. Its least solution is $\sigma_X = \mathrm{id} \vee \rho_{ab}$ and $\sigma_Y = \rho_b$. To terminate the iteration, use $\rho_{bab} = \rho_b$.

The main result in this section states that the fixed point σ_α is equivalent to the tree $\mathcal{T}_\alpha$ of all plays starting in α. To be precise, we understand the tree as a (typically infinite) formula where inner nodes owned by refuter yield disjunctions, prover's nodes are conjunctions, and terminal words are boxes. For the proof (see [25]), we develop machinery for infinite formulas.

▶ **Theorem 9.** $\mathcal{T}_\alpha \Leftrightarrow \sigma_\alpha$.

4 Winning Regions and Strategy Synthesis

Define the set of sentential forms $W^{\subseteq \mathcal{L}(A)} = \{\alpha \in \vartheta \mid \sigma_\alpha \text{ is not rejecting }\}$ and denote the complement by $W^{\not\subseteq \mathcal{L}(A)} = \vartheta \setminus W^{\subseteq \mathcal{L}(A)} = \{\alpha \in \vartheta \mid \sigma_\alpha \text{ is rejecting }\}$. Our goal is to prove the following result in a constructive way, by synthesizing strategies guided by the fixed-point solution to the system of equations.

▶ **Theorem 10.** *Inclusion games are determined:* $\vartheta = W^{\subseteq \mathcal{L}(A)} \cupdot W^{\not\subseteq \mathcal{L}(A)}$*, where* $W^{\subseteq \mathcal{L}(A)}$ *is the winning region of prover and* $W^{\not\subseteq \mathcal{L}(A)}$ *is the winning region of refuter.*

As a consequence, it is *decidable* whether a given sentential form α is winning for a player: Compute the formula σ_α and evaluate it under ν to check whether it is rejecting.

It has been shown in [36] that for all games on pushdown systems with ω-regular winning conditions, the winning regions are regular. Indeed, the winning region of the non-inclusion game can be accepted by a deterministic automaton. The set of equivalence classes of formulas forms its set of control states, the equivalence class of id is the initial state and rejecting formulas are final states. If the automaton in state F reads symbol $x \in N \cup T$, it switches to the state $F; \sigma_x$.

Representing sentential forms by formulas is too imprecise to do strategy synthesis. (In fact, the leftmost non-terminal is not even encoded in the formula.) Since relational composition is associative, we can represent the set of all sentential forms $\alpha = wX\beta$ by a set of triples $(\sigma_w, X, \sigma_\beta)$, where σ_w and σ_β are taken from a finite set of formulas (up to logical equivalence) and X is a non-terminal from a finite set. This finite representation will be sufficient for the strategy synthesis. Our synthesis operates on normalized formulas in CNF.

Conjunctive Normal Form. A formula in CNF is a conjunction of clauses, each clause being a disjunction of boxes. We use set notation and write clauses as sets of boxes and formulas as sets of clauses. The set of *CNF-formulas* over M_A is thus $\mathrm{CNF}_A = \mathcal{P}(\mathcal{P}(\mathrm{M}_A))$. Identify *true* $= \{\}$ and *false* $= \{\{\}\}$. In this section, all formulas will stem from CNF_A.

When computing a disjunction, we have to apply distributivity to obtain a CNF.

▶ **Lemma 11.** $F \vee G \Leftrightarrow \{K \cup H \mid K \in F, H \in G\}$ *and* $F \wedge G \Leftrightarrow F \cup G$.

As the result of the relational composition $F; G$ of CNF-formulas, we obtain a formula with three alternations between conjunction and disjunction. We apply distributivity to normalize $F; G$. Lemma 13 gives a closed-form representation of the result. To understand the idea, consider the composition of one clause with a CNF.

▶ **Example 12.** Consider $F; G = (\rho_a \vee \rho_b); (\rho_c \wedge \rho_d) = (\rho_a; \rho_c \wedge \rho_a; \rho_d) \vee (\rho_b; \rho_c \wedge \rho_b; \rho_d)$. Distributivity yields $(\rho_a; \rho_c \vee \rho_b; \rho_c) \wedge (\rho_a; \rho_c \vee \rho_b; \rho_d) \wedge (\rho_a; \rho_d \vee \rho_b; \rho_c) \wedge (\rho_a; \rho_d \vee \rho_b; \rho_d)$.

To turn $F;G$ to CNF, we normalize the composition $K;G$ for every clause $K \in F$. The formula $K;G$ is an alternation of disjunction (not in the example), conjunction, and disjunction. Distributivity, when applied to the topmost two operations, selects for every box $\rho \in K$ a clause $H \in G$ to compose ρ with. This justifies the following set-theoretic characterization.

▶ **Lemma 13.** $F;G \Leftrightarrow \bigcup_{K\in F} \bigcup_{z:K\to G} \big\{ \bigcup_{\rho\in K} \rho; z(\rho)\big\}$.

Inclusion (for Prover). Prover wins on infinite plays, and therefore does not have to care about termination. This yields a simple positional winning strategy. We focus on the more complex case of refuter.

▶ **Theorem 14.** *The strategy $s^{\subseteq\mathcal{L}(A)}$ that applies a rule such that the formula for the resulting position is not rejecting (if possible) is a winning strategy for prover for the inclusion game from all positions in $W^{\subseteq\mathcal{L}(A)}$.*

Non-Inclusion (for Refuter). A CNF-formula is rejecting iff for each clause chosen by prover, refuter can select a rejecting box in this clause. We formalize the selection process using the notion of choice functions. A *choice function* on $F \in \mathrm{CNF}_A$ is a function $c : F \to \mathrm{M}_A$ selecting a box from each clause, $c(K) \in K$ for all $K \in F$. We show that there is a strategy for refuter to derive a terminal word from one of the chosen boxes. In particular, the strategy will only generate finite plays. Note that a choice function can only exist if F does not contain the empty clause. Otherwise, the formula is equivalent to *false*, and refuter cannot enforce termination of the derivation process.

We show that by appropriately selecting the moves of refuter, we can refine the choice function along each play, independent on the choices of prover. Given a choice function c on a CNF-formula F, a choice function c' on G *refines* c if $\{c'(H) \mid H \in G\} \subseteq \{c(K) \mid K \in F\}$, denoted by $c'(G) \subseteq c(F)$. Given equivalent CNF-formulas, a choice function on the one can be refined to a choice function on the other formula. Hence, we can deal with representative formulas in the following development.

▶ **Lemma 15.** *Consider $F \Rightarrow G$. For any choice function c on F, there is a choice function c' on G that refines it.*

The strategy for refuter has to enforce termination. To this end, we consider formulas obtained from Kleene approximants. When composing the formulas for the non-terminals to obtain a formula for a sentential form, we use an intermediary solution of the Kleene iteration instead of the fixed-point solution. Define a *sequence of levels lvl* associated to a sentential form α to be a sequence of natural numbers of the same length as α. The formula σ_α^{lvl} corresponding to α and *lvl* is defined by $\sigma_a^i = \{\{\rho_a\}\}$ for all $a \in T \cup \{\varepsilon\}$, σ_X^i the solution to X from the i^{th} Kleene iteration, and $\sigma_{\alpha.\beta}^{lvl.lvl'} = \sigma_\alpha^{lvl}; \sigma_\beta^{lvl'}$. A choice function for α and *lvl* is a choice function on σ_α^{lvl}. Note that σ_a^i is independent of i for terminals a. Moreover, there is an i_0 so that $\sigma_X^{i_0} = \sigma_X$ for all non-terminals X. This means a choice function on σ_α can be understood as a choice function on $\sigma_\alpha^{i_0}$. Here, we use a single number i_0 to represent a sequence $lvl = i_0 \ldots i_0$ of the appropriate length.

By definition, σ_X^0 is *false* for all non-terminals, and *false* propagates through relational composition by definition. We combine this observation with the fact that choice functions do not exist on formulas that are equivalent to *false*.

▶ **Lemma 16.** *If there is a choice function for α and lvl, then lvl does not assign zero to any non-terminal X in α.*

The lemma has an important consequence. Consider a sentential form α with an associated sequence $\mathit{lvl} \in 0^*$ and a choice function c for α and lvl. Then α has to be a terminal word, $\alpha = w \in T^*$, $\sigma_\alpha^{\mathit{lvl}} = \{\{\rho_w\}\}$, and the choice function has to select ρ_w. In particular, w itself forms a maximal play from this position on, and indeed the play ends in a word whose box is contained in the image of the choice function.

Consider now $\alpha = wX\beta$ and lvl an associated sequence of levels. Assume lvl assigns a positive value to all non-terminals. Let j be the position of X in α and let $i = \mathit{lvl}_j$ be the corresponding entry of lvl. We split $\mathit{lvl} = \mathit{lvl}'.i.\mathit{lvl}''$ into the prefix for w, the entry i for X, and the suffix for β. For each rule $X \to \eta$, we define $\mathit{lvl}_\eta = \mathit{lvl}'.(i-1)\ldots(i-1).\mathit{lvl}''$ to be the sequence associated to $w\eta\beta$. It coincides with lvl on w and β and has entry $i-1$ for all symbols in η. Note that for a terminal word, the formula is independent of the associated level, so we have $\sigma_{wX}^{\mathit{lvl}'.i} = \sigma_{wX}^i$ and $\sigma_{w\eta}^{\mathit{lvl}'.(i-1)\ldots(i-1)} = \sigma_{w\eta}^{i-1}$.

We show that we can (1) always refine a choice function c on $\sigma_\alpha^{\mathit{lvl}}$ along the moves of prover and (2) whenever it is refuter's turn, pick a specific move to refine c.

▶ **Lemma 17.** *Let c be a choice function for $\alpha = wX\beta$ and lvl.*
(1) *If $X \in N_\square$, for all $X \to \eta$ there is a choice function c_η for $w\eta\beta$ and lvl_η that refines c.*
(2) *If $X \in N_\bigcirc$, there is $X \to \eta$ and a choice function c_η for $w\eta\beta$ and lvl_η that refines c.*

Proof. We prove (2). We show that there is a rule $X \to \eta$ and a choice function c_η on $\sigma_{w\eta\beta}^{\mathit{lvl}_\eta}$ refining c. Towards a contradiction, assume this is not the case. Then for each rule $X \to \eta$, there is at least one clause K''_η of $\sigma_{w\eta\beta}^{\mathit{lvl}_\eta}$ that does not contain a box in the image of c. By Lemma 13, this clause is defined by a clause $\rho_w; K'_\eta$ of $\sigma_{w\eta}^{i-1}$ and a function z_η mapping the boxes from this clause to $\sigma_\beta^{\mathit{lvl}''}$.

We have $\sigma_X^i = \bigvee_{X\to\eta} \sigma_\eta^{i-1}$. A clause of σ_{wX}^i is thus (Lemma 11) of the form $K = \rho_w; (\bigcup_{X\to\eta} K_\eta) = \bigcup_{X\to\eta} \rho_w; K_\eta$, where each K_η is a clause of σ_η^{i-1}. We construct the clause $K' = \rho_w; (\bigcup_{X\to\eta} K'_\eta)$ of σ_{wX}^i using the K'_η from above. On this clause, we define the map $z' = \bigcup_{X\to\eta} z_\eta$ that takes a box $\rho_w; \rho \in \rho_w; K'_\eta$ and returns $z_\eta(\rho_w; \rho)$. (If a box $\rho_w; \rho$ is contained in $\rho_w; K'_\eta$ for several η, pick an arbitrary η among these.) By Lemma 13, K' and z' define a clause of $\sigma_\alpha^{\mathit{lvl}}$. The choice function c selects a box $\rho_w; \rho; \tau$ out of this clause, where there is a rule $X \to \eta$ such that $\rho \in K'_\eta$ and $\tau \in z'(\rho_w; \rho) = z_\eta(\rho_w; \rho)$. This box is also contained in K''_η. A contradiction to the assumption that no box from K''_η is in the image of c. ◀

Notice that the sequence lvl_η is smaller than lvl in the following ordering $\prec$ on $\mathbb{N}^*$. Given $v, w \in \mathbb{N}^*$, we define $v \prec w$ if there are decompositions $v = xyz$ and $w = xiz$ so that $i > 0$ is a positive number and $y \in \mathbb{N}^*$ is a sequence of numbers that are all strictly smaller than i. Note that requiring i to be positive will prevent the sequence xz from being smaller than $x0z$, since we are not allowed to replace zeros by ε.

The next lemma states that $\prec$ is well founded. Consequently, the number of derivations $wX\beta \Rightarrow w\eta\beta$ following the strategy that refines an initial choice function will be finite.

▶ **Lemma 18.** *$\prec$ on $\mathbb{N}^*$ is well founded with minimal elements 0^*.*

Lemma 18 is used in the main technical result of this section. Proposition 19 in particular says that all maximal plays that conform to $s_{\alpha,c}$ are finite. If σ_α is rejecting, there is a choice function on σ_α that only selects rejecting boxes. The desired theorem is then immediate.

▶ **Proposition 19.** *Let c be a choice function on σ_α. There is a strategy $s_{\alpha,c}$ such that all maximal plays starting in α that conform to $s_{\alpha,c}$ end in a terminal word w with $\rho_w \in c(\sigma_\alpha)$.*

Proof. We show the following stronger claim: Given any triple consisting of a sentential form α, an associated sequence of levels lvl, and a choice function c for α and lvl, there is a strategy $s_{\alpha,c}$ such that all maximal plays conform to it and starting in α end in a terminal word w with $\rho_w \in \{c(K) \mid K \in \sigma_\alpha\}$. This proves the proposition by choosing α and c as given and $lvl = i_0 ... i_0$, where $i_0 \in \mathbb{N}$ is a number such that $\sigma = \sigma^{i_0}$.

To show the claim, note that $\prec$ on $\mathbb{N}^*$ is well founded and the minimal elements are exactly 0^* by Lemma 18, and $lvl_\eta \prec lvl$. This means we can combine Lemma 16 and Lemma 17 (for the step case) into a Noetherian induction. The latter lemma does not state that lvl_η assigns a positive value to each non-terminal, which was a requirement on lvl. This follows from Lemma 16 and the fact that c_η is a choice function. The strategy $s_{\alpha,c}$ for refuter always selects the rule that affords a refinement of the initial choice function c. ◀

▶ **Theorem 20.** *Let $\alpha \in W^{\not\subseteq \mathcal{L}(A)}$ and let c select a rejecting box in each clause of σ_α. Then $s_{\alpha,c}$ is a winning strategy for refuter for the non-inclusion game played from α.*

There are several ways of implementing a winning strategy for refuter. First, the strategy $s_{\alpha,c}$ from Proposition 19 can be implemented using bounded memory, but one then needs linear time to decide which move to pick. Alternatively, it can be implemented using a pushdown, which then only needs constant time to pick moves. Second, $s_{\alpha,c}$ is not positional, but one can obtain a position strategy by a breadth-first search in the tree of all plays from α.

▶ **Example 21.** In the running example, formula $\sigma_Y = \{\{\rho_b\}\}$ is rejecting. In fact, refuter can win the non-inclusion game played from Y. The initial choice function on σ_Y has to be $c(\{\rho_b\}) = \rho_b$. In the first step, prover has no alternative but $Y \to bX$. Position bX has the formula $\sigma_{bX} = \{\{\rho_b\}\}; \{\{\mathrm{id}, \rho_{ab}\}\} = \{\{\rho_b, \rho_{bab}\}\} = \{\{\rho_b\}\}$. Pick the same choice function as before. The rule $X \to \varepsilon$ causes id to enter σ_X in the first Kleene step. This causes ρ_b to enter σ_{bX} also in the first step. Indeed, by choosing $X \to \varepsilon$ refuter wins non-inclusion.

5 Complexity

We show that deciding whether refuter has a winning strategy for non-inclusion from a given position is a 2EXPTIME-complete problem. Moreover, the algorithm presented in the previous sections achieves this optimal time complexity. Our proof of the lower bound follows the proof of the analogue result for the games considered in [30].

▶ **Theorem 22.** *Given a non-inclusion game and an initial position, deciding whether refuter has a winning strategy from the specified position is* 2EXPTIME*-hard.*

The following algorithm implements the fixed-point iteration discussed in Section 3, executed on formulas in CNF (see the paragraph on CNF of Section 4).

▶ **Algorithm 23.** Given a non-inclusion game and an initial position α, the following algorithm computes whether refuter has a winning strategy from the given position.

1. Set $\sigma^0_X = \mathit{false}$ for all $X \in N$. Set $i = 0$.
2. Do until $\sigma^i_X \Leftrightarrow \sigma^{i-1}_X$ for all $X \in N$: $i = i + 1$; $\sigma^i = f(\sigma^{i-1})$.
3. Compute σ_α, and return *true* iff σ_α is rejecting.

Here, f is the function combining the right-hand sides of the equations as in Definition 7.

▶ **Theorem 24.** *Given a non-inclusion game and an initial position, Algorithm 23 computes whether refuter has a winning strategy from the given position in time $\mathcal{O}\Big(|G|^2 \cdot 2^{2^{|Q|^{c_1}}} + |\alpha| \cdot 2^{2^{|Q|^{c_2}}}\Big)$ for some constants $c_1, c_2 \in \mathbb{N}$.*

The following corollary is an immediate consequence of Theorem 22 and Theorem 24.

▶ **Corollary 25.** *Deciding whether refuter has a winning strategy for a given non-inclusion game and an initial position is* 2EXPTIME*-complete.*

One should note that the running time of the algorithm is only exponential in the size of the automaton. If the automaton is assumed to be fixed, the running time of the algorithm is polynomial in the size of the grammar and in the length l of the initial position. Namely, it can be executed in $\mathcal{O}(|G|^2 + l)$ steps.

Furthermore, the algorithm can solve games on the game arena induced by a grammar not only in the case of the non-inclusion winning condition, but also in a more general setting, e.g. if the winning condition is deriving a word in the regular target language after finitely many steps.

6 Experiments

We have implemented our algorithm in C++ [1] and compared it to an implementation of Cachat's algorithm [13] for games on pushdown systems. Cachat's input instances consist of a pushdown system P with an ownership partitioning on the control states and an alternating finite automaton over the stack alphabet of P (P-AFA). The first player wins by enforcing a run into a configuration accepted by the P-AFA. Cachat's algorithm constructs the winning region of the first player by saturating the automaton.

To convert instances of our game to that of Cachat, we construct a pushdown system P that encodes both the grammar G and the target automaton A. A sentential form $wX\beta$ (where X is the leftmost non-terminal) will be represented in P by a configuration $(Q, X\beta)$, where Q is the set of states A can be in after processing w. The translation thus embeds a determinized version of A in P. This may cause an exponential blow-up in the size of the input instance, which reflects the worst case complexity: Our problem is 2EXPTIME-complete while Cachat's algorithm is exponential.

For the experiments, we generated random automata using the Tabakov-Vardi model [40]. The generator is parameterized in the number of letters and control states, the percentage of final states, and the number of transitions per letter (given as a fraction of the number of states). We adapt the model to generate also grammars with rules of the form $X \to aYb$, with parameters being the number of rules and non-terminals for each player, and the chances of a, Y, and b to be present. Since sparse automata and grammars are likely to yield simpler instances, we focus on dense examples.

For the parameters $|Q|$, $|T|$, $|N_\bigcirc|$ and $|N_\Box|$, we tried out several combinations. The entry $x/y/z$ in the table below stands for $|Q| = x, |T| = y, |N_\bigcirc| = |N_\Box| = z$. For each combination, we generated 50 random automata and grammars, applied three algorithms to them, and measured how many instances could be solved within 10 seconds and how much time was consumed for the instances that could be solved on average.

We compared: (1) Our algorithm with a naive Kleene iteration, i.e. all components of the current solution are updated in each step. (2) Our algorithm with chaotic iteration implemented using a worklist, i.e. only components whose dependencies have been updated are modified. This is the common way of implementing a Kleene iteration. (3) Cachat's algorithm applied to our problem as described above. To improve the runtime, the target automaton has been determinized and minimized before creating the pushdown system.

We ran our experiments on an Intel i7-6700K, 4GHz. The durations are milliseconds.

Already the naive implementation of Kleene iteration outperforms Cachat's algorithm, which was not able to solve any instance with parameters greater than `10/15/15`. The worklist implementation is substantially faster, by three orders of magnitude on average. This confirms our hypothesis: The stack content is more information than needed for safety verification, and getting rid of it by moving to the summary domain speeds up the analysis.

One can also implement Cachat's algorithm using a worklist, but due to the shape of the instances resulting from the reduction (deterministic automata), this is not helpful. In our experiments, the worklist variant was slower by at least one order of magnitude, even on small examples.

	naive Kleene		worklist Kleene		Cachat	
	avg	%t/o	avg	%t/o	avg	%t/o
5/ 5/ 5	65.2	2	0.8	0	94.7	0
5/ 5/10	5.4	4	7.4	0	701.7	0
5/10/ 5	13.9	0	0.3	0	375.7	0
5/ 5/15	6.0	0	1.1	0	1618.6	0
5/10/10	32.0	2	122.1	0	2214.4	0
5/15/ 5	44.5	0	0.2	0	620.7	0
5/ 5/20	3.4	0	1.4	0	3434.6	4
5/10/15	217.7	0	7.4	0	5263.0	16
10/ 5/ 5	8.8	2	0.6	0	2737.8	2
10/ 5/10	9.0	6	69.8	0	6484.9	66
15/ 5/ 5	30.7	0	0.2	0	5442.4	52
10/10/ 5	9.7	0	0.2	0	7702.1	92
10/15/15	252.3	0	1.9	0	n/a	100
10/15/20	12.9	0	1.8	0	n/a	100

7 Algorithmic Considerations

We discuss how to further speed-up the worklist implementation by two heuristics prominent in verification: Lazy evaluation [17] and antichains [19, 3, 4]. The heuristics are not meant to be a contribution of the paper and they are not yet implemented. The point is to demonstrate that the proposed summary domain combines well with algorithmic techniques. For both heuristics, it is not clear to us how to apply them to the domain of alternating automata.

The idea of *lazy evaluation* is to keep composed formulas $(F \vee F'); M$ symbolic, i.e. we store them as a term rather than computing the resulting formula. When having to evaluate the formula represented by the term, we only compute up to the point where the value influences the overall answer. Consider the test whether $(F \vee F'); M$ is rejecting. If already $F; M$ is rejecting, the whole formula represented by the term will be rejecting and the evaluation of $F'; M$ can be skipped.

The idea of the *antichain optimization* is to identify representative elements in the search space that allow us to draw conclusions about all other elements. Here, the search space consists of formulas (representing the intermediate steps of the fixed-point computation). By Lemma 6, it is sufficient to reason modulo logical equivalence. This allows us to remove redundant disjuncts and conjuncts, in particular, if $F \Rightarrow G$ we can prune F from $F \vee G$ and G from $F \wedge G$. When reasoning over CNFs, this removes from a formula all clauses that are subsumed by other clauses. It is thus enough to store the CNFs in the form of antichains of $\subseteq$-minimal clauses. The antichain approach benefits from a weaker notion of implication.

In Boolean satisfiability, the antichain optimization corresponds to the subsumption rule, and it is known to have a limited impact on the performance of solvers. The setting we consider, however, is different. Our formulas are enriched during the computation by new clauses (that are not derived from others as in SAT). The antichain optimization can therefore be expected to yield better results for inclusion games and, in fact, has been successfully implemented for automata models [19, 3, 4].

8 Related Work

We already discussed the relation with Cachat's work [13]. Walukiewicz [42] studies games given by a pushdown automaton with a parity function on the states. Similar to our case, the derived strategies are implementable using a stack. The problem [42] is concerned with is different from ours in several respects. The game aspect is given by the specification (a μ-calculus formula), not by the system as in our case. Moreover, (infinite) parity games are generally harder than safety: [42] is exponential both in the system and in the specification, while our construction is exponential only in the specification. Piterman and Vardi [31] study a similar variant of the problem and come up with a solution originating in the automata-theoretic approach [28].

Walukiewicz reduces solving parity games on the infinite computation tree of a pushdown system to solving parity games on a finite graph. To do so, instead of the full stack, only the topmost stack symbol is stored. Whenever a push should be executed, one player guesses the behavior of the game until the corresponding pop, i.e. she proposes a set of control states. The other player can decide to skip the subgame between push and pop by selecting a control state from the set, and the game continues. Alternatively, she can decide to verify the subgame. In this case, the new symbol becomes top-of-stack, and the game continues until it is popped. After the pop, the game ends, and which player wins is dependent on whether the current control state is in the proposed set of states.

This approach can be applied to a context-free game to reduce it to a reachability game on a doubly-exponentially-sized graph. Before applying a rule to the leftmost non-terminal X, we let refuter propose a set of boxes that describes the effect of terminal words derivable from X. Prover can either accept the proposal and select one of the boxes, or she verifies the proposal. In the latter case, the rest of the sentential form can be dropped. A winning strategy for refuter in the finite game has to guess the effect of each non-terminal, while our method deterministically computes it: The guessed effects that will not lead to refuter losing the subgame are exactly the sets of boxes occurring as the image of a choice function.

The work [30] considers active context-free games where in each turn, player A picks the position of a non-terminal in the current sentential form and player B picks the rule that is applied to the non-terminal. It is shown to be undecidable whether player A can enforce the derivation of a word in a regular language. If one limits the moves of player A to left-to-right strategies (skipped non-terminals cannot be touched again, the regular target language may contain non-terminals), one obtains a game that is closely related to our setting. In fact, the authors show that allowing player A to pick the rules for some of the non-terminals does not increase the expressive power. Therefore, there are polynomial-time reductions of our type of game to their type of game and vice versa. In [30], the focus lies on establishing the lower bounds for the time complexity of various type of active context-free games. The authors show that deciding the existence of a left-to-right winning strategy is 2EXPTIME-complete, like the problem considered in this paper (Section 5). The upper bound is shown by using an exponential-time reduction to Walukiewicz [42], and they also present an optimal algorithm that uses Cachat's algorithm for pushdown systems. Our algorithm also has optimal time complexity, but contrary to [30], it is based on procedure summaries rather than on saturation. The lower bound is shown by encoding an alternating Turing machine with exponential space as a grammar game, and we adapted their proof to show Theorem 22 in [25]. [30] was further elaborated on and generalized in [8, 34].

Methods for solving variants of pushdown games, related mostly to saturation (see [14] for a survey on saturation-based methods), are implemented in several tools. [10] targets higher-order pushdown systems, related to it is [11], [39] implements an optimized saturation-based method, [23] solves the full case of parity games. [32] implements a type directed algorithm not based on saturation. None of the tools implements procedure summaries, but some can be used to solve instances of our problem. We plan to carry out a thorough comparison with these implementations in the future.

Antichain heuristics, discussed in Section 7, were developed in the context of finite automata and games [43, 44], and generalized to Büchi automata [19, 3, 4] with a fixed point over sets of boxes. Our lazy evaluation is inspired by [17]. Our framework is compatible with techniques for reachability in well-structured transition systems (WSTS) that proceed backwards [2]. We believe that techniques like [27, 20, 26, 5, 21] can be adapted to our setting. To instantiate general WSTS reachability algorithms, the ordering of configurations would be based on implication among formulas, the target set would be the upward closure of the assignment σ where σ_S is the conjunction of all rejecting boxes and $\sigma_X = \mathit{false}$ for every other $X \in N$, and the initial state would be the assignment $\perp$. Another interesting possibility would be to adapt Newton iteration [16].

The transition monoid can be traced back at least to Büchi [12], and was prominently used e.g. in [38].

Acknowledgments. We thank Olivier Serre, Matthew Hague, Georg Zetzsche, and Emanuele D'Osualdo for helpful discussions. We thank the reviewers for their feedback.

References

1 Implementation of our algorithm. Published: 2016-17-07. URL: `https://concurrency.informatik.uni-kl.de/rigg.html`.

2 P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313–321, 1996.

3 P. A. Abdulla, Y. Chen, L. Clemente, L. Holík, C.-D. Hong, R. Mayr, and T. Vojnar. Simulation subsumption in Ramsey-based Büchi automata universality and inclusion testing. In *CAV*, volume 6174 of *LNCS*, pages 132–147. Springer, 2010.

4 P. A. Abdulla, Y. Chen, L. Clemente, L. Holík, C.-D. Hong, R. Mayr, and T. Vojnar. Advanced Ramsey-based Büchi automata inclusion testing. In *CONCUR*, volume 6901 of *LNCS*, pages 187–202. Springer, 2011.

5 P. A. Abdulla, F. Haziza, and L. Holík. All for the price of few. In *VMCAI*, volume 7737 of *LNCS*, pages 476–495. Springer, 2013.

6 D. Beyer. Software verification and verifiable witnesses (report on sv-comp). In *TACAS*, volume 9035 of *LNCS*, pages 401–416. Springer, 2015.

7 D. Beyer. Reliable and reproducible competition results with benchexec and witnesses (report on sv-comp). In *TACAS*, volume 9636 of *LNCS*, pages 887–904. Springer, 2016.

8 H. Björklund, M. Schuster, T. Schwentick, and J. Kulbatzki. On optimum left-to-right strategies for active context-free games. In *ICDT*, pages 105–116. ACM, 2013.

9 A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997.

10 C. Broadbent, A. Carayol, M. Hague, and Olivier O. Serre. C-SHORe: A collapsible approach to higher-order verification. *ACM SIGPLAN Notices*, 48(9):13–24, 2013.

11 C. Broadbent and N. Kobayashi. Saturation-Based Model Checking of Higher-Order Recursion Schemes. In *CSL*, volume 23 of *LIPIcs*, pages 129–148. Dagstuhl, 2013.

12 J. R. Büchi. *On a Decision Method in Restricted Second Order Arithmetic*, pages 425–435. Springer, 1990.

13 T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *ICALP*, volume 2380 of *LNCS*, pages 704–715. Springer, 2002.

14 A. Carayol and M. Hague. Saturation algorithms for model-checking pushdown systems. In *AFL*, volume 151 of *EPTCS*, pages 1–24, 2014.

15 B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. CUP, 1990.

16 J. Esparza, S. Kiefer, and M. Luttenberger. Newtonian program analysis. *JACM*, 57(6), 2010.

17 J. Fiedor, L. Holík, P. Jankøu, O. Lengál, and T. Vojnar. Lazy automata techniques for WS1S. Technical Report FIT-TR-2016-01, Brno University of Technology, 2016.

18 A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. *ENTCS*, 9:27–37, 1997.

19 S. Fogarty and M. Y. Vardi. Efficient Büchi universality checking. In *TACAS*, volume 6015 of *LNCS*, pages 205–220. Springer, 2010.

20 Z. Ganjei, A. Rezine, P. Eles, and Z. Peng. Lazy constrained monotonic abstraction. In *VMCAI*, volume 9583 of *LNCS*, pages 147–165. Springer, 2016.

21 P. Ganty, J.-F. Raskin, and L. Begin. A complete abstract interpretation framework for coverability properties of WSTS. In *VMCAI*, pages 49–64. Springer, 2006.

22 M. Hague and C.-H. L. Ong. Winning regions of pushdown parity games: A saturation method. In *CONCUR*, volume 5710 of *LNCS*, pages 384–398. Springer, 2009.

23 M. Hague and C.-H. L. Ong. Analysing mu-calculus properties of pushdown systems. In *SPIN*, volume 6349 of *LNCS*, pages 187–192. Springer, 2010.

24 M. Heizmann, J. Hoenicke, and A. Podelski. Nested interpolants. In *POPL*, pages 471–482. ACM, 2010.

25 L. Holík, R. Meyer, and S. Muskalla. Summaries for context-free games. *CoRR*, abs/1603.07256, 2016. URL: `http://arxiv.org/abs/1603.07256`.

26 A. Kaiser, D. Kroening, and T. Wahl. Efficient coverability analysis by proof minimization. In *CONCUR*, pages 500–515. Springer, 2012.

27 J. Kloos, R. Majumdar, F. Niksic, and R. Piskac. Incremental, inductive coverability. In *CAV*, volume 9206 of *LNCS*, pages 158–173. Springer, 2013.

28 O. Kupferman, N. Piterman, and M. Y. Vardi. An automata-theoretic approach to infinite-state systems. In *Time for Verification: Essays in Memory of Amir Pnueli*, volume 6200 of *LNCS*, pages 202–259. Springer, 2010.

29 Z. Long, G. Calin, R. Majumdar, and R. Meyer. Language-theoretic abstraction refinement. In *FASE*, volume 7212 of *LNCS*, pages 362–376. Springer, 2012.

30 A. Muscholl, T. Schwentick, and L. Segoufin. Active context-free games. *Theory of Computing Systems*, 39(1):237–276, 2005.

31 N. Piterman and M. Y. Vardi. Global model-checking of infinite-state systems. In *CAV*, volume 3114 of *LNCS*, pages 387–400. Springer, 2004.

32 S. J. Ramsay, R. P. Neatherway, and C.-H. L. Ong. A type-directed abstraction refinement approach to higher-order model checking. In *POPL*, pages 61–72. ACM, 2014.

33 T. Reps, S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *POPL*, pages 49–61. ACM, 1995.

34 M. Schuster and T. Schwentick. Games for active XML revisited. In *ICDT*, volume 31 of *LIPIcs*, pages 60–75. Dagstuhl, 2015.

35 S. Schwoon. *Model-Checking Pushdown Systems*. PhD thesis, TU Munich, 2002.

36 O. Serre. Note on winning positions on pushdown games with omega-regular conditions. *IPL*, 85(6):285–291, 2003.

37 M. Sharir and A. Pnueli. Two approaches to interprocedural data flow analysis. Technical Report 2, New York University, 1978.

38 A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. In *ICALP*, volume 194 of *LNCS*, pages 217–237. Springer, 1985.

39 D. Suwimonteerabuth, S. Schwoon, and J. Esparza. Efficient algorithms for alternating pushdown systems with an application to the computation of certificate chains. In *ATVA*, volume 4218 of *LNCS*, pages 141–153. Springer, 2006.

40 D. Tabakov and M. Y. Vardi. Experimental evaluation of classical automata constructions. In *LPAR*, volume 3835 of *LNCS*, pages 396–411. Springer, 2005.

41 WALi. Visited: 2016-16-07. URL: `https://research.cs.wisc.edu/wpis/wpds/download.php`.

42 I. Walukiewicz. Pushdown processes: Games and model-checking. *IC*, 164(2):234–263, 2001.

43 M. Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *CAV*, volume 4144 of *LNCS*. Springer, 2006.

44 M. De Wulf, L. Doyen, and J.-F. Raskin. A lattice theory for solving games of imperfect information. In *HSCC*, volume 3927 of *LNCS*, pages 153–168. Springer, 2006.