

# Summaries for Context-Free Games

Lukáš Holík<sup>1</sup>, Roland Meyer<sup>2</sup>, and Sebastian Muskalla<sup>2</sup>

<sup>1</sup> Brno University of Technology, holik@fit.vutbr.cz

<sup>2</sup> TU Kaiserslautern, {meyer, 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.

## 1 Introduction

The motivation of our work is to generalize the language-theoretic approach to verification of recursive programs [27, 31] 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* [40, 35] and *saturation* [12, 21]. Procedure summaries compute the effect of a procedure in 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 [37, 2, 9, 10]. 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 [9, 10]. The reason, besides simpler implementability, may be that the stack present in pre\* increases the search space.

Saturation has been lifted to games and synthesis in [16, 25], from which closest to our setting is the work of Cachat [16], 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 [32], where a problem similar to ours is solved by a reduction to [16]. In this paper, 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	[12, 21]	[40, 35]
Synthesis	[16, 32, 25]	



© Lukáš Holík, Roland Meyer, and Sebastian Muskalla;  
licensed under Creative Commons License CC-BY

Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

**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 [15]. 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  $\alpha$  and  $\beta$ . Indeed, since we consider left-derivations, each play starting in  $\alpha\beta$  will have a prefix that coincides with a maximal play starting in  $\alpha$ , followed by a suffix that essentially is a play from  $\beta$ . 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 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 [32]. 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  $A$  [32].

Our domain is compatible with algorithmic techniques that have proven efficient in a number of applications (see Section 9). We show how to adapt two heuristics to our fixed-point computation over formulas over boxes, namely **antichains** from [22, 5, 6] and **lazy evaluation** inspired by [20]. 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 [42] 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**.

**Acknowledgements.** We thank Olivier Serre, Matthew Hague, Georg Zetsche, and Emanuele D’Osualdo for helpful discussions.

## 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  $X \rightarrow \eta$  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  $\alpha$  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  $X \rightarrow \eta \in P$ . 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*  $N = N_\circ \cup N_\square$  of the set of non-terminals. We say that the non-terminals in  $N_\star$  are owned by player  $\star \in \{\circ, \square\}$ . The ownership partitioning is lifted to the sentential forms ( $\vartheta = \vartheta_\circ \cup \vartheta_\square$ ) as follows:  $\alpha \in \vartheta_\square$  if the leftmost non-terminal in  $\alpha$  is owned by  $\square$ , and  $\vartheta_\circ = \vartheta \setminus \vartheta_\square$ . In particular,  $\circ$  owns all terminal words. Combined with the left-derivation relation, this yields a game arena.

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

A *play*  $p = p_0 p_1 \dots$  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  $A = (T, Q, q_0, Q_F, \rightarrow)$ . 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.

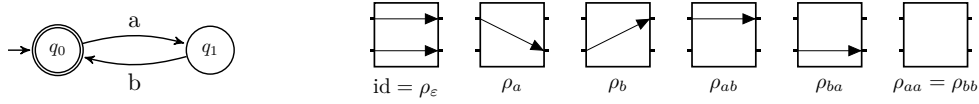
From now on, we use  $A = (T, Q, q_0, Q_F, \rightarrow)$  for finite automata and  $G = (N_\circ \cup N_\square, T, P)$  for grammars with ownership. Note that both use the terminal symbols  $T$ .

► **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  $\circ$  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. what about giving infinite plays as winning for refuter?

A *strategy* for player  $\star \in \{\circ, \square\}$  is a function that takes a non-maximal play  $p$  with  $p_{last} \in \vartheta_\star$  (it is  $\star$ 's turn) and returns a successor of this last position. A play is *conform* to a strategy if whenever it is the turn of  $\star$ , her next move coincides with the position returned

inclusion game:  
whatever terminal word  
is derived, it must belong  
to the given regular  
language



■ **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  $q_1$ .

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.

#### 3.1 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  $M_A = (\mathcal{P}(Q \times Q), ;, \text{id})$ . We refer to the elements  $\rho, \tau \in M_A$  as *boxes*. Since boxes are relations over the states of  $A$ , their relational composition is defined as usual,  $\rho; \tau = \{(q, q'') \mid \exists q' \in Q : (q, q') \in \rho \text{ and } (q', q'') \in \tau\}$ . Relational composition is associative. The identity box  $\text{id} = \{(q, q) \mid q \in Q\}$  is the neutral element wrt. relational composition.

A box  $\rho$  represents the set of words  $\mathcal{L}(\rho)$  with  $w \in \mathcal{L}(\rho)$ , if  $q \xrightarrow{w} q'$  iff  $(q, q') \in \rho$  for all pairs of states. 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  $\rho$ . The function  $\rho_- : T^* \rightarrow M_A$  maps  $w$  to the unique box  $\rho_w$  representing the word,  $w \in \mathcal{L}(\rho_w)$ . More explicitly,  $\rho_\varepsilon = \text{id}$ ,  $\rho_a = \{(q, q') \mid q \xrightarrow{a} q'\}$  for all  $a \in T$ , and  $\rho_{uv} = \rho_u; \rho_v$ . The image  $\rho_{T^*}$  contains exactly the boxes  $\rho$  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  $\text{BF}_A$  of *negation-free Boolean formulas over the transition monoid* (propositions are boxes). This set includes the unsatisfiable formula *false*. We use the rules  $\text{false} \wedge F = F \wedge \text{false} = \text{false}$  and  $\text{false} \vee F = F \vee \text{false} = F$  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  $\text{BF}_A$ .

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  $\nu : M_A \rightarrow \{\text{true}, \text{false}\}$  such that  $\nu(\rho) = \text{true}$  if and only if  $\rho$  does not contain a pair  $(q_0, q_f)$  with  $q_f \in Q_F$ . good boxes are those inducing rejecting words

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  $\text{BF}_A$  is not antisymmetric. To factor out the symmetries, we reason modulo logical equivalence,  $\text{BF}/\Leftrightarrow$ . Every formula is understood as a representative of the class of logically equivalent formulas of  $\text{BF}_A$ . Extending  $\Rightarrow$  to  $\text{BF}/\Leftrightarrow$  by comparing representatives then yields a partial order. The least element of the partial order is the equivalence class of *false*.

## 3.2 Operations

this is just a compact representation of sets of boxes

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  $\alpha$  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  $\beta$ , except that all sentential forms have a prefix  $w$ .

Let  $F$  and  $G$  represent all plays starting in  $\alpha$  and  $\beta$ , respectively. In  $F$ , terminal words like  $w$  are represented by boxes  $\rho$ . We append the plays from  $\beta$  by replacing  $\rho$  with  $\rho;G$ . To take into account all plays from  $\alpha$ , we do this replacement for all boxes in  $F$ . It remains to add the prefix  $w$  to the sentential forms in the plays from  $\beta$ . We let  $\rho;G$  replace every box  $\tau$  in  $G$  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 equation replaces  $\rho_a$  and  $\rho_b$  by  $\rho_a;G$  and  $\rho_b;G$ , respectively. The second equation prefixes  $\rho_c$  and  $\rho_d$  in  $G$  by the corresponding box  $\rho_a$  or  $\rho_b$ . ◀

► **Definition 5.** *Relational composition* over  $\text{BF}_A$  is defined by  $F;\text{false} = \text{false};G = \text{false}$  and for composite formulas  $(\star \in \{\wedge, \vee\}, \rho \in M_A)$  by

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

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  $F;G \Rightarrow F';G'$ .

**Proof.** The proof proceeds in phases (1) to (4) so that the claim in each phase is proven under the assumption of the claim proven in the previous phase. Let  $\{\star, \bar{\star}\} = \{\wedge, \vee\}$ . In the following, we will use  $\star$  and  $\bar{\star}$  as syntactic parts of formulas as well as to connect statements in the proof.

- (1) First, we prove the lemma for the case when  $F, F', G' \in M_A$  by induction on the structure of  $G$ . In the base case, all formulas are boxes, hence  $F = F'$  and  $G = G'$ , and the lemma holds trivially. For the induction step, let  $G = G_1 \star G_2$ . Note that the Boolean formulas

- $(a \star b) \Rightarrow c$  and  $(a \Rightarrow c) \bar{\star} (b \Rightarrow c)$  are equivalent, called Equivalence (i) in the following. By (i), we get  $(G_1 \Rightarrow G') \bar{\star} (G_2 \Rightarrow G')$ . Hence, by the induction hypothesis applied twice and by the monotonicity of  $\bar{\star}$ ,  $(F; G_1 \Rightarrow F'; G') \bar{\star} (F; G_2 \Rightarrow F'; G')$ . Again by Equivalence (i), we get  $(F; G_1 \star F; G_2) \Rightarrow F'; G'$ . This is  $F; G \Rightarrow F'; G'$  by the definition of relational composition since  $F$  is a box.
- (2) Next, we assume that  $F', G' \in M_A$  and  $F$  and  $G$  are arbitrary formulas. We prove the statement by induction on  $F$ . In the base case, all formulas except  $G$  are boxes, hence (1) proves the statement. For the induction step, let  $F = F_1 \star F_2$ . By Equivalence (i), we get  $(F_1 \Rightarrow F') \bar{\star} (F_2 \Rightarrow F')$ . Therefore, by the induction hypothesis and the monotonicity of  $\bar{\star}$ ,  $(F_1; G \Rightarrow F'; G') \bar{\star} (F_2; G \Rightarrow F'; G')$ . This is by (i) equivalent to  $(F_1; G \star F_2; G) \Rightarrow F'; G'$ . This shows  $F; G \Rightarrow F'; G'$  by the definition of relational composition.
- (3) Next, we assume only that  $F' \in M_A$  and prove the statement using induction on the structure of  $G'$ . In the base case, all formulas except  $F$  and  $G$  are boxes, hence the statement is proven by (2). Let  $G' = G'_1 \star G'_2$ . By the general equivalence of the Boolean formulas  $a \Rightarrow (b \star c)$  and  $(a \Rightarrow b) \star (a \Rightarrow c)$ , called Equivalence (ii) in the following, we get  $(G \Rightarrow G'_1) \star (G \Rightarrow G'_2)$ . Therefore, by the induction hypothesis and the monotonicity of  $\star$ ,  $(F; G \Rightarrow F'; G'_1) \star (F; G \Rightarrow F'; G'_2)$  holds. Again by (ii), we get  $F; G \Rightarrow (F'; G'_1 \star F'; G'_2)$ . This is  $F; G \Rightarrow F'; G'$  by the definition of relational composition since  $F'$  is a box.
- (4) Finally, we show the general case by induction to the structure of  $F'$ . In the base case,  $F'$  is a box boxes, hence (3) proves the statement. Let  $F' = F'_1 \star F'_2$ . By Equivalence (ii), we get  $(F \Rightarrow F'_1) \star (F \Rightarrow F'_2)$ . Therefore, by the induction hypothesis, and the monotonicity of  $\star$ ,  $(F; G \Rightarrow F'_1; G') \star (F; G \Rightarrow F'_2; G')$  holds. Again by Equivalence (ii), we get  $F; G \Rightarrow (F'_1; G' \star F'_2; G')$ , which is  $F; G \Rightarrow F'; G'$  by the definition of relational composition.  $\blacktriangleleft$

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 monotonic on  $\text{BF}/\Leftrightarrow$ . From now on, we can thus identify formulas with the classes they represent.

### 3.3 System of Equations

essentially, nonterminals are abstracted as set of boxes (instead of set of words).  
and branching is interpreted conjunctively/disjunctively depending on ownership thereof.

We introduce one variable  $\Delta_X$  for each non-terminal  $X \in N$ . Terminals  $a \in T$  yield boxes, and we write  $\Delta_a$  for  $\rho_a$ . We lift the notation  $\Delta_-$  to sentential forms:  $\Delta_\epsilon = \text{id}$  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  $\text{BF}/\Leftrightarrow$ ) induced by  $G$  and  $A$*  has one equation for each non-terminal  $X \in N_\star$  with  $\star \in \{\circ, \square\}$ . If  $X \rightarrow \eta_1, \dots, X \rightarrow \eta_k$  are all rules with  $X$  as their left-hand side, the equation is

$$\begin{aligned} \Delta_X &= \Delta_{\eta_1} \wedge \dots \wedge \Delta_{\eta_k}, & \text{if } X \in N_\square, \\ \Delta_X &= \Delta_{\eta_1} \vee \dots \vee \Delta_{\eta_k}, & \text{if } X \in N_\circ. \end{aligned}$$

With Lemma 6, for each non-terminal  $X$  we can understand the right-hand side of the associated equation as a monotonic function  $f_X : (\text{BF}/\Leftrightarrow)^N \rightarrow \text{BF}/\Leftrightarrow$ . It takes as input a vector of formulas (one for each non-terminal) and computes a new formula for  $\Delta_X$ . We



combine the functions for each non-terminal to a single function  $f : (\text{BF}/\Leftrightarrow)^N \rightarrow (\text{BF}/\Leftrightarrow)^N$ . It is monotonic on the product domain wrt. the product order  $\Rightarrow^N$ .

Since  $\text{BF}/\Leftrightarrow$  with  $\Rightarrow$  is a finite bottomed partial order, there is a unique least solution  $\sigma$  for the equation  $\Delta = f(\Delta)$ , namely  $\sigma = \bigsqcup_{i \in \mathbb{N}} f^i(\perp)$  [18]. 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

$$\perp \Rightarrow^N f(\perp) \Rightarrow^N f(f(\perp)) \Rightarrow^N \dots$$

stabilizes on a finite domain.

The solution  $\sigma : N \rightarrow \text{BF}/\Leftrightarrow$  yields a value  $\sigma_X$  for each non-terminal  $X \in N$ . We lift the notation to sentential forms by  $\sigma_\varepsilon = \text{id}$ ,  $\sigma_a = \rho_a$  for all  $a \in T$ , and  $\sigma_{\alpha\beta} = \sigma_\alpha; \sigma_\beta$ . From now on,  $\sigma$  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  $\Delta_X = \Delta_a; \Delta_Y \vee \Delta_\varepsilon = \rho_a; \Delta_Y \vee \text{id}$  and  $\Delta_Y = \Delta_b; \Delta_X = \rho_b; \Delta_X$ . Its least solution is  $\sigma_X = \text{id} \vee \rho_{ab}$  and  $\sigma_Y = \rho_b$ . To terminate the iteration, use  $\rho_{bab} = \rho_b$ . ◀

## 4 Semantics

Our goal is to determine whether refuter has a winning strategy for the non-inclusion game played from a given sentential form  $\alpha$ . The plays starting in  $\alpha$  form a (typically infinite) tree. The given sentential form is the root and each node has one successor for each sentential form that can be obtained by a left-derivation step. This means the leaves are precisely the terminal words derivable in plays from  $\alpha$ . Recall that it is refuters goal to disprove the inclusion in  $\mathcal{L}(A)$ . We call a leaf *rejecting* if it corresponds to a word outside this language. An inner node is rejecting if it is either owned by refuter and it has a rejecting successor, or it is owned by prover and all successors are rejecting. Refuter then has a winning strategy for non-inclusion when playing from  $\alpha$  if and only if the root is rejecting.

We understand the tree as an infinite negation-free Boolean formula. The terminal words at the leaves are the atomic propositions. Each inner node corresponds to an operation of conjunction or disjunction, depending on who is the owner of the sentential form. Deciding whether the root is rejecting then amounts to computing the truth value of this infinite formula under the assignment that sets to true precisely the words outside  $\mathcal{L}(A)$ .

Our goal is to compare this infinite formula with the finite formula obtained as the least solution of the fixed-point iteration presented in Section 3.

### 4.1 Emptiness Games

As a preparation, we note that the fixed-point solution to the inclusion game in particular solves the so-called *emptiness game*, where a maximal play is winning if it is infinite. The emptiness game can be understood as the inclusion game wrt.  $\mathcal{L}(A) = \emptyset$ . A winning strategy of prover for the emptiness game is thus a strategy that only generates infinite plays. We show that such a strategy exists when playing from  $\alpha$  if and only if  $\sigma_\alpha = \text{false}$ .

To prove the equivalence, we will also show that if  $\sigma_\alpha \neq \text{false}$  refuter has a strategy to enforce finite plays. To define this strategy, we use the following notation. Let  $\sigma^i$  be the  $i^{\text{th}}$  Kleene approximant of the least solution to the system of equations, so  $\sigma_X^i$  is the  $i^{\text{th}}$

approximation to the value of the non-terminal  $X$ . We define  $\sigma_a^i = \rho_a$  for  $a \in T \cup \{\varepsilon\}$  and all  $i \in \mathbb{N}$ . Just as we did for the fixed-point solution, we inductively define  $\sigma_{\alpha, \beta}^i = \sigma_\alpha^i; \sigma_\beta^i$ .

Note that we deal with negation free-formulas and evaluate conjunctions and disjunctions involving *false* on the syntactic level. As a consequence, the result of a conjunction is *false* if and only if at least one of the conjuncts was *false* and the result of a disjunction is *false* if and only if both disjuncts were *false*.

► **Theorem 9.** *Prover has a winning strategy for the emptiness game iff  $\sigma_\alpha = \text{false}$ .*

**Proof.** We have  $\sigma_X^0 = \text{false}$  for every non-terminal  $X$ . If  $i_0$  is a number such that  $\sigma_X^{i_0} \neq \text{false}$ , then  $\sigma_X^{i_0+k} \neq \text{false}$  for all  $k \in \mathbb{N}$ . This follows from monotonicity of the solution to the system of equations,  $\sigma_X^i \Rightarrow \sigma_X^{i+k}$  for all  $i, k \in \mathbb{N}$ , and from the fact that  $F \Rightarrow \text{false}$  implies  $F \Leftrightarrow \text{false}$ . Note that for a terminal word, we have  $\sigma_w^i \neq \text{false}$  for all  $i \in \mathbb{N}$ . Therefore,  $\sigma_\beta \neq \text{false}$  holds if and only if  $\sigma_X \neq \text{false}$  for all non-terminals  $X$  occurring in  $\beta$  by the definition of relational composition.

Assume  $\sigma_\alpha = \text{false}$ . We define a strategy  $s_\infty$  for prover such that all plays conform to it satisfy  $\sigma_\beta = \text{false}$  for all positions  $\beta$  occurring in the play. Since we have  $\sigma_w \neq \text{false}$  for terminal words, this means the plays have to be infinite.

Assume  $wX\beta$  is given and it is prover's turn. If there is  $Y$  in  $\beta$  so that  $\sigma_Y = \text{false}$ , we can pick any rule  $X \rightarrow \eta$  and the result will still satisfy  $\sigma_{w\eta\beta} = \text{false}$ . If there is no such non-terminal, we know that  $\sigma_X = \text{false}$  has to hold. The fixed-point solution to the system of equations satisfies  $\sigma_X = \bigwedge_{X \rightarrow \eta} \sigma_\eta$ . Since the conjunction is *false*, there is at least one rule  $X \rightarrow \eta$  with  $\sigma_\eta = \text{false}$ . If we pick this rule, the resulting position will satisfy  $\sigma_{w\eta\beta} = \text{false}$ .

Assume  $wX\beta$  is given and it is refuter's turn. If there is a non-terminal  $Y$  in  $\beta$  with  $\sigma_Y = \text{false}$ , the position resulting from any rule will still satisfy  $\sigma_{w\eta\beta} = \text{false}$ . If there is no such position, we know that  $\sigma_X = \text{false}$ . The fixed-point solution to the system of equations satisfies  $\sigma_X = \bigvee_{X \rightarrow \eta} \sigma_\eta$ . Since the disjunction is *false*, for any rule  $X \rightarrow \eta$  we get  $\sigma_\eta = \text{false}$ . Therefore,  $\sigma_{w\eta\beta} = \text{false}$  has to hold, no matter which rule  $X \rightarrow \eta$  refuter chooses.

For the other direction, we show that whenever  $\sigma_\alpha \neq \text{false}$  refuter has a strategy such that all plays conform to it are finite (a winning strategy for the non-emptiness game). In this case, prover cannot have a strategy that only generates infinite plays.

We define the *i-step attractor*  $\text{Attr}_i$  to be the set of non-terminals  $X$  such that  $\sigma_X^i \neq \text{false}$ . Note that the *i-step attractors* form a chain  $\text{Attr}_0 \subseteq \text{Attr}_1 \subseteq \dots$ . The chain has to stabilize because the set of non-terminals is finite. Given a non-terminal  $X$ , we define its *attractor level* to be the lowest index  $i$  such that  $X \in \text{Attr}_i$ , and  $\infty$  if no such index exists. We define the attractor level to be 0 for terminal symbols  $a \in T$ . The *attractor level* of a sentential form  $\beta$  is defined to be

$$\text{level}(\beta) = \sum_{j=1, \dots, |\beta|} |G|^{\text{attractor level of } \beta_j}.$$

We give a strategy for refuter such that for each play conform to it, the levels of the occurring positions form a strictly decreasing chain. Since such a chain has to be finite, this proves the claim.

Consider  $wX\beta$  with  $\sigma_{wX\beta} \neq \text{false}$ . In particular,  $\sigma_X \neq \text{false}$  and  $\sigma_Y \neq \text{false}$  for all non-terminals  $Y$  in  $\beta$ . This means  $\text{level}(wX\beta) \neq \infty$ . Let  $i$  be the attractor level of  $X$ .

Assume it is prover's turn. We have  $\sigma_X^i = \bigwedge_{X \rightarrow \eta} \sigma_\eta^{i-1}$ . Since  $X$  is in the *i-step attractor*, we get  $\sigma_X^i \neq \text{false}$  and thus  $\sigma_\eta^{i-1} \neq \text{false}$  for all rules  $X \rightarrow \eta$ . Hence, any symbol occurring in the right-hand side  $\eta$  of a rule for  $X$  has attractor level at most  $i-1$ . Applying such a rule replaces  $X$ , which contributes  $|G|^i$  to the level of  $wX\beta$ , by  $\eta$ , a sequence with

$$\text{level}(\eta) \leq |\eta||G|^{i-1} < |G||G|^{i-1} = |G|^i.$$



We conclude  $level(wX\beta) > level(w\eta\beta)$  for any rule  $X \rightarrow \eta$ .

Assume it is refuter's turn. We have  $\sigma_X^i = \bigvee_{X \rightarrow \eta} \sigma_\eta^{i-1}$ . Since  $\sigma_X^i \neq false$ , there is a rule  $X \rightarrow \eta$  with  $\sigma_\eta^{i-1} \neq false$ . In particular, all symbols in  $\eta$  have attractor level at most  $i - 1$ . If we pick the rule  $X \rightarrow \eta$ , we get  $level(wX\beta) > level(w\eta\beta)$ . ◀

## 4.2 The Infinite Tree of Plays vs. the Fixed-Point Solution

We discuss how the fixed-point solution to the system of equations relates to the (typically infinite) formula representing the tree of all plays from a given position. Let  $\mathcal{T}_\alpha$  be the tree of plays from  $\alpha$ . We argued that we can understand  $\mathcal{T}_\alpha$  as an infinite negation-free Boolean formula, where inner nodes are disjunctions or conjunctions (logical connectives) and leaves are boxes (atomic propositions). We call  $\mathcal{T}_\alpha$  a *formula tree* to emphasize the fact that we can see it as both, a formula and a tree. Note that the outdegree of inner nodes in  $\mathcal{T}_\alpha$  is bounded by the maximal number of rules for each non-terminal. We identify the empty formula tree with *false*. To generalize the notions of assignment and value from finite formulas to infinite formula trees, we need a least fixed point that propagates the values from the leaves up the tree. The following constructions are conservative extensions of the finite case and behave as expected when applied to finite formulas.

Given a formula tree  $\mathcal{T}$ , an *evaluation*  $e$  is a map from the nodes to  $\{true, false\}$ . We understand an evaluation as the set of nodes with value *true*. This helps us see that the set of evaluations on a fixed tree  $\mathcal{T}$  ordered by inclusion forms a complete lattice: The least element is the empty set, the join is the union, and the meet is the intersection.

Given  $e$ , we define the *1-step propagation*  $p(e)$  to be the evaluation obtained from  $e$  by adding (1) all disjunctions  $n$  so that some successor of  $n$  is in  $e$  and (2) all conjunctions  $n$  so that all successors of  $n$  are in  $e$ . Note that  $e \subseteq p(e)$ , so  $p$  is monotone over tree evaluations.

An assignment of boxes  $\nu : M_A \rightarrow \{true, false\}$  induces an evaluation  $e_\nu$  on a formula tree over  $M_A$ . The leaves  $\rho$  evaluate to  $\nu(\rho)$ , all other nodes evaluate to *false*. We define the *propagation* of  $\nu$  on  $\mathcal{T}$  to be the join  $\bigsqcup_{i \in \mathbb{N}} p^i(e_\nu)$ . The *value* of  $\mathcal{T}$  under  $\nu$  is the value of the root node in the propagation of  $\nu$  on  $\mathcal{T}$ . We define implication as usual:  $\mathcal{T} \Rightarrow \mathcal{T}'$  if under all  $\nu$  the value of  $\mathcal{T}'$  is at least the value of  $\mathcal{T}$ .

► **Lemma 10.** *The value of  $\mathcal{T}$  under  $\nu$  is true iff the root of  $\mathcal{T}$  is in  $p^{i_0}(e_\nu)$ , for some  $i_0 \in \mathbb{N}$ .*

**Proof.** If  $i_0$  exists, the claim follows from  $p^{i_0}(e_\nu) \subseteq \bigsqcup_{i \in \mathbb{N}} p^i(e_\nu)$ . Assume for all  $i$ , the root is not in  $p^i(e_\nu)$ . Since the join is the union, the root node will not be in the propagation. ◀

Given a formula tree  $\mathcal{T}$ , a subset of its nodes  $S$  induces the *prefix*  $\mathcal{T}^S$  defined by removing from  $\mathcal{T}$  the subtrees with root in  $S$ . We think of a removed subtree as being evaluated to *false*. We simplify the formula by propagating *false* upwards from the removed subtrees. Formally,  $\mathcal{T}^S$  is created by (1) marking all subtrees with root in  $S$ , (2) repeating until fixed point: marking all disjunctions that have all successors marked and marking all conjunctions that have some successor marked, including their subtrees, (3) deleting all marked nodes.  $\mathcal{T}^S$  is again a formula tree, each disjunction and conjunction has at least one successor.

Since we deal with negation-free formulas, the removed subtrees will never lead to the value of a tree being *true* that would have been *false* prior to the removal. Hence, a larger prefix will be easier to satisfy than a smaller one (created by removing more nodes).

► **Lemma 11.** *Let  $\mathcal{T}$  be a formula tree and let  $S \subseteq R$  be subsets of its nodes. Then  $\mathcal{T}^R \Rightarrow \mathcal{T}^S$ .*

**Proof.** The nodes of  $\mathcal{T}^R$  form a subset of the nodes of  $\mathcal{T}^S$ . Hence, for all assignments  $\nu$  we have  $e_\nu(\mathcal{T}^R) \subseteq e_\nu(\mathcal{T}^S)$  and  $p^i(e_\nu(\mathcal{T}^R)) \subseteq p^i(e_\nu(\mathcal{T}^S))$  for all  $i$ . In the limit, this yields

$\bigsqcup_{i \in \mathbb{N}} p^i(e_\varphi(\mathcal{T}^R)) \subseteq \bigsqcup_{i \in \mathbb{N}} p^i(e_\varphi(\mathcal{T}^S))$ . In particular, the value of the root node of  $\mathcal{T}^S$  is at least the value of the root node of  $\mathcal{T}^R$ .  $\blacktriangleleft$

Let  $S_i$  be the set of nodes in  $\mathcal{T}$  of depth strictly greater than  $i$ . We write  $\mathcal{T}^i$  for  $\mathcal{T}^{S_i}$  and call this tree the *cut at level  $i$* . The previous lemma implies  $\mathcal{T}_\alpha^i \Rightarrow \mathcal{T}_\alpha^{i+k}$  for all  $i, k \in \mathbb{N}$ . The value of the whole tree  $\mathcal{T}$  is *true* if and only if there is a finite cut such that the value of the prefix is already *true*.

► **Lemma 12.** *The value of  $\mathcal{T}$  under  $\nu$  is true iff the value of  $\mathcal{T}^{i_0}$  is true, for some  $i_0 \in \mathbb{N}$ .*

**Proof.** By Lemma 10, it is sufficient to show that for all  $i_0 \in \mathbb{N}$ , the root of  $\mathcal{T}$  is in  $p^{i_0}(e_\nu)$  if and only if the value of  $\mathcal{T}^{i_0}$  is *true*.

Assume the root of  $\mathcal{T}$  is in  $p^{i_0}(e_\nu)$ . Since each step of propagation can only propagate the value of a node to its immediate predecessors, there is a set of leaves of height at most  $i_0$  assigned to *true* by  $\nu$  that are sufficient to cause the root to be in  $p^{i_0}(e_\nu)$ . Consider the evaluation  $e'$  that coincides with  $e_\nu$  on nodes up to height at most  $i_0$  and evaluates all other nodes to *false*. We obtain  $e' \subseteq e_\nu$ , and by the monotonicity of propagation  $p^{i_0}(e') \subseteq p^{i_0}(e_\nu)$ . Since *true* got propagated to the root by leaves of depth  $i_0$  and those leaves are still evaluated to *true* by  $e'$ , the root is contained in  $p^{i_0}(e')$ . To finish the proof, note that the semantics of removing nodes of depth greater than  $i_0$  is defined by propagating *false* from nodes of depth greater than  $i_0$ . This coincides with the behavior of  $p^{i_0}(e')$ .

Assume the value of  $\mathcal{T}^{i_0}$  is *true*, which means the root of  $\mathcal{T}^{i_0}$  is in  $\bigsqcup_{i \in \mathbb{N}} p^i(e_\nu(\mathcal{T}^{i_0}))$ . Since  $\mathcal{T}^{i_0}$  has height at most  $i_0$ , we know  $\bigsqcup_{i \in \mathbb{N}} p^i(e_\nu(\mathcal{T}^{i_0})) = p^{i_0}(e_\nu(\mathcal{T}^{i_0}))$ . By  $e_\nu(\mathcal{T}^{i_0}) \subseteq e_\nu(\mathcal{T})$  we get  $p^{i_0}(e_\nu(\mathcal{T}^{i_0})) \subseteq p^{i_0}(e_\nu(\mathcal{T}))$ , as desired.  $\blacktriangleleft$

The next lemma shows that the composition of two trees for sentential forms, each cut at some level, is a prefix of the tree for the concatenation of the sentential forms, cut at the sum of levels. The composition appends the second tree to every leaf of the first, as before.

► **Lemma 13.** *Let  $\alpha, \beta$  be sentential forms,  $i, j \in \mathbb{N}$ . The tree  $\mathcal{T}_\alpha^i; \mathcal{T}_\beta^j$  is a prefix of  $\mathcal{T}_{\alpha\beta}^{i+j}$ .*

**Proof.** By the definition of relational composition, a branch of  $\mathcal{T}_\alpha^i; \mathcal{T}_\beta^j$  can be decomposed into a branch of  $\mathcal{T}_\alpha^i$  of length at most  $i$  and a branch of  $\mathcal{T}_\beta^j$  of length at most  $j$ . The total length of the branch is at most  $i + j$ , which means it will also occur in  $\mathcal{T}_{\alpha\beta}^{i+j}$ , unless other nodes that were removed in  $\mathcal{T}_{\alpha\beta}^{i+j}$  trigger some node on the branch to be marked. Towards a contradiction, take the deepest node on the branch that gets marked triggered by a set of nodes of depth greater than  $i + j$  being marked. Since it got marked although it has one successor that will not be marked, namely the one on the branch, it has to be a conjunction. Each triggering node is either in the part of the tree corresponding to the derivation process of  $\alpha$  or in the part of the tree corresponding to the derivation process of  $\beta$ , i.e., there is a corresponding node in  $\mathcal{T}_\alpha$  or  $\mathcal{T}_\beta$ . In the first case, the corresponding node has depth  $i + j + 1 > i$ , so it will be removed in  $\mathcal{T}_\alpha^i$ . In the second case, it either occurs on a branch of  $\mathcal{T}_\beta$  that was appended to a branch of  $\mathcal{T}_\alpha$  of length greater than  $i$ , which will be removed in  $\mathcal{T}_\alpha^i$ , or it occurs on a branch of length greater than  $j$  in  $\mathcal{T}_\beta$ , which will be removed in  $\mathcal{T}_\beta^j$ . In any case, all triggering nodes will also be removed to obtain  $\mathcal{T}_\alpha^i; \mathcal{T}_\beta^j$ , which would lead to the removal of the node on the branch in  $\mathcal{T}_\alpha^i; \mathcal{T}_\beta^j$ . This is a contradiction to the fact that the branch appears there.

To obtain  $\mathcal{T}_\alpha^i; \mathcal{T}_\beta^j$  from  $\mathcal{T}_{\alpha\beta}^{i+j}$  by removing nodes, note that any branch of  $\mathcal{T}_{\alpha\beta}^{i+j}$  decomposes into a branch of  $\mathcal{T}_\alpha^{i+j}$  and a branch of  $\mathcal{T}_\beta^{i+j}$  so that their total length is at most  $i + j$ . If the length of the  $\alpha$ -part is longer than  $i$ , remove the first node of depth strictly greater than  $i$

(this will also remove the  $\beta$ -part potentially appended to it). If the length of the  $\beta$ -part is longer than  $j$ , remove the first node of relative depth strictly greater than  $j$ . ◀

Combined with Lemma 11, we obtain  $\mathcal{T}_\alpha^i, \mathcal{T}_\beta^j \Rightarrow \mathcal{T}_{\alpha\beta}^{i+j}$ .

We can now prove the fundamental correspondence between the  $i^{\text{th}}$  Kleene approximant and the formula tree cut at level  $i$ . We do not get a precise result like  $\sigma_\alpha^i \Leftrightarrow \mathcal{T}_\alpha^{f(i)}$ . This is due to the fact that exploring the tree for one more level and doing one step of Kleene iteration behave differently. Exploring the tree for one more level will consider one more derivation step in each branch. This derivation step will be applied only to the leftmost non-terminal of the sentential form forming the last node in a branch of length  $i$ . Doing one step of Kleene iteration will replace  $\sigma_X^i$  by  $\sigma_X^{i+1}$  for every non-terminal, which means applying one derivation to each non-terminal that occurs at level  $i$ . A sentential form obtained from  $\alpha$  and represented by the  $i^{\text{th}}$  Kleene approximation has at most  $|\alpha||G|^i$  symbols. Indeed, in each step, we replace a non-terminal by at most the number of symbols in the largest right-hand side of any rule in  $G$ . We can think the Kleene iteration as exploring the trees for all non-terminals simultaneously and composing them to get the formula for  $\alpha$ , instead of just exploring the tree for  $\alpha$  directly.

► **Lemma 14.**  $\mathcal{T}_\alpha^i \Rightarrow \sigma_\alpha^i \Rightarrow \mathcal{T}_\alpha^{|\alpha||G|^i}$ .

**Proof.** We prove the statement using induction on  $i$ .

Let  $i = 0$ . We consider two cases. If  $\alpha$  is a terminal word,  $\alpha = w \in T^*$ , then the tree of plays actually only consists of the root node labeled by  $\rho_w$ . Therefore, we have

$$\mathcal{T}_w^{|\alpha||G|^0} = \mathcal{T}_w^{|\alpha|} = \mathcal{T}_w^0 = \mathcal{T}_w = \rho_w \Leftrightarrow \sigma_w^0.$$

If  $\alpha$  contains a non-terminal, then  $\mathcal{T}_\alpha$  has height at least 1. This means the root node is a conjunction or disjunction, which will be deleted since all its successors of depth larger than 0 will be marked. The empty tree remains, which we identify with *false*. Furthermore,  $\sigma_\alpha^0 = \text{false}$ , since we initialize the solution with *false* for every non-terminal. As the implication  $\mathcal{T}_\alpha^i \Rightarrow \mathcal{T}_\alpha^{|\alpha||G|^i}$  always holds, we obtain  $\sigma_\alpha \Leftrightarrow \mathcal{T}_\alpha^0 \Rightarrow \mathcal{T}_\alpha^{|\alpha||G|^0}$ .

Let  $i > 0$ . We prove the statement by an inner induction on the structure of  $\alpha$ . Assume  $\alpha = X$  is a single non-terminal. (If  $\alpha = \varepsilon$  or  $\alpha = a \in T$ , the proof for  $i = 0$  carries over.) Assume  $X$  is owned by prover. By the induction hypothesis,

$$\mathcal{T}_\eta^{i-1} \Rightarrow \sigma_\eta^{i-1} \Rightarrow \mathcal{T}_\eta^{|\eta||G|^{i-1}}$$

for any rule  $X \rightarrow \eta$ . By the monotonicity of conjunction,

$$\bigwedge_{X \rightarrow \eta} \mathcal{T}_\eta^{i-1} \Rightarrow \bigwedge_{X \rightarrow \eta} \sigma_\eta^{i-1} \Rightarrow \bigwedge_{X \rightarrow \eta} \mathcal{T}_\eta^{|\eta||G|^{i-1}}.$$

We have  $\sigma_X^i = \bigwedge_{X \rightarrow \eta} \sigma_\eta^{i-1}$ , and since formula trees are created by applying grammar rules, we know that  $\bigwedge_{X \rightarrow \eta} \mathcal{T}_\eta^{i-1} = \mathcal{T}_X^i$ . Furthermore,

$$\bigwedge_{X \rightarrow \eta} \mathcal{T}_\eta^{|\eta||G|^{i-1}} = \mathcal{T}_X^{|\eta||G|^{i-1}+1} \Rightarrow \mathcal{T}_X^{|\alpha||G|^i}.$$

The implication is due to  $|\eta| \leq |G| - 1$  and Lemma 11. Altogether,  $\mathcal{T}_X^i \Rightarrow \sigma_X^i \Rightarrow \mathcal{T}_X^{|\alpha||G|^i}$ . If  $X$  is owned by refuter, the proof is similar (disjunction is also monotonic).

Let  $\alpha = x.\gamma$ . We may assume by the inner induction that  $\mathcal{T}_\gamma^i \Rightarrow \sigma_\gamma^i \Rightarrow \mathcal{T}_\gamma^{|\gamma||G|^i}$  holds. As in the base case, we know  $\mathcal{T}_x^i \Rightarrow \sigma_x^i \Rightarrow \mathcal{T}_x^{|\gamma||G|^i}$ . By an argumentation analogous to the one used in the proof Lemma 13,  $\mathcal{T}_{x\gamma}^i$  is a prefix of  $\mathcal{T}_x^i; \mathcal{T}_\gamma^i$ , so

$$\mathcal{T}_{x\gamma}^i \Rightarrow \mathcal{T}_x^i; \mathcal{T}_\gamma^i \Rightarrow \sigma_x^i; \sigma_\gamma^i = \sigma_{x\gamma}^i.$$

The first implication is Lemma 11, the second is by monotonicity of composition. The equality holds by definition.

Similarly,  $\sigma_x^i; \sigma_\gamma^i \Rightarrow \mathcal{T}_x^{|\gamma||G|^i}; \mathcal{T}_\gamma^{|\gamma||G|^i}$  holds by monotonicity, and  $\mathcal{T}_x^{|\gamma||G|^i}; \mathcal{T}_\gamma^{|\gamma||G|^i}$  is a prefix of  $\mathcal{T}_{x\gamma}^{(|\gamma|+1)|G|^i}$  by Lemma 13, so we conclude  $\sigma_{x\gamma}^i \Rightarrow \mathcal{T}_{x\gamma}^{(|\gamma|+1)|G|^i} = \mathcal{T}_{x\gamma}^{|\gamma||G|^i}$ . ◀

We now lift the correspondence between the  $i^{\text{th}}$  Kleene approximant and the tree cut at level  $i$  to a correspondence between the (usually infinite) formula tree and the fixed-point solution. The result is an exact characterization of the fixed-point solution.

► **Theorem 15.**  $\mathcal{T}_\alpha \Leftrightarrow \sigma_\alpha$ .

**Proof.** Assume the value of  $\mathcal{T}_\alpha$  under  $\nu$  is *true*. By Lemma 12, there is a finite index  $i_0$  so that the value of  $\mathcal{T}_\alpha^{i_0}$  is *true*. By Lemma 14 and the fact that the fixed-point solution is implied by any approximant,  $\mathcal{T}_\alpha^{i_0} \Rightarrow \sigma_\alpha^{i_0} \Rightarrow \sigma_\alpha$ , so  $\sigma_\alpha$  also evaluates to *true* under  $\nu$ .

Assume  $\sigma_\alpha$  evaluates to *true* under  $\nu$ . Note that  $\sigma_\alpha = \sigma_\alpha^{i_0}$  for some index  $i_0 \in \mathbb{N}$ . Again by Lemma 14, we have  $\sigma_\alpha^{i_0} \Rightarrow \mathcal{T}_\alpha^{|\alpha||G|^{i_0}}$ , so the value of  $\mathcal{T}_\alpha^{|\alpha||G|^{i_0}}$  is *true*. Since  $\mathcal{T}_\alpha^{|\alpha||G|^{i_0}}$  is a prefix of  $\mathcal{T}_\alpha$ , the value of  $\mathcal{T}_\alpha$  is *true* by Lemma 11. ◀

The theorem yields another method for computing the fixed-point solution by exploring the tree of plays up to a finite level. We could use the fact that  $\sigma = \sigma^{i_0}$ , where an upper bound for  $i_0$  can be computed using  $|G|$  and the number of equivalence classes of formulas over  $M_A$ . By Lemma 14, we obtain  $\sigma_\alpha = \sigma_\alpha^{i_0} \Leftrightarrow \mathcal{T}_\alpha^{|\alpha||G|^{i_0}}$ , so we may also explore  $\mathcal{T}_\alpha$  up to level  $|\alpha||G|^{i_0}$ .

Note that this will not only require exponentially more iterations to obtain the fixed point in the worst case, it is also impractical because  $\mathcal{T}_\alpha^i \Leftrightarrow \mathcal{T}_\alpha^{i+1}$  does not necessarily imply  $\mathcal{T}_\alpha^i \Leftrightarrow \mathcal{T}_\alpha$ . Unlike in the Kleene iteration, we cannot conclude to have reached the fixed point as soon as one unfolding does not change the solution. Consider the grammar given by  $S \rightarrow X, S \rightarrow a, X \rightarrow Y, X \rightarrow a, Y \rightarrow b$  where refuter owns all non-terminals. Let the automaton be so that  $\rho_a \neq \rho_b$ . Then we have  $\mathcal{T}_S^1 \Leftrightarrow \mathcal{T}_S^2 \Leftrightarrow \rho_a$ , but  $\mathcal{T}_S^2 \not\Leftrightarrow \mathcal{T}_S^3 \Leftrightarrow \mathcal{T}_S \Leftrightarrow \rho_a \vee \rho_b$ .

## 5 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 16.** *Inclusion games are determined:*

$$\vartheta = W^{\subseteq \mathcal{L}(A)} \cup W^{\not\subseteq \mathcal{L}(A)},$$

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  $\alpha$  is winning for a player: Compute the formula  $\sigma_\alpha$  and evaluate it under  $\nu$  to check whether it is rejecting.

It has been shown in [39] that for all games on pushdown systems with  $\omega$ -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  $\text{id}$  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  $\sigma_w$  and  $\sigma_\beta$  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.

## 5.1 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  $\text{CNF}_A = \mathcal{P}(\mathcal{P}(M_A))$ . Identify  $\text{true} = \{\}$  and  $\text{false} = \{\{\}\}$ . In this section, all formulas will stem from  $\text{CNF}_A$ .

Since our CNF-formulas are negation-free, implication has a simple characterization.

► **Lemma 17.**  $F \Rightarrow G$  if and only if there is  $j : G \rightarrow F$  so that  $j(H) \subseteq H$  for all  $H \in G$ .

**Proof.** The implication from right to left is immediate. Assume  $F \Rightarrow M$  but there is no map  $j$  as required. Then there is some clause  $H \in M$  so that for every clause  $C \in F$  we find a variable  $x_C \in C$  with  $x_C \notin H$ . Consider the assignment  $\nu(x_C) = \text{true}$  for all  $x_C$  and  $\nu(y) = \text{false}$  for the remaining variables. Then  $\nu(F) = \text{true}$ . At the same time,  $\nu(M) = \text{false}$  as  $\nu(H) = \text{false}$ . This contradicts the assumption  $F \Rightarrow M$ , which means  $\nu(F) = \text{true}$  implies  $\nu(M) = \text{true}$  for every assignment  $\nu$ . ◀

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

► **Lemma 18.**  $F \vee G \Leftrightarrow \{K \cup H \mid K \in F, H \in G\}$  and  $F \wedge G \Leftrightarrow F \cup G$ .

When computing 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 20 gives a closed-form representation of the result. To understand the idea, consider the composition of one clause with a CNF.

► **Example 19.** 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  $K; M$  for every clause  $K \in G$ . 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  $\rho$  with. This justifies the following set-theoretic characterization.

► **Lemma 20.**  $F; G \Leftrightarrow \bigcup_{K \in F} \bigcup_{z: K \rightarrow G} \{ \bigcup_{\rho \in K} \rho; z(\rho) \}$ .

For negation-free formulas, the presence of the empty clause characterizes *false*.

► **Lemma 21.**  $F \Leftrightarrow \text{false}$  if and only if  $\{\}$  is a clause of  $F$ .

**Proof.** Note that  $\text{false} \Rightarrow F$  holds for any formula. In the following, we use the characterization of implication given in Lemma 17. Assume that  $F$  contains the empty clause. We define  $j : \text{false} \rightarrow F$  by mapping the clause  $\{\}$  of  $\text{false} = \{\{\}\}$  to the empty clause in  $F$ . Assume that  $F \Leftrightarrow \text{false}$ . In particular, the empty clause in  $\text{false}$  embeds a clause of  $F$ . The embedded clause has to be the empty clause. ◀

## 5.2 Strategy for Prover

Prover wins on infinite plays, and therefore does not have to care about termination. This yields a simple positional winning strategy.

► **Theorem 22.** *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  $W^{\subseteq \mathcal{L}(A)}$ .*

**Proof.** For  $w \in T^*$ , we have  $\sigma_w = \{\{\rho_w\}\}$ . In particular,  $w \in \mathcal{L}(A)$  if and only if  $\sigma_w$  is not rejecting. This shows that  $\mathcal{L}(A) \subseteq W^{\subseteq \mathcal{L}(A)}$  and  $\overline{\mathcal{L}(A)} \cap W^{\subseteq \mathcal{L}(A)} = \emptyset$ .

We show that all positions occurring in a play conform to  $s^{\subseteq \mathcal{L}(A)}$  and starting in a position from  $W^{\subseteq \mathcal{L}(A)}$  remain in  $W^{\subseteq \mathcal{L}(A)}$ . This proves the claim since we either obtain an infinite play or we end up in a position in  $\mathcal{L}(A)$ . In both cases, the inclusion winning condition is satisfied. Technically, whenever prover owns the leftmost non-terminal, the strategy will choose a rule such that the new position still has a rejecting formula. By Lemma 23(1) below, this is possible. Whenever refuter owns the leftmost non-terminal, she can only choose a rule such that new position still has a non-rejecting formula by Lemma 23(2). ◀

Strategy  $s^{\subseteq \mathcal{L}(A)}$  is *uniform* and *positional*. Uniform means it is winning from all position in  $W^{\subseteq \mathcal{L}(A)}$ . The strategy is positional in that it only needs to consider the current position in order to make a choice. Moreover, we can precompute  $\sigma_\eta$  for all  $\eta$  occurring as the right-hand side of a rule. Together with the representation of sentential forms as triples  $(\sigma_w, X, \sigma_\beta)$ , this will be sufficient to decide on prover's next move. Hence, the strategy is not only positional (but depending on an infinite set of positions), it can even be *finitely represented*. One can also understand it as the winning strategy for a finite game where the triples  $(\{\{\rho_w\}\}, X, \sigma_\beta)$  form the nodes and the production rules induce the edges.

For the proof, we show that whenever we are in a non-rejecting position and it is prover's turn, there is a move to a non-rejecting position. Hence, if we start from  $W^{\subseteq \mathcal{L}(A)}$ , the condition on the existence of a move (stated in the theorem as *if possible*) does not apply. Refuter can only move to positions with non-rejecting formulas.

► **Lemma 23.** *Let  $\alpha = wX\beta \in \vartheta$  with  $\sigma_\alpha$  not rejecting.*

- (1) *If  $X \in N_\square$ , there is  $X \rightarrow \eta$  so that  $\sigma_{w\eta\beta}$  is not rejecting.*
- (2) *If  $X \in N_\circ$ , then  $\sigma_{w\eta\beta}$  is not rejecting for all  $X \rightarrow \eta$ .*

**Proof.** To prove Lemma 23, note that  $\sigma_{wX\beta} = \sigma_w; \sigma_X; \sigma_\beta$ . Since  $w$  is a terminal word, we have  $\sigma_w = \{\{\rho_w\}\}$ . Hence, a clause of  $\sigma_w; \sigma_X$  will be of type  $\rho_w; K$ , where  $K$  is a clause of  $\sigma_X$ .

- (1) If  $\sigma_\alpha$  is not rejecting, it contains a clause  $K'$  without a rejecting box. By Lemma 20,  $K'$  is defined by a clause  $K$  of  $\sigma_{wX}$  and a mapping  $z$  from the boxes of  $K$  to the clauses of  $\sigma_\beta$ . Since  $X$  is owned by prover, we have  $\sigma_X = \bigwedge_{X \rightarrow \eta} \sigma_\eta$ . The conjunction is a union of the sets of clauses, Lemma 18. Hence, clause  $K$  also occurs in  $\sigma_{w\eta}$  for some  $\eta$ . By choosing the same mapping  $z$ , we get that  $K'$  is also a clause in  $\sigma_{w\eta\beta}$ .



- (2) We prove (2). If  $\sigma_\alpha$  is not rejecting, it contains a clause  $K'$  without a rejecting box. By Lemma 20,  $K'$  is determined by a clause  $\rho_w; K$  of  $\sigma_{wX}$  and a map  $z : \rho_w; K \rightarrow \sigma_\beta$  mapping boxes to clauses. Since  $X$  is owned by refuter, we have  $\sigma_X = \bigvee_{X \rightarrow \eta} \sigma_\eta$ . By the characterization of disjunction (Lemma 18), there is a representation of  $\rho_w; K$  as  $\bigcup_{X \rightarrow \eta} \rho_w; K_\eta$  with  $K_\eta \in \sigma_\eta$  for all  $X \rightarrow \eta$ . Consider a rule  $X \rightarrow \eta$ . Let  $K''$  be the clause of  $\sigma_{w\eta\beta}$  determined by the clause  $\rho_w; K_\eta$  and the map  $z$  (nproperly restricted). This clause is not rejecting. If  $K''$  contained a rejecting box,  $K'$  would also contain this box since  $\rho_w; K_\eta \subseteq \rho_w; K$ . A contradiction  $\blacktriangleleft$

### 5.3 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  $F \in \text{CNF}_A$  is a function  $c : F \rightarrow M_A$  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 at least one 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* (Lemma 21), 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 proofs.

► **Lemma 24.** *Consider  $F \Rightarrow G$ . For any choice function  $c$  on  $F$ , there is a choice function  $c'$  on  $G$  that refines it.*

**Proof of Lemma 24.** By Lemma 17, any clause  $H$  of  $G$  embeds a clause  $j(H)$  of  $F$ . We can define  $c'(H)$  as  $c(j(H))$  to get a choice function with  $c'(G) \subseteq c(F)$ .  $\blacktriangleleft$

To construct the strategy, we consider formulas obtained from Kleene approximants. Define a *sequence of levels*  $lvl$  associated to a sentential form  $\alpha$  to be a sequence of natural numbers of the same length. The formula  $\sigma_\alpha^{lvl}$  corresponding to  $\alpha$  and  $lvl$  is defined by  $\sigma_a^i = \{\{\rho_a\}\}$  for all  $a \in T \cup \{\varepsilon\}$ ,  $\sigma_X^i$  the solution to  $X$  from the  $i^{\text{th}}$  Kleene iteration, and  $\sigma_{\alpha,\beta}^{lvl,lvl'} = \sigma_\alpha^{lvl}; \sigma_\beta^{lvl'}$ . A choice function for  $\alpha$  and  $lvl$  is a choice function on  $\sigma_\alpha^{lvl}$ . Note that  $\sigma_a^i$  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  $\sigma_\alpha$  can be understood as a choice function on  $\sigma_\alpha^{i_0}$ . Here, we use a single number  $i_0$  to represent a sequence  $lvl = i_0 \dots i_0$  of the appropriate length.

By definition,  $\sigma_X^0$  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 25.** *If there is a choice function for  $\alpha$  and  $lvl$ , then  $lvl$  does not assign zero to any non-terminal  $X$  in  $\alpha$ .*

The lemma has an important consequence. Consider a sentential form  $\alpha$  with an associated sequence  $lvl \in 0^*$  and a choice function  $c$  for  $\alpha$  and  $lvl$ . Then  $\alpha$  has to be a terminal word,  $\alpha = w \in T^*$ ,  $\sigma_\alpha^{lvl} = \{\{\rho_w\}\}$ , and the choice function has to select  $\rho_w$ . 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  $\alpha$  and let  $i = lvl_j$  be the corresponding entry of  $lvl$ . We split  $lvl = lvl'.i.lvl''$  into the prefix for  $w$ , the entry  $i$  for  $X$ , and the suffix for  $\beta$ . For each rule  $X \rightarrow \eta$ , we define  $lvl_\eta = lvl'.(i-1) \dots (i-1).lvl''$  to be the sequence associated to  $w\eta\beta$ . It coincides with  $lvl$  on  $w$  and  $\beta$  and has entry  $i-1$  for all symbols in  $\eta$ . Note that for a terminal word, the formula is independent of the associated level, so we have  $\sigma_{wX}^{lvl'.i} = \sigma_{wX}^i$  and  $\sigma_{w\eta}^{lvl'.(i-1) \dots (i-1)} = \sigma_{w\eta}^{i-1}$ .

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

► **Lemma 26.** *Let  $c$  be a choice function for  $\alpha = wX\beta$  and  $lvl$ .*

- (1) *If  $X \in N_\square$ , for all  $X \rightarrow \eta$  there is a choice function  $c_\eta$  for  $w\eta\beta$  and  $lvl_\eta$  that refines  $c$ .*
- (2) *If  $X \in N_\circ$ , there is  $X \rightarrow \eta$  and a choice function  $c_\eta$  for  $w\eta\beta$  and  $lvl_\eta$  that refines  $c$ .*

**Proof.**

- (1) Note that any element of  $\mathbb{N}^*$  containing a non-zero entry is certainly not minimal, since we can obtain a smaller element by replacing any non-zero entry by  $\varepsilon$ . Any element of the form  $0^*$  is minimal, since there is no  $i$  as required by the definition of  $\prec$ .

Assume  $v_0 \succ v_1 \succ \dots$  is an infinite descending chain. Let  $b$  be the maximal entry of  $v_0$ , i.e.  $b = \max_{j=1, \dots, |v_0|} v_{0j}$ , and note that no  $v_l$  with  $l \in \mathbb{N}$  can contain an entry larger than  $b$  by the definition of  $\prec$ . Therefore, we may map each  $v_l$  to its Parikh image  $\psi(v_l) \in \mathbb{N}^{b+1}$ , the vector such that  $\psi(v_l)_j$  (for  $j \in \{0, \dots, b\}$ ) is the number of entries equal to  $j$  in  $v_l$ .

Now note that we have  $\psi(v_i) > \psi(v_{i+1})$  with respect to the lexicographic ordering on  $\mathbb{N}^{b+1}$ . Hence, the chain  $\psi(v_0) > \psi(v_1) > \dots$  is an infinite descending chain, which cannot exist since the lexicographic ordering is known to be well-founded.

- (2) We show that there is a rule  $X \rightarrow \eta$  and a choice function  $c_\eta$  on  $\sigma_{w\eta\beta}^{lvl_\eta}$  refining  $c$ . Towards a contradiction, assume this is not the case. Then for each rule  $X \rightarrow \eta$ , there is at least one clause  $K''_\eta$  of  $\sigma_{w\eta\beta}^{lvl_\eta}$  that does not contain a box in the image of  $c$ . By Lemma 20, this clause is defined by a clause  $\rho_w; K'_\eta$  of  $\sigma_{w\eta}^{i-1}$  and a function  $z_\eta$  mapping the boxes from this clause to  $\sigma_\beta^{lvl''}$ .

We have  $\sigma_X^i = \bigvee_{X \rightarrow \eta} \sigma_\eta^{i-1}$ . A clause of  $\sigma_{wX}^i$  is thus (Lemma 18) of the form

$$K = \rho_w; \left( \bigcup_{X \rightarrow \eta} K_\eta \right) = \bigcup_{X \rightarrow \eta} \rho_w; K_\eta,$$

where each  $K_\eta$  is a clause of  $\sigma_\eta^{i-1}$ . We construct the clause  $K' = \rho_w; (\bigcup_{X \rightarrow \eta} K'_\eta)$  of  $\sigma_{wX}^i$  using the  $K'_\eta$  from above. On this clause, we define the map  $z' = \bigcup_{X \rightarrow \eta} z_\eta$  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  $\eta$ , pick an arbitrary  $\eta$  among these.) By Lemma 20,  $K'$  and  $z'$  define a clause of  $\sigma_\alpha^{lvl}$ . The choice function  $c$  selects a box  $\rho_w; \rho; \tau$  out of this clause, where there is a rule  $X \rightarrow \eta$  such that  $\rho \in K'_\eta$  and  $\tau \in z'(\rho_w; \rho) = z_\eta(\rho_w; \rho)$ . This box is also contained in  $K''_\eta$ . A contradiction to the assumption that no box from  $K''_\eta$  is in the image of  $c$ . ◀

Notice that the sequence  $lvl_\eta$  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  $\varepsilon$ .

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

**Proof.** Note that any element of  $\mathbb{N}^*$  containing a non-zero entry is certainly not minimal, since we can obtain a smaller element by replacing any non-zero entry by  $\varepsilon$ . Any element of the form  $0^*$  is minimal, since there is no  $i$  as required by the definition of  $\prec$ .

Assume  $v_0 \succ v_1 \succ \dots$  is an infinite descending chain. Let  $b$  be the maximal entry of  $v_0$ , i.e.  $b = \max_{j=1, \dots, |v_0|} v_{0j}$ , and note that no  $v_l$  with  $l \in \mathbb{N}$  can contain an entry larger than  $b$  by the definition of  $\prec$ . Therefore, we may map each  $v_l$  to its Parikh image  $\psi(v_l) \in \mathbb{N}^{b+1}$ , the vector such that  $\psi(v_l)_j$  (for  $j \in \{0, \dots, b\}$ ) is the number of entries equal to  $j$  in  $v_l$ .

Now note that we have  $\psi(v_i) > \psi(v_{i+1})$  with respect to the lexicographic ordering on  $\mathbb{N}^{b+1}$ . Hence, the chain  $\psi(v_0) > \psi(v_1) > \dots$  is an infinite descending chain, which cannot exist since the lexicographic ordering is known to be well-founded. ◀

Lemma 27 is used in the main technical result of this section. Proposition 28 in particular says that all maximal plays that conform to  $s_{\alpha,c}$  are finite. If  $\sigma_\alpha$  is rejecting, there is a choice function on  $\sigma_\alpha$  that only selects rejecting boxes. The desired theorem is then immediate.

► **Proposition 28.** *Let  $c$  be a choice function on  $\sigma_\alpha$ . There is a strategy  $s_{\alpha,c}$  such that all maximal plays starting in  $\alpha$  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  $\alpha$ , an associated sequence of levels  $lvl$ , and a choice function  $c$  for  $\alpha$  and  $lvl$ , there is a strategy  $s_{\alpha,c}$  such that all maximal plays conform to it and starting in  $\alpha$  end in a terminal word  $w$  with  $\rho_w \in \{c(K) \mid K \in \sigma_\alpha\}$ . This proves the proposition by choosing  $\alpha$  and  $c$  as given and  $lvl = i_0 \dots i_0$ , 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 27, and  $lvl_\eta \prec lvl$ . This means we can combine Lemma 25 and Lemma 26 (for the step case) into a Noetherian induction. The latter lemma does not state that  $lvl_\eta$  assigns a positive value to each non-terminal, which was a requirement on  $lvl$ . This follows from Lemma 25 and the fact that  $c_\eta$  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 29.** *Let  $\alpha \in W^{\mathcal{L}(A)}$  and let  $c$  select a rejecting box in each clause of  $\sigma_\alpha$ . Then  $s_{\alpha,c}$  is a winning strategy for refuter for the non-inclusion game played from  $\alpha$ .*

**Implementing Winning Strategies.** The strategy  $s_{\alpha,c}$  from Proposition 28 is not positional. However, it can be implemented as a *strategy with finite memory*. We initialize it with a choice function on  $\sigma_\alpha$  and keep track of the current refinement of this function in each play. Observe that it suffices to store the image of the current choice function, which is a set of boxes, and the number of boxes is finite. To further optimize the implementation, instead of storing all Kleene approximants we annotate every box in the fixed point by the iteration step in which it entered the solution. Rather than selecting an arbitrary rejecting box from each clause to initialize the choice function, one should then choose the rejecting box which entered the solution the earliest.

The strategy can be implemented without levels but at the expense of an enumeration. Whenever prover makes a move, the strategy computes the refinement of the choice function.

Given a position  $wX\beta$  owned by refuter, it checks in increasing order, for all  $j = 0, 1, \dots$  whether there is a rule  $X \rightarrow \eta$  such that there is a refinement of the current choice function on  $\rho_w; \sigma_\eta^j; \sigma_\beta$ . Lemma 26(2) guarantees the existence of such a rule. Moreover, in terms of levels,  $j$  will be smaller than the current level of  $X$ , say  $i$ . This shows termination of the overall procedure. Another advantage of this enumeration of Kleene approximants is that we may find a refinement of the choice function with an iteration number  $j < i - 1$ .

One can also use a pushdown automaton to implement the strategy. Its stack will always have one entry for each non-terminal of the sentential form currently under consideration, storing the symbol, the level, and the formula for the corresponding suffix of the sentential form. The automaton iterates over the grammar rules and uses the stored level for the current non-terminal and the formula for the suffix to determine which rule to pick. The implementation using bounded space requires linear time (in the size of the current sentential form) to select rules. In contrast to this, the pushdown strategy needs to be initialized once for the initial position and then can determine rules in constant time.

As soon as we know that a winning strategy for refuter exists (i.e. by evaluating the formula for the initial position), we can find one by a breadth-first search in the tree formed by all plays. The winning strategy will only generate finite plays and the tree has finite out-degree. So, by König's lemma, the tree formed by all plays conform to this strategy has to be finite. This allows us to obtain a *positional strategy*.

► **Example 30.** 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  $\sigma_Y$  has to be  $c(\{\rho_b\}) = \rho_b$ . In the first step, prover has no alternative but  $Y \rightarrow bX$ . Position  $bX$  has the formula  $\sigma_{bX} = \{\{\rho_b\}\}; \{\{\text{id}, \rho_{ab}\}\} = \{\{\rho_b, \rho_{bab}\}\} = \{\{\rho_b\}\}$ . Pick the same choice function as before. The rule  $X \rightarrow \varepsilon$  causes  $\text{id}$  to enter  $\sigma_X$  in the first Kleene step. This causes  $\rho_b$  to enter  $\sigma_{bX}$  also in the first step. Indeed, by choosing  $X \rightarrow \varepsilon$  refuter wins non-inclusion. ◀

## 6 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.

### 6.1 Hardness

We prove that deciding the non-inclusion game is 2EXPTIME-hard. Our proof of the lower bound follows the proof of the analogue result for the games considered in [32].

► **Theorem 31.** *Given a non-inclusion game and an initial position, deciding whether refuter has a winning strategy from the specified position is 2EXPTIME-hard.*

**Proof of Theorem 31.** Assume an alternating Turing machine  $M$  with exponential space bound - say  $2^{n^c}$  - and an input  $w$  is given. We construct a polynomial-sized game such that refuter has a winning strategy if and only if  $M$  accepts  $w$ .

A similar reduction has been given in [32]. There, the authors consider left-to-right games which work as follows: The first player picks a position in the current sentential form and the second player replaces the non-terminal at this position using a rule of his choice. Furthermore, whenever the player picking the positions skips a non-terminal, she is not allowed to choose it later. We cannot simply reduce the games considered in [32] to our setting, since in left-to-right games, the player who needs to have the finite winning strategy has to reach the regular target language instead of avoiding it.

Our approach is to use the grammar to generate a sequence of configurations of the Turing machine. Afterwards, the non-deterministic automaton detects whether the sequence forms a invalid or non-accepting computation. (This is similar to the proof that universality of NFAs is PSPACE-hard [8].) The game aspect allows to us generate a computation of an alternating Turing machine. We let refuter choose the transitions originating in existential states and we let prover choose the transitions originating in universal states. Since each configuration has exponential size, the polynomial-sized automaton is not able to check whether the head pointer of the Turing machine has been moved in an incorrect way (and therefore, the sequence of configurations is invalid). To solve this problem, we divide the derivation process into two parts: In the first part, refuter generates a sequence of configurations from right-to-left, only giving control over to prover to pick the transitions originating in universal control states. In the second part, prover does a right-to-left pass in which she wants to show that the configuration is invalid (or non-accepting). To this end, she places a marker on the encoding of a cell. Refuter tries to justify the computation and rebut the objection. To this end, she places a marker on the encoding of the cell with the same position in the previous configuration. With the help of the markers, the automaton is able to detect violations. The automaton will accept the resulting word if and only if the markers prove that it is an invalid computation.

We will use the symbols  $0, 1, [, ], (, ), \langle, \rangle$ , and  $, .$  Furthermore, each control state  $q \in Q_M$ , each tape symbol  $a \in \Gamma_M$ , and each transition  $\delta \in \Delta_M$  of the Turing machine is used as a symbol. Each of these symbols  $x$  will occur in four different versions in the grammar. Without any decoration, it is a non non-terminal  $x$  owned by prover. It may also be a non-terminal  $\bar{x}$  owned by refuter or a terminal  $x_p$  or  $x_r$ , where the subscripts indicate which player caused the symbol to be derived. Furthermore, there is an initial non-terminal  $S$  owned by refuter, several versions of the non-terminal  $\#$ , and the special terminal symbols  $START, STOP, ONUM, OTAPE, J$ .

At first, the derivation process will generate a sequence of configuration in a right-to-left fashion. This means that production rules of the shape  $\# \rightarrow \#x$  are applied. Note that since we only consider left-derivations, the symbols  $x$  that are inserted will not be touched as long as  $\#$  has not yet been replaced. Each configuration is encoded as a sequence

$$(a_0, \text{bin}(0)) (a_1, \text{bin}(1)) \dots (a_{i-1}, \text{bin}(i-1)) [q, (a_i, \text{bin}(i))] \dots (a_{2^{n^c}}, \text{bin}(2^{n^c})) ,$$

where  $a_0 \dots a_{2^{n^c}}$  is the content of the tape,  $q$  is the control state, and  $i$  is the position of the head pointer. Each cell also contains an encoding of its index on the tape. This is crucial for being able to detect invalid head pointer movement later. We require that each configuration has length  $2^{n^c}$ , i.e. we will show the trailing blank symbols explicitly. Two configurations are separated by a sequence  $\langle \delta \rangle$ , where  $\delta$  should be the transition of the Turing machine that was used to get from the old configuration (on the right) to the new configuration (on the left).

Once the symbol  $\#$  has been replaced, the sentential form is a sequence of non-terminals that are all owned by prover. Prover now can do a left-to-right pass in which she can place the marker as explained above.

To be precise, the derivation process will proceed in four phases: (1) The initial configuration of the machine is generated. (2) A sequence of configurations of the machine, separated by the transitions that were used, is generated from right to left. (3) Afterwards, we need to check that the sequence indeed encodes an accepting branch of a computation of  $M$  on  $w$ . A part of this is implemented in the grammar. Another part of this is checked by the inclusion in the language of the automaton. We elaborate on this below. For two of the

conditions, we need to use the help of the players, since grammar and automaton are only allowed to be polynomial-sized, and thus cannot process the exponential-sized configurations easily. Prover does a left-to-right pass over the sequence in which she can place an objection marker to show why the sequence is not accepting. (4) Refuter can then place a justification marker in the next configuration to rebut the objection.

The automaton checks that the resulting terminal word contains exactly one objection that was not correctly rebutted. This means prover wins if the sequence does not encode an accepting configuration. We will now look at each phase in detail.

- (1) In the first phase, the initial configuration is generated on the tape. During this phase, none of the players has a choice. After it has finished, the resulting sentential form is

$$\# [q_0, (\$, \text{bin}(0))] (w_1, \text{bin}(1)) \dots (w_k, \text{bin}(k)) (\sqcup, \text{bin}(k+1)) \dots (\sqcup, \text{bin}(2^{n^c})) \langle \text{START} \rangle ,$$

where  $\$$  is the marker for the left end of the tape,  $q_0$  is the initial state and  $w = w_1 \dots w_k$ . Note that it is possible to generate the exponential amount of  $(2^{n^c} - k)$  trailing blank symbols with a polynomial-sized grammar.

- (2) In the second phase, the string is prolonged to the left to represent a branch of the computation tree of  $M$  on  $w$ . We give a rough overview of how this can be implemented in the grammar. Later, we will refine this construction.

Initially, refuter has to replace  $\#$ . To do this, refuter has four choices: She can (a) write an arbitrary sequence of symbols  $x$ , representing control states, tape symbols and the symbols  $(, ), [, ], 0, 1$  and  $,$  using rules of the shape  $\# \rightarrow \#x$ , (b) write a transition  $\delta \in \Delta_\vee$  of the Turing machine originating in an existential state by choosing  $\# \rightarrow \# \langle \delta \rangle$ , (c) give control over to prover by choosing  $\# \rightarrow \#_P$ . Prover can then write a transition  $\delta \in \Delta_\wedge$  originating in a universal state and by choosing  $\#_P \rightarrow \# \langle \delta \rangle$ . Or refuter can (d) stop writing the branch by replacing  $\#$  by the sequence  $\langle \text{STOP} \rangle$ , which will end the second phase.

The sentential form obtained after Phase (2) is not a correctly encoded sequence of configurations that forms an accepting branch of a computation of  $M$  on  $w$  if and only if at least one of the following conditions is satisfied.

- (a) There is a top-level syntax error, i.e. the sentential form is not of the shape

$$\langle \text{STOP} \rangle \{ \{ 0 + 1 + q + a + [ + ] + ( + ) + , \}^* \langle \delta \rangle \}^* \{ 0 + 1 + q + a + [ + ] + ( + ) + , \}^* \langle \text{START} \rangle .$$

Here, we use curly braces for regular expressions to avoid ambiguity. The symbols  $q, a, \delta$  denote control states, tape symbols, and transitions respectively.

- (b) There is a low-level syntax error, i.e. one single configuration is not encoded as

$$\{ (a, \{ 0 + 1 \}^*) \}^* [q, (a, \{ 0 + 1 \}^*)] \{ (a, \{ 0 + 1 \}^*) \}^* .$$

- (c) There is a transition that was picked although it was not applicable. This means that control state or the symbol at the position of the head pointer do not match with the preconditions of the transition. Since prover can exclusively write transitions that require a universal control state, this will enforce that we let prover choose such transitions.
- (d) There is a configuration such that its control state is not the state resulting from the transition that was applied before.
- (e) The leftmost configuration does not contain the accepting control state  $q_{\text{accept}}$ .
- (f) There is a configuration in which any cell is numbered with a binary string of incorrect length or the first cell is not numbered with  $\text{bin}(0)$  or the last cell is not numbered with  $\text{bin}(2^{n^c})$ .



- (g) There is a configuration containing two successive cells  $(q, v)(q', v')$  such that  $v, v'$  encode numbers  $i, j$  with  $j \neq i + 1$ .
- (h) There is a configuration that was modified in an incorrect way (compared to the previous configuration and the chosen transition). This means the head pointer was moved inconsistently or a cell content was changed inconsistently. The latter might be the case if a cell was changed that was not affected by a transition, or the cell that was previously the position of the head pointer was not changed according to the chosen transition.

The Conditions (a) - (e) can be implemented in the grammar. We create polynomially-many copies of the symbol  $\#$  that keep track of a constant amount of information. For example, to handle a part of Condition (b), we can easily guarantee that exactly one control state is written between two transitions by going to a copy  $\#^{ctrl}$  after a transition  $\langle \delta \rangle$  was inserted. This version of the symbols that allows writing a control state (i.e. there is a rule  $\#^{ctrl} \rightarrow \#^{trans} q$  for each  $q \in Q$ ), but does not allow writing a transition. As shown in the rule, we go to a copy  $\#^{trans}$  after the control states has been inserted. Thus version allows writing a transition (i.e. there is a rule  $\#^{trans} \rightarrow \#^{ctrl} \langle \delta \rangle$ ), but does not allow writing a control state. To prevent violations of Condition (d), we can also keep track of last control state  $q$  and of the symbol  $a$  that is stored in the cell which is the position of the head pointer. We can implement the transition relation of the Turing machine in the grammar rules to ensure that only transitions can be picked that require the stored control state and tape symbol. This means we can only use a rule that inserts  $\langle \delta \rangle$  if  $\delta$  is a transition of the shape  $(q, a) \mapsto (q', a', d)$ . After the transition has been inserted, we enforce that in the following configuration, the control state coincides with the control state  $q'$  resulting from applying the transition, thus ensuring (d) to hold. We can also enforce that the rule that ends the second phase by inserting  $\langle STOP \rangle$  can only be picked if the previous control state was  $q_{accept}$  to handle Condition (e).

Condition (f) can be implemented in the automaton. To check that all binary encodings of numbers have the correct length  $n^c$ , the automaton guesses non-deterministically where a pattern of the form  $(q, v)$  occurs, where  $q$  is a control state and  $v \in \{0 + 1\}^*$  is a sequence of length not equal to  $n^c$ . To detect this, we create  $n^c$  branches of the automaton that detect sequences of length  $0, \dots, n^c - 1$  and one branch that detects sequences of length at least  $n^c + 1$ . The index of the first cell of each configuration has the encoding  $\text{bin}(0) = 0\dots 0$ . The automaton can check whether a illegal 1 occurs in the encoding. To do this, it guesses non-deterministically where a pattern of the form

$$\rangle(a, \{0 + 1\}^* 1 \{0 + 1\}^*) + \rangle[q, (a, \{0 + 1\}^* 1 \{0 + 1\}^*)]$$

occurs, where  $a$  is a tape symbol, and  $q$  is a control state. Note that the symbol  $\rangle$  marks the left end of configuration since it belongs to the sequence  $\langle \delta \rangle$  encoding a transition (or to  $\langle STOP \rangle$ ). Analogously, the automaton can verify that the index of the last cell has the encoding  $\text{bin}(2^{n^c}) = 1\dots 1$ .

To check Conditions (g) and (h), we require the help of the players. The Phases (3) and (4) of the derivation process will implement this.

- (3) After refuter has decided to end the process by choosing  $\# \rightarrow \langle STOP \rangle$ , we have a sentential form in which all other symbols are non-terminals owned by prover. Prover now does a left-to-right pass, in which she can mark a place by an objection symbol to show that the configuration is not valid. Formally, she can (a) replace a symbol by her terminal version by choosing  $x \rightarrow x_p$ , (b) place one of two kinds of objection symbols by choosing a rule  $x \rightarrow ONUM\ x$  or  $x \rightarrow OTAPE\ x$ , or (c) give control over to refuter by choosing  $x \rightarrow \bar{x}$ .

The automaton will later check that the resulting terminal word contains exactly one objection symbol. If there is no objection, prover admits that the branch is accepting.

Prover has two types of objection symbols. The symbol *ONUM* can be placed in front of two successive cells  $(a, v)(a', v')$  (respectively the variants where one of the cells is the position of the head pointer) to claim that  $v$  and  $v'$  encode numbers  $i$  and  $j$  such that  $j \neq i + 1$ . It is crucial that  $j \neq i + 1$  can be accepted for a polynomial-sized NFA for binary encoded numbers  $i, j \in \{0, \dots, 2^{n^c}\}$ .

To do the check, we guess the position  $l$  of  $v'$  (encoding  $j$ ) that will be the rightmost deviation from the encoding of  $i + 1$  prior to looking at the two cells. We store the position in the control state (which we can do since there are only polynomially-many possibilities, namely  $\log(2^{n^c}) = n^c$ ).

When the automaton reads the  $l^{\text{th}}$  bit of  $v$ , it stores the bit in its control states. When it then reads the  $l^{\text{th}}$  bit of  $v'$  it is able to accept if the  $v'$  does not encode  $i + 1$ . It is crucial that adding one to a binary number will change the rightmost 0 to 1 and flip all following 1s to 0. The following possibilities arise: (a) After position  $l$ , at least one zero follows in  $v$ . In this case, the bit at position  $l$  in  $v'$  should coincide since the addition can be performed further to the right. Check this and accept if the bits are different. (b) After position  $l$ , only 1s follow in  $v$ . In this case, the bit at position  $l$  in  $v'$  should be different since it is affected by the addition. Check this and accept if the bits coincide.

The symbol *OTAPE* can be placed in front of a cell  $(a, v)$  to claim that it was modified in an incorrect way. This means it was modified compared to the previous configuration although it was not the position of the head pointer in the previous configuration (and therefore should have stayed the same) or it was the position of the head pointer previously but was not modified according to the transition.

Since the automaton cannot identify the correct position in the next configuration due to the polynomial bound on its number of states, we will employ the help of refuter to identify the cell of the next configuration that the automaton has to compare against.

- (4) The automaton guarantees that after an objection symbols is placed, the control is always given to refuter (by choosing the rule  $x \rightarrow \bar{x}$  instead of  $x \rightarrow x_p$ ). Formally, the automaton rejects the input if a second objection symbol occurs or if the prover-version  $x_p$  of any terminal occurs.

Assume prover placed the symbol *OTAPE* in front of a cell in configuration  $c$ . Prior to Phase (3), the sentential form had the shape  $\dots c \langle \delta \rangle c' \dots$ , where  $c$  is the configuration that resulted from applying transition  $\delta$  to configuration  $c'$ . Refuter can now place a justification symbol  $J$  in  $c'$  to show that the objection is not valid.

If *OTAPE* was inserted in front of a cell  $(a, v)$  in a configuration  $c$ , refuter should place  $J$  in front of the preceding cell  $(b, v')$  of the previous configuration  $c'$ . This means if  $v$  and  $v'$  encode numbers  $i$  and  $j$ , we have  $j = i - 1$ . (If one of the cells is the position of the head pointer, the symbol should be inserted in front of  $[\cdot]$ .)

First, the algorithm will check that refuter marked the correct cell. To do this, a procedure analogous to the one explained in Phase (3) can be used to accept if  $v'$  is not as desired. Then the automaton can use the marked cell  $(a, v)$  in  $c$  and the three marked cells  $(b_j, \text{bin}(j))$ ,  $(b_i, \text{bin}(i))$ ,  $(b_{i+1}, \text{bin}(i+1))$  in  $c'$  (respectively the version with the control state annotation somewhere) to verify: (a) If  $(b_i, \text{bin}(i))$  was not the position of the head pointer in  $c'$ , then the cell should remain unchanged, i.e.  $b_i = a$ . (b) If  $(b_i, \text{bin}(i))$  was the position of the head pointer in  $c'$ , then  $a$  is the symbol that results from applying the chosen transition. (c) If  $(a, \text{bin}(i))$  is the position of the head pointer in  $c$ , then

the control state was previously placed at  $(b_j, \text{bin}(j))$ ,  $(b_i, \text{bin}(i))$ , or  $(b_{i+1}, \text{bin}(i+1))$ , according to the chosen transition.

To check (b) and (c), the automaton enforces that there is exactly one transition  $\langle \delta \rangle$  separating the configuration containing *OTAPE* and the configuration containing *J*. It can store the transition  $\delta$  in its control state while switching between the configurations. The cases in which  $(b', \text{bin}(i))$  is the first or last cell of a configuration can be treated similarly.

Note that the operations performed by the automaton that are described as if they would happen sequentially here are implemented in parallel. This can be done using a constant number of intersections and unions, without changing the size of the automaton being polynomial.

Overall, we obtain that if the ATM  $M$  accepts  $w$ , refuter can find an accepting branch, no matter which transitions are chosen by prover in the universal states. If refuter writes down the correct encoding of the corresponding finite sequence of configurations, she is able to react to any objection prover might place in Phase (3). The finite automaton will not accept, since it cannot detect a valid objection, so we have derived a word outside its language.

If the ATM  $M$  does not accept  $w$ , prover has a way of selecting transitions originating in universal states such that the resulting branch of the computation tree is not accepting, independent of the choices of refuter in the existential states. The construction guarantees that prover is allowed to choose the grammar rule that inserts the transition if the control state in the previous configuration was universal. By choosing the grammar rules that insert the transitions leading to a non-accepting branch, prover can ensure that no sentential form encoding a finite sequence of configurations ending in an accepting configuration can be derived. If refuter tries to cheat by enforcing an infinite derivation, refuter loses the game by definition. If she cheats by choosing an incorrect encoding (that is not prevented by the grammar), prover can place an objection mark to which refuter can not react. The automaton will then detect the valid objection and accept, so we have derived a word in its language. ◀

## 6.2 Membership

The following algorithm implements the fixed-point iteration discussed in Section 3, executed on formulas in CNF (see the Subsection 5.1).

► **Algorithm 32.** Given a non-inclusion game and an initial position  $\alpha$ , the following algorithm computes whether refuter has a winning strategy from the given position.

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

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

► **Theorem 33.** *Given a non-inclusion game and an initial position, Algorithm 32 computes whether refuter has a winning strategy from the given position in time  $\mathcal{O}\left(|G|^2 \cdot 2^{2^{|Q|^{c_1}}} + |\alpha| \cdot 2^{2^{|Q|^{c_2}}}\right)$  for some constants  $c_1, c_2 \in \mathbb{N}$ .*

**Proof of Theorem 33.** We will analyze (1) the number of iterations needed to obtain the fixed-point solution (2) the time consumption per iteration (3) the cost of constructing and evaluating the formula for the given initial position.

Let  $k = 2^{|Q|^2}$  be the number of boxes. Every clause has at most size  $k$  and there are  $2^k$  different clauses, so every formula has size at most  $k \cdot 2^k$ . Computing conjunction and disjunction according to Lemma 18 is polynomial in the size of the formulas.

To compute the relational composition according to Lemma 20, we need to iterate over the at most  $2^k$  clauses and over the at most  $(2^k)^k = 2^{k^2}$  functions mapping boxes to clauses. Each clause  $K$  and function  $z$  determines a clause of the relation composition. To obtain its boxes, we need to iterate over the at most  $k$  boxes  $\rho$  of  $K$  and compute  $\rho; z(\rho)$ . To do this, we need to iterate over the at most  $k$  boxes  $\tau$  of  $z(\rho)$  and compute  $\rho; \tau$ . This requires that we check for each pair  $(q, q')$  of states whether there is a suitable  $q''$  such that  $(q, q') \in \rho, (q, q'') \in \tau$ . Overall, to compute the relational composition of two formulas, we need

$$2^k \cdot 2^{k^2} \cdot k \cdot k \cdot |Q|^3 \in \mathcal{O}\left(2^{2|Q|^{c_2}}\right)$$

steps, for some constant  $c_2 \in \mathbb{N}$ .

- (1) The length of any chain of strict implications of formulas over a set of  $k$  atomic propositions is at most  $2^k$ . To prove this, note that modulo logical equivalence, a formula is uniquely characterized by the set of assignments such that the formula evaluates to *true* under them. Strict implication between two formulas implies strict inclusion between the sets. The statement follows since there are at most  $2^k$  different truth assignments. We can use this to obtain that the number of iterations is bounded by  $|N| \cdot 2^k$ , since the sequence of intermediary solutions is a chain in the product domain, and the height of the product domain is the height of the base domain multiplied by the number of components.
- (2) Per iteration, we need to carry out at most  $|G|$  conjunctions, disjunctions and relational compositions. Per grammar rule, we need to compute at most one conjunction or disjunction, depending on the owner of the non-terminal. For each symbol on the right-hand side of a grammar rule, we need to compute at most one relational composition. Overall, for one iteration, we need

$$|G| \cdot \left(2^{2|Q|^{c_2}} + 2^{2|Q|^{c_3}} + 2^{2|Q|^{c_4}}\right) \in \mathcal{O}\left(|G| \cdot 2^{2|Q|^{c_1}}\right)$$

steps, for some constants  $c_1, c_3, c_4 \in \mathbb{N}$ . Here,  $2^{2|Q|^{c_2}}$  is the cost of computing the relational composition as calculated earlier and  $2^{2|Q|^{c_3}}$  and  $2^{2|Q|^{c_4}}$  are rough estimations for computing conjunction and disjunction.

Combining (1) and (2) together with the rough estimation  $|N| \leq |G|$  yields the first summand

$$\mathcal{O}\left(|G|^2 \cdot 2^{2|Q|^{c_1}}\right).$$

- (3) Assume the initial position  $\alpha$  has length  $|\alpha| = l$ . It remains to compute  $l - 1$  relational compositions, which can be done in  $(l - 1) \cdot 2^{2|Q|^{c_2}}$  steps, and to check whether the resulting formula is rejecting. The latter can be done by iterating over all at most  $2^k$  clauses, and checking whether one of the at most  $k$  boxes in them is rejecting. Checking whether a box  $\rho$  is rejecting can be done in  $|Q|$  time, since we need to check for the absence of all pairs  $(q_0, q_f)$  in  $\rho$ , where  $q_f \in Q_F$ .

Overall, we need

$$(l - 1) \cdot 2^{2|Q|^{c_1}} + 2^k \cdot k \cdot |Q| \in \mathcal{O}\left(l \cdot 2^{2|Q|^{c_2}}\right)$$

steps. ◀

The following corollary is an immediate consequence of Theorem 31 and Theorem 33.

► **Corollary 34.** *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.

### 6.3 Solving More General Games

One should also note that 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. Assume the winning condition is specified by a predicate on the boxes of the automaton (respectively by a predicate on words that is well-defined on the equivalence classes introduced by boxes). We can lift the definition to obtain a predicate on CNF-formulas over boxes by distributing it over conjunction and disjunction. We get that a formula satisfies the predicate if and only if there is a choice function picking a box out of each clause that satisfies the predicate. Initializing Proposition 28 with such a choice function will provide a winning strategy that ensures that the game ends in a word such that its box satisfies the predicate after finitely many steps.

If the predicate can be evaluated in doubly exponential time (in the size of the whole input), the time complexity of the algorithm does not change. In this paper, we mostly considered the *reject*-predicate, that checks for the absence of a transition  $(q_0, q_f)$  with  $q_f \in Q_F$  in boxes (and therefore is satisfied if the words are not in the language of the automaton). We can also consider its negation, the *accept*-predicate, that checks for the presence of such a transition (and therefore is satisfied if the words are in the language of the automaton). Instantiating the algorithm for this predicate yields a procedure to solve a type of game in which refuter wants to obtain a word in the regular target language after finitely many steps. One can show that solving this problem is also 2EXPTIME-complete by reducing the left-to-right games considered in [32].

## 7 Experiments

We have implemented our algorithm in C++ [1] and compared it to an implementation of Cachat's algorithm [16] 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 left-most non-terminal) will be represented in  $P$  by a configuration  $(Q', X\beta)$ , where  $Q'$  is the set of states that  $A$  can be in after processing  $w$ . To be precise, for each subset  $Q' \subseteq Q$  of states of  $A$ ,  $P$  has two control states  $Q'_\square$  and  $Q'_\circ$ , one for each player.

- (1) If  $P$  is in control state  $Q'_\star$ , and the topmost stack symbol is a terminal  $a \in T$ , it can be popped and the control state is changed to  $Q''_\star$ , where  $Q''$  is the set of states with  $Q' \xrightarrow{a} Q''$ .

- (2) If  $P$  is in control state  $Q'_{\star}$ , and the topmost stack symbol is a non-terminal  $Y$  owned by the other player  $\overline{\star}$ ,  $P$  goes to control state  $Q'_{\overline{\star}}$  without modifying the stack.
- (3) If  $P$  is in control state  $Q'_{\star}$ , and the topmost stack symbol is a non-terminal  $X$  owned by  $\star$ , there is one transition in  $P$  for each rule  $X \rightarrow \alpha$  with  $X$  on its left-hand side that pops  $X$  and pushes  $\alpha$  without changing the control state.

The  $P$ -AFA just checks that the set of states  $Q'$  of the current control state  $Q'_{\star}$  does not contain a final state and that the stack is empty.

We have to encode both the automaton and the grammar into the pushdown system. If we keep the whole sentential form on the stack, the terminal prefix prevents us from modifying the non-terminals.

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 [42]. 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  $X \rightarrow aYb$ , 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  $|Q|$ ,  $|T|$ ,  $|N_{\circ}|$  and  $|N_{\square}|$ , we tried out several combinations. The entry  $x/y/z$  in the table below stands for  $|Q| = x, |T| = y, |N_{\circ}| = |N_{\square}| = z$ . 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.

	naive Kleene		worklist Kleene		Cachat	
	average time	% timeout	average time	% timeout	average time	% timeout
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



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 using a worklist. Since in every step not only single transitions of the  $P$ -AFA but whole paths in the  $P$ -AFA are considered, handling the states of the  $P$ -AFA with a worklist is not possible. But it is possible to handle the states of the pushdown system  $P$  using a worklist: Whenever a transition labeled by stack symbol  $a$  is added to the  $P$ -AFA, all control states of  $P$  that have a transition that pushes  $a$  to the stack have to be added to the worklist.

Unfortunately, this does not help for the instances obtained by encoding our type of game. For every non-terminal  $X$  owned by player  $\star$  (1) all states  $Q'_{\star}$  of the of the opponent  $\bar{\star}$  have to be added to the worklist, since there is a transition from  $Q'_{\star}$  to  $Q'_{\star}$  that pops  $X$  and pushes  $X$ , and (2) if  $X$  occurs on the right-hand side of at least one rule of the grammar, say in  $Y \rightarrow \alpha$ , all states  $Q'_{\star}$  of the of the player  $\star$  have to be added to the worklist, since there is a transition that pops  $Y$  and pushes  $\alpha$  for each such state.

The terminal symbols are handled in the very first iteration of Cachat. This means that starting from the second iteration, adding a transition to the  $P$ -AFA will cause almost all states of  $P$  to be added to the worklist. In our experiments, the worklist variant was slower by at least one order of magnitude, even on small examples.

## 8 Algorithmic Considerations

We discuss how to further speed-up the worklist implementation by two heuristics prominent in verification: Lazy evaluation [20] and antichains [22, 5, 6]. 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.

### 8.1 Lazy Evaluation

Assume we are interested in whether refuter wins from a given sentential form  $\alpha$ . For simplicity, introduce a fresh non-terminal  $S$  with single rule  $S \rightarrow \alpha$ . That refuter wins the game can be concluded as soon as the formula  $\sigma_S$  becomes rejecting. The fixed-point iteration

does not have to continue beyond this point. Inspired by [20], the idea of early termination upon reaching a target can be elaborated further into a *lazy evaluation* technique: Also compositions of formulas have to be evaluated only up to the point where the value influences the overall answer. Consider the test whether  $(F \vee F'); M$  is rejecting. If  $F; M$  is already rejecting, then the whole formula is rejecting, and the evaluation of  $F'; M$  can be skipped.

The lazy algorithm evaluates the predicate  $reject(\perp)$ , which will yield *true* iff  $\sigma_S$  is rejecting. The test is done on the fly, while unfolding the fixed-point computation from  $\perp$  ( $X \mapsto false$  for all  $X \in N$ ). The algorithm uses a set of rules (discussed below) that reduce the test on their left-hand side into a Boolean combination of tests on the right, and which return a Boolean value on boxes.

The first rule unfolds the fixed-point computation. A reasonable implementation of the algorithm would not use the Kleene iteration on the product domain from Section 3 but a variant of the more efficient chaotic iteration (e.g. the worklist algorithm) [38]. The following rule corresponds to one unfolding of chaotic iteration:

$$reject(\sigma) \iff reject(\sigma_S) \vee unfold \quad (I)$$

$$\text{where } unfold = \begin{cases} reject(\sigma[X \mapsto f_X(\sigma)]) & \text{for some } X \in N \text{ with } f_X(\sigma) \not\sqsubseteq \sigma_X \\ false & \text{if } f_X(\sigma) \sqsubseteq \sigma_X \text{ for all } X \in N. \end{cases}$$

Intuitively, either the formula  $\sigma_S$  is already rejecting in the current assignment, or it may become rejecting in some further unfolding of the fixed-point computation. The disjunct *unfold* updates one of the variables based on the current assignment  $\sigma$ , or it terminates the fixed-point computation if no variable can be updated, that is, if the current assignment is already the least fixed point.

To evaluate the first disjunct in Rule (I), we have to evaluate  $reject(\sigma_S)$  on a term  $\sigma_S$  built on top of boxes using Boolean connectives and relational composition. To evaluate the second disjunct, we need to decide subsumption. On boxes,  $reject(\rho) = true$  iff  $\rho$  is rejecting, and  $\rho \sqsubseteq \tau$  is evaluated according to the notion of subsumption in use. On Boolean connectives, rejection and subsumption queries are evaluated as Boolean combinations of the respective queries over the subformulas. If the tested formulas are relational compositions, then the composition is first pushed down one level using Definition 5, or, on leaf level, it is eliminated by composing the boxes. This is expressed by the following rules where  $\rho \in M_A$ ,  $F, F', M, M', G, G' \in BF_A$ , and  $\{\star, \bar{\star}\} = \{\wedge, \vee\}$ :

$$\begin{aligned} reject(F \star G) &\iff reject(F) \star reject(G) & reject(F; M) &\iff reject(eval(F; M)) \\ F \star F' \sqsubseteq G &\iff F \sqsubseteq G \bar{\star} F' \sqsubseteq G & F; M \sqsubseteq G &\iff eval(F; M) \sqsubseteq G \\ F \sqsubseteq G \star G' &\iff F \sqsubseteq G \star F \sqsubseteq G' & F \sqsubseteq G; M &\iff F \sqsubseteq eval(G; M) \end{aligned}$$

$$\begin{aligned} \text{where } eval(F \star F'; M) &= F; M \star F'; M \\ eval(\rho; M \star M') &= \rho; M \star \rho; M'. \end{aligned}$$

Using these rules, the algorithm constructs an alternating proof tree rooted at  $reject(\perp)$  which branches at Boolean connectives on the right-hand sides of the rules. It may conclude before the fixed-point computation terminates, and when it concludes, the proof tree may still contain not fully evaluated queries and compositions which do not influence the final answer. These unfinished computations are efficiency gains over the basic algorithm which always iterates until fixed point and evaluates all compositions.

We note that if the rules were implemented verbatim, the tree would contain repetitive evaluations of the same queries and compositions. This could be countered by joining

equivalent queries and keeping the proof in the form of a directed acyclic graph. Similarly, it is useful to keep occurrences of the same compositions as references to one representative and evaluate all of them simultaneously by evaluating the representative.

## 8.2 Antichains

The antichain approach benefits from a weaker notion of implication. Consider a preorder  $\sqsubseteq \subseteq M_A \times M_A$  that reflects the reject predicate as follows: If  $\rho \sqsubseteq \tau$  and  $\rho$  is rejecting, then  $\tau$  should be rejecting. These implications among boxes justify further implications among formulas. Formally, formula  $F$  *subsumes*  $G$ , written  $F \sqsubseteq G$ , if  $\bigwedge_{\rho, \tau \in M_A: \rho \sqsubseteq \tau} \rho \Rightarrow \tau \models F \Rightarrow G$ . Under the condition that the following analogue of Lemma 6 holds, it is possible to reason modulo subsumption instead of implication:

$$\forall F, F', M, M' \in \text{BF}_A: \text{ If } F \sqsubseteq F' \text{ and } M \sqsubseteq M', \text{ then } F; M \sqsubseteq F'; M'. \quad (\text{II})$$

Subsumption was used in [22, 5, 6], where a fixed point over sets of boxes is computed to check language inclusion among Büchi automata. For the definition of  $\sqsubseteq$ , the simplest possibility is to take inclusion among boxes [22]. A more advanced option, following [6], further weakens the set inclusion using simulation on states of  $A$ .

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 [22, 5, 6].

## 9 Related Work

We already discussed the relation with Cachat’s work [16]. Walukiewicz [43] 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 [43] is concerned with is different from ours in several respects. The game aspect is given by the specification (a  $\mu$ -calculus formula), not by the system as in our case. Moreover, (infinite) parity games are generally harder than safety: [43] is exponential both in the system and in the specification, while our construction is exponential only in the specification. Piterman and Vardi [33] study a similar variant of the problem and come up with a solution originating in the automata-theoretic approach [30].

The work [32] 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. 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 6). They give an optimal algorithm that uses Cachat’s algorithm for pushdown systems. Our algorithm also has optimal time complexity, but contrary to [32], it is based on procedure summaries rather than on saturation. [32] was further elaborated on and generalized in [11, 36].

Methods for solving variants of pushdown games, related mostly to saturation (see [17] for a survey on saturation-based methods), are implemented in several tools. [13] targets higher-

order pushdown systems, related to it is [14], [41] implements an optimized saturation-based method, [26] solves the full case of parity games. [34] 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 8, were developed in the context of finite automata and games [44, 45], and generalized to Büchi automata [22, 5, 6] with a fixed point over sets of boxes. Our lazy evaluation is inspired by [20]. Our framework is compatible with techniques for reachability in well-structured transition systems (WSTS) that proceed backwards [4]. We believe that techniques like [29, 23, 28, 7, 24] 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  $\sigma$  where  $\sigma_S$  is the conjunction of all rejecting boxes and  $\sigma_X = \text{false}$  for every other  $X \in N$ , and the initial state would be the assignment  $\perp$ . Another interesting possibility would be to adapt Newton iteration [19].

The transition monoid can be traced back at least to Büchi [15], and was prominently used e.g. in [3].

## References

- 1 Implementation of our algorithm. <https://concurrency.informatik.uni-kl.de/rigg.html>. Published: 2016-17-07.
- 2 WALi. <https://research.cs.wisc.edu/wpis/wpds/download.php>. Visited: 2016-16-07.
- 3 A. A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for  $\text{bu}\&\text{uml}\&\text{chi}$  automata with applications to temporal logic. In *ICALP*, volume 194 of *LNCS*, pages 217–237. Springer, 1985.
- 4 P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313–321, 1996.
- 5 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.
- 6 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.
- 7 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.
- 8 A. V. Alfred and J. E. Hopcroft. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing Co., 1974.
- 9 D. Beyer. Software verification and verifiable witnesses (report on sv-comp). In *TACAS*, volume 9035 of *LNCS*, pages 401–416. Springer, 2015.
- 10 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.
- 11 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, 2013.
- 12 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.
- 13 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.
- 14 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.
- 15 J. R. Büchi. *On a Decision Method in Restricted Second Order Arithmetic*, pages 425–435. Springer, 1990.
- 16 T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *ICALP*, volume 2380 of *LNCS*, pages 704–715. Springer, 2002.
- 17 A. Carayol and M. Hague. Saturation algorithms for model-checking pushdown systems. In *AFL*, volume 151 of *EPTCS*, pages 1–24, 2014.
- 18 B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. CUP, 1990.
- 19 J. Esparza, S. Kiefer, and M. Luttenberger. Newtonian program analysis. *JACM*, 57(6), 2010.
- 20 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.
- 21 A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. *ENTCS*, 9:27–37, 1997.
- 22 S. Fogarty and M. Y. Vardi. Efficient Büchi universality checking. In *TACAS*, volume 6015 of *LNCS*, pages 205–220. Springer, 2010.
- 23 Z. Ganjei, A. Rezzine, P. Eles, and Z. Peng. Lazy constrained monotonic abstraction. In *VMCAI*, volume 9583 of *LNCS*, pages 147–165. Springer, 2016.

- 24 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.
- 25 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.
- 26 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.
- 27 M. Heizmann, J. Hoenicke, and A. Podelski. Nested interpolants. In *POPL*, pages 471–482. ACM, 2010.
- 28 A. Kaiser, D. Kroening, and T. Wahl. Efficient coverability analysis by proof minimization. In *CONCUR*, pages 500–515. Springer, 2012.
- 29 J. Kloos, R. Majumdar, F. Niksic, and R. Piskac. Incremental, inductive coverability. In *CAV*, volume 9206 of *LNCS*, pages 158–173. Springer, 2013.
- 30 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.
- 31 Z. Long, G. Calin, R. Majumdar, and R. Meyer. Language-theoretic abstraction refinement. In *FASE*, volume 7212 of *LNCS*, pages 362–376. Springer, 2012.
- 32 A. Muscholl, T. Schwentick, and L. Segoufin. Active context-free games. *Theory of Computing Systems*, 39(1):237–276, 2005.
- 33 N. Piterman and M. Y. Vardi. Global model-checking of infinite-state systems. In *CAV*, volume 3114 of *LNCS*, pages 387–400. Springer, 2004.
- 34 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.
- 35 T. Reps, S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *POPL*, pages 49–61. ACM, 1995.
- 36 M. Schuster and T. Schwentick. Games for active XML revisited. In *ICDT*, volume 31 of *LIPICs*, pages 60–75. Dagstuhl, 2015.
- 37 S. Schwoon. *Model-Checking Pushdown Systems*. PhD thesis, 2002.
- 38 H. Seidl, R. Wilhelm, and S. Hack. *Compiler Design - Analysis and Transformation*. Springer, 2012.
- 39 O. Serre. Note on winning positions on pushdown games with omega-regular conditions. *IPL*, 85(6):285–291, 2003.
- 40 M. Sharir and A. Pnueli. Two approaches to interprocedural data flow analysis. Technical Report 2, New York University, 1978.
- 41 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.
- 42 D. Tabakov and M. Y. Vardi. Experimental evaluation of classical automata constructions. In *LPAR*, volume 3835 of *LNCS*, pages 396–411. Springer, 2005.
- 43 I. Walukiewicz. Pushdown processes: Games and model-checking. *IC*, 164(2):234–263, 2001.
- 44 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.
- 45 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.