Liveness Verification and Synthesis: New Algorithms for Recursive Programs

Roland Meyer, Sebastian Muskalla, and Elisabeth Neumann

TU Braunschweig, {roland.meyer, s.muskalla, e.neumann}@tu-bs.de

Abstract

We consider the problems of liveness verification and liveness synthesis for recursive programs. The liveness verification problem (LVP) is to decide whether a given ω-context-free language is contained in a given ω-regular language. The liveness synthesis problem (LSP) is to compute a strategy so that a given ω-context-free game, when played along the strategy, is guaranteed to derive a word in a given ω-regular language. The problems are known to be EXPTIME-complete and 2EXPTIME-complete, respectively. Our contributions are new algorithms with optimal time complexity. For LVP, we generalize recent lasso-finding algorithms (also known as Ramsey-based algorithms) from finite to recursive programs. For LSP, we generalize a recent summary-based algorithm from finite to infinite words. Lasso finding and summaries have proven to be efficient in a number of implementations for the finite state and finite word setting.

1 Introduction

A major difficulty in program analysis is the combination of control and data aspects that naturally arises in programs but is not matched in the analysis: Control aspects are typically checked using techniques from automata theory whereas the data handling is proven correct using logical reasoning. A promising approach to overcome this separation of techniques is a CEGAR loop recently proposed by Podelski et al. [15]. The loop iteratively checks inclusions of the form $\mathcal{L}(G) \subseteq \mathcal{L}(B)$. Here, G is a model of the program, in the recursive setting a context-free grammar. The automaton B is a union consisting of (1) the property of interest and (2) languages of computations that were found infeasible during the iteration by logical reasoning. The approach has been generalized from recursive to parallel [12, 19] and to parameterized programs [10], from safety to liveness [11], and from verification to synthesis [16].

We focus on the algorithmic problem behind Podelski's CEGAR loop: Inclusion checking. To be precise, we consider the case of recursive programs and study the problems of liveness verification and synthesis defined as follows. The *liveness verification problem (LVP)* takes as input a context-free grammar G abstracting the recursive program of interest and a Büchi automaton B specifying the liveness property. The task is to check whether the ω-context-free language generated by the grammar is included in the ω-regular language of the automaton, $\mathcal{L}^\omega(G) \subseteq \mathcal{L}^\omega(B)$. The *liveness synthesis problem (LSP)* replaces the context-free grammar by a context-free game between two players: Player prover tries to establish the inclusion in an ω-regular language and player refuter tries to disprove it. The task is to synthesize a strategy s such that prover is guaranteed to win all plays by following the strategy, $\mathcal{L}^\omega(G@s) \subseteq \mathcal{L}^\omega(B)$. The precise complexity of both problems is known (see below).

Our contribution is a generalization of two recent algorithms that have proven efficient in the context of finite-state systems (for verification) and finite word languages (for synthesis) to the setting of ω-languages of recursive programs. The study of new algorithms is motivated by the characteristics of the inclusion checks invoked in Podelski's approach: (1) The left-hand side modeling the program is substantially large. (2) The right-hand side for the specification is typically small but grows with the addition of counterexample languages. (3)

These inclusion checks are invoked in an iterative fashion. Our algorithms take into account these characteristics as follows. First, they avoid any computation on the grammar (like intersections that would be executed in an iterative fashion). Second, they may terminate early if the automaton bears redundancies that may occur when languages are added. This early termination makes them particularly suitable for use in a refinement loop.

For the liveness verification problem LVP, we develop a *lasso-finding algorithm*. Lasso algorithms have been proposed in [14] and further refined in [1, 2]. They rely on the fact that ω-regular languages can be stratified into finite unions of languages $L(\tau)L(\rho)^\omega$ [6]. Here, τ and ρ are relations over the states of the Büchi automaton. They denote the regular languages of all words that yield the prescribed state changes. With this stratification, disproving the inclusion amounts to finding a word derivable in the ω-context-free language whose representation $L(\tau')L(\rho')^\omega$ belongs to the complement of $\mathcal{L}^\omega(B)$. Checking membership in the complement amounts to proving the absence of an accepting cycle, a lasso, in the relation ρ' (when seen as a graph).

Algorithmically, the challenge is to compute the languages $L(\tau')L(\rho')^\omega$ induced by the words derivable in the ω-context-free grammar. We view the grammar as a system of inequalities and compute the least solution over (sets of) such relations. The problem is to make sure that the words represented by $L(\rho')$ can be ω-iterated. The solution is to find a lasso also in the ω-context-free grammar. To this end, we let the system of equations not only represent the non-terminal from which a terminal word is generated, but also the non-terminal from which the infinite computation will continue. With this idea, the system is quadratic in the size of the grammar. The height of the lattice is exponential in the number of states of the automaton. Indeed, LVP is known to be EXPTIME-complete [4]. The algorithm may terminate early if a language is found that disproves the inclusion.

For the liveness synthesis problem LSP, we develop a *summary-based approach*. Summaries [27, 22] represent procedures in terms of their input-output relationship on the shared memory.[1]. Recently, summary-based analyses have been generalized to safety games by replacing relations by positive Boolean formulas of relations [16]. We build upon this generalization and tackle the case of infinite words as follows. In a first step, we determinize the given Büchi automaton into a parity automaton. The second step computes formula summaries for safety games that have the parity automaton as the right-hand side. Interestingly, it is sufficient to only maintain the output effect of a procedure. In a third step, we connect the formula summaries to a parity game. Overall, the algorithm runs in 2EXPTIME and indeed the problem is 2EXPTIME-complete. The hardness is because finite games as considered in [20, 16] can be seen as a special case of LSP. Membership in 2EXPTIME can be shown using the techniques from [29].

It is well known that pushdown parity games can be reduced to finite state parity games, even with a summary-like approach [29]. Our algorithm can therefore bee seen as a symbolic implementation of Walukiewicz's technique where formulas represent attractor information. Besides the compact representation, it has the advantage of being able to make use of all techniques and tools that are developed for solving fixed point equations.

Related Work. We already mentioned the related work on the LVP and lasso finding. Parity games on the computation graphs of pushdown systems have been studied by Walukiewicz in [29]. He reduces them to parity games on finite graphs; a technique that could also be employed to solve the LSP in 2EXPTIME. Our algorithm which is based on solving a system

[1] Summaries resemble the aforementioned relations τ and ρ, see Section 3.

of equations has the same worst-case complexity but is amenable to recent algorithmic improvements, as has been shown in [16].

Cachat [7] considers games defined by pushdown systems in which the winning condition is reaching a configuration accepted by an alternating finite automaton once resp. infinitely often. He solves them by saturating the finite automaton, in contrast to our method which uses summarization. Although the LSP could be reduced to this type of game, the reduction has been shown to be inefficient for the case of finite games in [16]. Extensions towards higher-order systems exist [5].

The decidability and complexity of games defined by context-free grammars has been studied by Muscholl et. al. in [20] and extended in [3, 25]. Their study considers finite games and has an emphasis on lower bounds. We rely on their result for the 2EXPTIME-hardness and focus on the algorithmic side.

Acknowledgements. We thank Prakash Saivasan and Igor Walukiewicz for discussions.

2 ω-Context-Free Languages

To formulate the problem of liveness verification for recursive programs, we recall the notion of ω-context-free languages [18, 8, 13]. A language $\mathcal{L} \subseteq T^\omega$ of infinite words is *ω-context-free* if it can be written as a finite union of the form

$$\mathcal{L} = \bigcup_{i=1\dots n} V_i U_i^\omega \quad \text{with } V_i, U_i \subseteq T^* \text{ context-free languages of finite words.}$$

To accept or generate ω-context-free lanugages, Linna [18] as well as Cohen and Gold [8] define ω-languages of pushdown automata and context-free grammars, respectively. We choose the grammar-based formulation as it fits better the algebraic nature of our development. Actually, we slightly modify the definition in [8] to get rid of the operational notion of repetition sets. The correspondence will be re-established in a moment.

A *context-free grammar (CFG)* is a tuple $G = (N, T, P, S)$, where N is a finite set of non-terminals, T is a finite set of terminals with $N \cap T = \emptyset$, $P \subseteq N \times \vartheta$ is a finite set of production rules and $S \in N$ is an initial symbol. Here, $\vartheta = (N \cup T)^*$ denotes the set of sentential forms. We write $X \to \eta$ if $(X, \eta) \in P$. We assume that every non-terminal is the left-hand side of some rule. The *derivation relation* $\Rightarrow$ replaces a non-terminal X in α by the right-hand side of a corresponding rule. Formally, $\alpha \Rightarrow \beta$ if $\alpha = \gamma X \gamma'$, $\beta = \gamma\eta\gamma'$, and there is a rule $X \to \eta \in P$. For a non-terminal $X \in N$, we define the language $\mathcal{L}(X) = \{w \in T^* \mid X \Rightarrow w\}$ to be the set of terminal words derivable from X. The language of the grammar $\mathcal{L}(G) = \mathcal{L}(S)$ is the language of its initial symbol. For sentential forms, we define $\mathcal{L}(\alpha\beta) = \mathcal{L}(\alpha)\mathcal{L}(\beta)$, where $\mathcal{L}(a) = \{a\}$ for $a \in T \cup \{\varepsilon\}$.

Given a CFG G, we define its ω-language $\mathcal{L}^\omega(G)$ to contain all infinite words obtainable by right-infinite derivations. A *right-infinite derivation process* π of G is an infinite sequence of rules $\pi = X_0 \to \alpha_0 X_1, X_1 \to \alpha_1 X_2, \dots$ where the rightmost symbol of the right-hand side of each rule is the symbol on the left-hand side of the next rule, and $X_0 = S$ is the initial symbol of the grammar. The language of such a right-infinite derivation process is the language of infinite words

$$\mathcal{L}^\omega(\pi) = \mathcal{L}(\alpha_0)\mathcal{L}(\alpha_1)\dots$$

Note that $\mathcal{L}^\omega(\pi)$ is restricted to proper infinite words and thus does not contain $w_0 \dots w_k \varepsilon^\omega$. The *$\omega$-language of G*, denoted by $\mathcal{L}^\omega(G)$, is the union over the languages $\mathcal{L}^\omega(\pi)$ for all

right-infinite derivation processes π of G. The ω-languages obtained in this way are precisely the ω-context-free languages.

▶ **Example 1.** Consider the CFG G_{ex} with the rules $X \to req\ Y\ ack \mid XX$ and $Y \to s\ Y\ t \mid \epsilon$ and the initial symbol X. One can show that the grammar generates the ω-language $\mathcal{L}^\omega(G_{ex}) = \left\{ \left(req(s^{n_i}t^{n_i})ack\right)^\omega \mid n_i \geq 0\ \forall i \mathbb{N} \right\}$

▶ **Proposition 2.** *$\mathcal{L} \subseteq T^\omega$ is ω-context-free if and only if $\mathcal{L} = \mathcal{L}^\omega(G)$ for some CFG G.*

The same correspondence with the ω-context-free languages has also been shown for the models in [18, 8]. Hence, the three definitions capture the same class of languages. This not only justifies our modification of [8], it also allows us to convert a grammar into a pushdown system and vice versa in a way that is faithful wrt. ω-languages.

One might ask why we do not allow intermediary infiniteness. This would in fact decrease the expressiveness of our model. We elaborate on this in section A.

The remainder of this section is dedicated to proving the proposition. The implication from left to right is immediate. For the reverse implication, we observe that the right-infinite derivation processes of a CFG can be understood as infinite paths in the ω-graph, a finite graph associated with the CFG. From this finiteness, we can derive the structure required for an ω-context-free language. Technically, the *ω-graph* of G is a directed graph with edges labeled by sentential forms. There is one vertex for each non-terminal. Moreover, for each production rule $X \to \alpha Y$ there is an edge from X to Y labeled by α. For the grammar G_{ex} in our running example, the ω-graph is depicted to the right. The correspondence of the derivation processes and the paths is immediate.

Since the ω-graph is finite, every infinite path from S visits some vertex X infinitely often. We use this observation to decompose the infinite path into a finite path from S to X and an infinite sequence of cycles in X. This proves the next lemma. In the statement, $\mathcal{P}(Y, Z)$ is the set of words obtained as labels of paths from Y to Z in the ω-graph of G.

▶ **Lemma 3.** *Let G be a CFG, then*

$$\mathcal{L}^\omega(G) \;=\; \bigcup_{X \in N} \; \Big(\bigcup_{p \in \mathcal{P}(S,X)} \mathcal{L}(p)\Big)\Big(\bigcup_{c \in \mathcal{P}(X,X)} \mathcal{L}(c) \setminus \{\varepsilon\}\Big)^\omega \;.$$

The lemma does not yet give us the desired representation for $\mathcal{L}^\omega(G)$: The inner unions are not finite in general, and therefore it is not clear that they define context-free languages. The following lemma states that this is the case. It concludes the proof of Proposition 2.

▶ **Lemma 4.** *$\bigcup_{p \in \mathcal{P}(X,Y)} \mathcal{L}(p)$ is context free for all non-terminals X, Y.*

3 Liveness Verification

The liveness verification problem takes as input a context-free grammar G and a Büchi automaton A and checks whether $\mathcal{L}^\omega(G) \subseteq \mathcal{L}^\omega(A)$ holds. In the setting where G is a Büchi automaton, recent works [14, 1, 2] have proposed so-called lasso-finding algorithms as efficient means for checking the inclusion. Our contribution is a generalization of lasso finding to the ω-context-free case (modeling recursive rather than finite state programs).

A *non-deterministic Büchi automaton (NBA)* is a tuple $A = (T, Q, q_{init}, Q_F, \to)$, where T is a finite alphabet, Q is a finite set of states, $q_{init} \in Q$ is the initial state, $Q_F \subseteq Q$ is the set of final states, and $\to \subseteq Q \times T \times Q$ is the transition relation. We write $q \xrightarrow{a} q'$ for $(q, a, q') \in \to$ 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. Furthermore, we write $q \xrightarrow{w}_f q'$ if $q \xrightarrow{w} q'$ and at least one of the states in the sequence is final. The language of infinite words $\mathcal{L}^\omega(A)$ consists of all words $w \in T^\omega$ such that there is an infinite sequence of states labeled by w in which infinitely many final states occur. From now on, we use $A = (T, Q, q_{init}, Q_F, \to)$ for Büchi automata and $G = (N, T, P, S)$ for grammars. Note that both use the terminal symbols T.

Key to our generalization are *procedure summaries*. A procedure summary captures the changes that a procedure call may induce on the global state. In our setting, procedures correspond to non-terminals X, evaluated procedure calls to terminal words derivable from X, and the global state is reflected by the states of A. Hence, for every terminal word w derivable from X we should summarize the effect of w on A. This effect are the state changes that the word may induce on the automaton. For the set of all terminal words derivable from X, we thus compute the corresponding set of state changes.

We formalize procedure summaries as elements of the transition monoid [24]. The *transition monoid* of A is the monoid $\mathrm{M}(A) := (\mathrm{B}(A) \cup \{\mathrm{id}\}, ;, \mathrm{id})$. The state changes on A are captured by so-called *boxes*, labeled relations over the states. The label is a flag that will be used to indicate whether the words giving rise to the relation may pass through a final state when being processed:

$$\mathrm{B}(A) := \{\rho \in \mathcal{P}(Q \times Q \times \mathbb{B}) \mid \forall q, q' \in Q : \ (q, q', 1), (q, q', 0) \text{ not both in } \rho\} \ .$$

The additional element id is the neutral element of the monoid and will play a particular role when assigning languages to boxes. The composition of boxes ; is a relational composition that remembers visits to final states. Formally, given $\rho, \tau \in \mathrm{B}(A)$, we define:

$$\begin{aligned} \rho;\tau \quad := \quad & \{(q, q', 1) \mid \quad \exists q'' : (q, q'', x) \in \rho, \ (q'', q', y) \in \tau, \ \max(x, y) = 1\} \\ \cup \quad & \{(q, q', 0) \mid \quad \exists q'' : (q, q'', 0) \in \rho, \ (q'', q', 0) \in \tau, \\ & \qquad\quad \nexists q^* : (q, q^*, x) \in \rho, \ (q^*, q', y) \in \tau, \ \max(x, y) = 1\} \ . \end{aligned}$$

Since id is the neutral element, we have $\mathrm{id};\rho = \rho;\mathrm{id} = \rho$ for all $\rho \in \mathrm{M}(A)$.

To use boxes for checking inclusion, we have to retrieve the words represented by a box. A box represents the set of all words that yield the prescribed effect. We assign to $\rho \in \mathrm{B}(A)$ the language $\mathcal{L}(\rho)$ of all words $u \in T^+$ that satisfy $q \xrightarrow{u} q'$ for all $(q, q', *) \in \rho$ and $q \xrightarrow{u}_f q'$ for all $(q, q', 1) \in \rho$. There is a u-labeled path from q to q' iff the box contains a corresponding triple (where the label is not important). Moreover, one of the u-labeled paths can visit a final state iff this is required.

$$\mathcal{L}(\rho) = \left\{ u \in T^+ \;\middle|\; \text{for all } (q, q', *) \in \rho\colon q \xrightarrow{u} q' \text{ and for all } (q, q', 1) \in \rho\colon q \xrightarrow{u}_f q' \right\}$$

To the element id we assign the singleton language $\mathcal{L}(\mathrm{id}) := \{\varepsilon\}$. The empty word cannot be lifted to an infinite word through ω-iteration, and therefore has to be handled with care. We use function $\rho : T^* \to \mathrm{M}(A)$ to abstract a word $w \in T^*$ to the unique box ρ_w representing it in the sense that $w \in \mathcal{L}(\rho_w)$. Note that $\rho_{uv} = \rho_u;\rho_v$. This means the boxes with a non-empty language can be computed from the boxes of the letters ρ_a, where ρ_a contains (q, q', i) iff there is an a-labeled edge from q to q'. We have $i = 1$ iff q or q' is final. In particular, the image ρ_{T^*} is precisely the set of boxes ρ with $\mathcal{L}(\rho) \neq \emptyset$. Figure 1 illustrates the representation of words of Büchi automaton A_{ex} by boxes.

The representation of finite words by boxes can be lifted to infinite words. We recall two results that date back to [28]. The first result states that every infinite word is contained in a composition $\mathcal{L}(\tau).\mathcal{L}(\rho)^\omega$ of the languages of only two boxes. The proof uses Ramsey's theorem in a way similar to Theorem 10, and indeed inspired our result. The second result

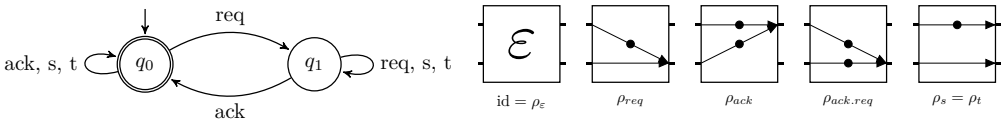


Figure 1 The automaton A_{ex} and its boxes. The upper dash on each side of a box represents state q_0, the lower dash represents q_1. A dot on the dash marks that a final state has been visited.

states that a language $\mathcal{L}(\tau).\mathcal{L}(\rho)^\omega$ is either contained in $\mathcal{L}^\omega(A)$ or it is disjoint from $\mathcal{L}^\omega(A)$. It follows from the definition of box languages. Together, one can understand the set of languages $\mathcal{L}(\tau).\mathcal{L}(\rho)^\omega$ as a finite abstraction of T^ω that is precise enough wrt. inclusion in $\mathcal{L}(A)$. We refer to the languages $\mathcal{L}(\tau).\mathcal{L}(\rho)^\omega$ included in $\mathcal{L}(A)$ as the *cover* of $\mathcal{L}(A)$.

▶ **Lemma 5.** *(1) For every $w \in T^\omega$ there are $\tau, \rho \in \mathrm{B}(A)$ with $w \in \mathcal{L}(\tau).\mathcal{L}(\rho)^\omega$.*
(2) Let $\rho, \tau \in \mathrm{B}(A)$. We have $\mathcal{L}(\tau).\mathcal{L}(\rho)^\omega \subseteq \mathcal{L}^\omega(A)$ or $\mathcal{L}(\tau).\mathcal{L}(\rho)^\omega \subseteq \overline{\mathcal{L}^\omega(A)}$.

We compute the set of boxes summarizing the effect of the words derivable from a non-terminal as the least solution to a system of inequalities. The system is interpreted over the complete lattice $(\mathcal{P}(\mathrm{M}(A)), \subseteq)$. For two sets of boxes $S, R \subseteq \mathrm{M}(A)$, we define their composition $S;R = \{\tau;\rho \mid \tau \in S, \rho \in R\}$ to be the set of all pairwise compositions.

There are two types of variables, Λ_X and $\Delta_{X,Y}$ for all non-terminals X and Y. The solution to a variable Λ_X will contain the boxes for the words derivable from X. The task of $\Delta_{X,Y}$ is to additionally remember the rightmost non-terminal. This means we compute the boxes of all words w with $X \Rightarrow^* wY$. The point is that the lasso-finding test has to match successive non-terminals Y in the right-infinite derivation.

The inequalities for the variables Λ_X are as follows. For every rule $X \to \alpha$, we require $\Lambda_X \geq \Lambda_\alpha$. Here, we generalize the notation Λ from non-terminals to sentential forms by setting $\Lambda_\varepsilon := \{\mathrm{id}\}$, $\Lambda_a := \{\rho_a\}$, and $\Lambda_{\alpha\beta} := \Lambda_\alpha;\Lambda_\beta$. For the second set of variables, there is a base case. For every non-terminal Y, we require $\Delta_{Y,Y} \geq \{\mathrm{id}\}$. The empty word takes Y to Y and is represented by id. For every pair of non-terminals X, Y and for every edge (X, α, Z) in the ω-graph, we have the inequality

$$\Delta_{X,Y} \geq \Lambda_\alpha;\Delta_{Z,Y} \ .$$

To understand the requirement, assume $Z = Y$ and thus $\Delta_{Y,Y} \geq \{\mathrm{id}\}$. Then the solution to $\Delta_{X,Y}$ indeed contains the boxes of the words w with $X \Rightarrow \alpha Y \Rightarrow^* wY$. If $Z \neq Y$, we compose the solution to Λ_α with further boxes found on the way from Z to Y.

The least solution to the above system of inequalities is computed as the least fixed point of the function on the product domain induced by the right-hand sides. A standard Kleene iteration [9] and more efficient methods like chaotic iteration [26] apply. We use σ_X and $\sigma_{X,Y}$ to denote the least solution to Λ_X and $\Delta_{X,Y}$, respectively. Again, we generalize the notation to sentential forms, σ_α.

▶ **Example 6.** In our running example, the system of inequalities (for G_{ex} and A_{ex}) is

$$\begin{array}{llll} \Lambda_X \geq \{\rho_{req}\};\Lambda_Y;\{\rho_{ack}\} & \Lambda_Y \geq \{\rho_s\};\Lambda_Y;\{\rho_t\} & \Delta_{X,X} \geq \Lambda_X;\Delta_{X,X} & \Delta_{X,X} \geq \{\mathrm{id}\} \\ \Lambda_X \geq \Lambda_X;\Lambda_X & \Lambda_Y \geq \{\mathrm{id}\} & \Delta_{X,Y} \geq \Lambda_X;\Delta_{X,Y} & \Delta_{Y,Y} \geq \{\mathrm{id}\} \end{array}$$

The least solution is $\sigma_X = \{\rho_{ack}\}$, $\sigma_Y = \{\mathrm{id}, \rho_s\}$, $\sigma_{X,X} = \{\mathrm{id}, \rho_{ack}\}$, $\sigma_{Y,Y} = \{\mathrm{id}\}$, $\sigma_{X,Y} = \emptyset$, and $\sigma_{Y,X} = \emptyset$.

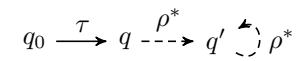
The following lemma states the indicated correspondence between the solution and the words in the language.

▶ **Lemma 7.** *$\sigma_X = \rho_{\mathcal{L}(X)}$ and $\sigma_{X,Y} = \rho_{\mathcal{L}(\mathcal{P}(X,Y))} = \{\rho_w \mid w \in \mathcal{L}(p), p \in \mathcal{P}(X,Y)\}$. In particular, all occurring boxes have a non-empty equivalence class.*

With the semantical results at hand, we can develop our lasso-finding algorithm. Lassos, a notion proposed in [14], denote elements $\mathcal{L}(\tau).\mathcal{L}(\rho)^\omega$ in the cover of $\mathcal{L}(A)$ (see the discussion before Lemma 5). Intuitively, a pair of boxes (τ, ρ) forms a lasso if box ρ, when seen as a graph with the set of states as its vertex set, contains a strongly connected component that is accepting (contains a final state) and that is reachable from the first box.

▶ **Definition 8.** A pair $(\tau, \rho) \in \mathrm{M}(A) \times \mathrm{M}(A)$ is a *lasso*, if either $\rho = \mathrm{id}$ holds or there are states $q, q' \in Q$, a transition $(q_0, q, x) \in \tau$, a path from q to q' in ρ, and an accepting path from q' to q' (a loop) in ρ.

The definition is illustrated to the right. With the aforementioned graph-theoretic interpretation of lassos, it can be checked in linear time whether a pair of boxes actually forms a lasso.



Lassos characterize the cover in the following sense.

▶ **Lemma 9.** *Let $\rho, \tau \in \mathrm{M}(A)$ with $\mathcal{L}(\tau)\mathcal{L}(\rho)^\omega \neq \emptyset$. Then $\mathcal{L}(\tau)\mathcal{L}(\rho)^\omega \subseteq \mathcal{L}^\omega(A)$ holds if and only if (τ, ρ) is a lasso.*

Note that if $\rho = \mathrm{id}$ then $\mathcal{L}(\tau)\mathcal{L}(\rho)^\omega = \emptyset$. Hence, we can assume that the first case in the definition of lassos does not apply. It is routine to check the correspondence.

Let us consider A_{ex} again and choose $\tau = \rho_{req} = \{(q_0, q_1, 1), (q_1, q_1, 0)\}$ and $\rho = \rho_s = \{(q_0, q_0, 1), (q_1, q_1, 0)\}$. The only transition in τ starting from initial state q_0 is $(q_0, q_1, 1)$ and the only accepting loop in ρ is $(q_0, q_0, 1)$. However, there is no path from q_1 to q_0 in ρ. Thus, (τ, ρ) is not a lasso and $\mathcal{L}(\tau)\mathcal{L}(\rho)^\omega \not\subseteq \mathcal{L}^\omega(A_{ex})$.

▶ **Theorem 10.** *The inclusion $\mathcal{L}^\omega(G) \subseteq \mathcal{L}^\omega(A)$ holds if and only if for every non-terminal $X \in N$, for every box τ in $\sigma_{S,X}$ and for every box ρ in $\sigma_{X,X}$, the pair (τ, ρ) is a lasso.*

One may check that using the grammar G_{ex} from our running example and the automaton A_{ex} from Figure 1, the condition is fulfilled and inclusion holds, i.e. $\mathcal{L}^\omega(G_{ex}) \subseteq \mathcal{L}^\omega(A_{ex})$.

4 Liveness Synthesis

Two player games with perfect information form the theory behind synthesis problems. In this section, we generalize a recent algorithm for solving context-free games with regular inclusion as the winning condition [16] to ω-context-free games with ω-regular winning conditions. An ω-context-free game is given as a context-free grammar $G = (N, T, P, S)$ where the non-terminals $N = N_\Box \uplus N_\bigcirc$ are partitioned into the non-terminals owned by player *prover* $\Box$ and the ones owned by player *refuter* $\bigcirc$. The winning condition is defined by a Büchi automaton A. Player $\bigcirc$ will win the game if she enforces the derivation of an infinite word not in the language of A. Player $\Box$ will win the remaining plays.

Formally, the game induces a *game arena*, a directed graph defined as follows. (1) The set of vertices is the set of all sentential forms $\vartheta = (N \uplus T)^*$. (2) A vertex is owned by the player owning the leftmost non-terminal. Terminal words are owned by refuter. (3) The edges are defined by the left-derivation relation: If $\alpha = wX\beta$ with $\beta \neq \varepsilon$, then $\alpha \to \gamma$ in the game arena if $\alpha \Rightarrow \gamma$ by replacing X. If $\alpha = wX$, i.e. X is the leftmost and only non-terminal, then $\alpha \to \gamma$ if $\alpha \Rightarrow \gamma$ by a left-derivation using a rule $X \to \eta Y$ having a rightmost non-terminal. A *(maximal) play* of the game is a path in the game arena that is either infinite or ends in

a deadlock, i.e. in a vertex that has no successor. We think of the moves originating from vertices owned by $\Box$ resp. $\bigcirc$ as chosen by prover resp. refuter.

The goal of refuter is to derive an infinite word outside $\mathcal{L}^\omega(A)$, we also say that refuter plays a non-inclusion game. We define the infinite word derived by a play as the limit of the sequence of terminal prefixes. Given a sentential form $\alpha = wX\beta \in \vartheta$, we define its *terminal prefix* to be $w \in T^*$. An infinite play $p = \alpha_0, \alpha_1, \ldots$ of the game induces an infinite sequence of such prefixes $w_0 = \text{prefix}(\alpha_0), w_1 = \text{prefix}(\alpha_1), \ldots$, where each w_j itself is a prefix of w_{j+1}. Assume the words in the sequence of prefixes grow unboundedly, i.e. for any $i \in \mathbb{N}$, there is j such that $|w_j| > i$. The *limit of the prefixes* of p is the infinite word $\lim \text{prefix}(p)$ defined by $\big(\lim \text{prefix}(p)\big)_i = (w_j)_i$, where w_j with $|w_j| > i$ is an arbitrary terminal prefix.

An infinite play p is winning with respect to the *non-inclusion winning condition* if (1) the prefixes of the positions in p grow unboundedly, and (2) $\lim \text{prefix}(p) \notin \mathcal{L}^\omega(A)$, and (3) positions of shape wX occur infinitely often in p. Otherwise p is winning with respect to the *inclusion winning condition*. This is in particular the case if p is finite but maximal. Condition (1) enforces that $\lim \text{prefix}(p)$ is a well-defined infinite word. Condition (3) guarantees that it stems from a right-infinite derivation process.

Our goal is to develop an algorithm that, given a grammar and a Büchi automaton, decides whether refuter can win non-inclusion from the initial position S. Our overall strategy, following [29], is to reduce the problem to a finite parity game. The observation behind our reduction is the following. Each play that wins non-inclusion contains infinitely many positions of shape wX. We can therefore split the play into infinitely many parts of finite length, each starting with a position of shape wX. In a first step, we compute for every X a description of all plays from X to sentential forms of the shape uY. In a second step, we combine the information on the finite parts into a finite parity game.

Lifting the characterization of finite plays computed in the first step to the infinite plays under study is non-trivial. Our approach is to determinize the given non-deterministic Büchi automaton into a deterministic parity automaton. A *deterministic parity automaton (DPA)* is a tuple $(Q, T, q_{init}, \to, \Omega)$, where Q is a finite set of states, $q_{init} \in Q$ is the initial state, and $\to : Q \times T \to Q$ is the transition function. Rather than final states, $\Omega : Q \to \mathbb{N}$ assigns a priority $i \in \mathbb{N}$ to each state. We extend the transition function to words and augment it by the highest occurring priority: We write $q \xrightarrow{w}_i q'$ if processing w starting in q leads to state q' and the highest priority of q, q', and any intermediary state is i. The language $\mathcal{L}^\omega(A_P)$ consists of all words $w \in T^\omega$ such that the highest priority occurring infinitely often on the states in the run of A_P on w is even.

Non-deterministic Büchi automata can be converted to deterministic Rabin automata [23], which in turn can be transformed to deterministic parity automata, see e.g. [21].

▶ **Theorem 11** ([23, 21]). *For an NBA A with n states, one can construct a DPA A_P with at most $2^{\mathcal{O}(n \log n)}$ states and maximal priority $\leq 2n + 2$ so that $\mathcal{L}^\omega(A) = \mathcal{L}^\omega(A_P)$.*

From now on, we will work with the computed DPA $A_P = (Q, T, q_{init}, \to, \Omega)$.

4.1 From Context-Free Games to Formulas

Our goal is to employ the characterization of inclusion games over finite words developed in [16]. Semantically, the characterization is given as a positive Boolean formula over a finite set of atomic propositions. The formula captures the tree of all plays starting in a non-terminal by interpreting refuter positions as disjunctions, prover positions as conjunctions, and terminal words as atomic propositions. Algorithmically, the formulas for all non-terminals are computed as the least solution to a system of equations.

In the current setting, (1) we have to track the priorities obtained when processing a terminal word and (2) we are given a deterministic rather than a non-deterministic automaton. To reflect (1), we will consider as atomic propositions pairs (p, i) consisting of a state $p \in Q$ and a priority $i \in \Omega(Q)$. Using (2), we define a system of equations with variables Δ_{qX} for each state $q \in Q$ and each non-terminal $X \in N$. Intuitively, in the formula for Δ_{qX} atomic propositions (p, i) represent terminal words w such that $X \Rightarrow^* w$ and $q \xrightarrow{w}_i p$. In the following, we define the domain and then set up the system of equations.

Let $\mathrm{pBF}(Q \times \Omega(Q))$ be the set of positive Boolean formulas over atomic propositions consisting of a state and a priority. We will assume that the unsatisfiable formula *false* is also contained in $\mathrm{pBF}(Q \times \Omega(Q))$. Conjunction $\wedge$ and disjunction $\vee$ are defined as usual. To simplify the technical development, we evaluate operations involving *false* on a syntactical level by using the rules $F \vee \mathit{false} = \mathit{false} \vee F = F$ and $F \wedge \mathit{false} = \mathit{false} \wedge F = \mathit{false}$.

Assume F represents the plays from state q and non-terminal X, and for each state q' the formula $G_{q'}$ represents the plays from q' and Y. To obtain the formula representing the plays from q and the sentential form XY, we can combine F and the family $(G_{q'})_{q' \in Q}$: A play from XY to a terminal word can be decomposed into a play from XY to wY, and a play from wY to wv. The first part has the same structure as a play from X to w, and the second part is essentially a play from Y to v with w prepended. We think of each atomic proposition (p, i) in F as describing the behavior of a word w, i.e. $q \xrightarrow{w}_i p$. We obtain the formula imitating this behavior for XY by replacing each atomic proposition (p, i) in F by the formula G_p that describes the effect of Y from p on. To reflect that the highest priority seen while processing wv is the maximum of the priorities seen while processing w and v, we will have to modify the priorities occurring in G_p.

We formalize the above discussion in the definitions of the composition operator ; on formulas and the operator : that composes one formula with a family of formulas. Here and in the rest of the paper, we assume that $F, F', G, G' \in \mathrm{pBF}(Q \times \Omega(Q))$ are formulas, and $(G_q)_{q \in Q}$ and $(H_q)_{q \in Q}$ are families of formulas. Furthermore, $(p, i), (p', i') \in Q \times \Omega(Q)$ are atomic propositions and $* \in \{\wedge, \vee\}$:

$$(F * F') : (G_q)_{q \in Q} = F : (G_q)_{q \in Q} * F' : (G_q)_{q \in Q} \qquad (p, i) : (G_q)_{q \in Q} = (p, i); G_p$$
$$(p, i); (G * G') = (p, i); G * (p, i); G' \qquad (p, i); (p', i') = (p', max\{i, i'\}) \ .$$

Also here, we handle *false* on a syntactic level by defining $F; \mathit{false} = \mathit{false}; G = \mathit{false}$ and $\mathit{false} : (G_q)_{q \in Q} = \mathit{false}$. The case $(F * F'); (p, i)$ does not occur.

We will also need to represent the terminal symbols and ε. Given a state q and $a \in T$, qa is the formula formed by the atomic proposition (p, i), where $q \xrightarrow{a}_i p$ and $i = \max\{\Omega(q), \Omega(p)\}$. To handle ε, we define $q\varepsilon$ to be $(q, 0)$. One might expect the second component to be $\Omega(q)$, but setting it to 0 makes the case ε^ω (which is not an infinite word) simpler.

To guarantee that a system of equations interpreted over $\mathrm{pBF}(Q \times \Omega(Q))$ has a unique least solution, we need a partial order on the domain. It has to have a least element and the operations have to be monotonic. Since we deal with Boolean formulas, implication $\Rightarrow$ is the obvious choice for the order. Unfortunately, it is not antisymmetric, which we will tackle in a moment. The least element is *false*. Monotonicity is the following lemma.

▶ **Lemma 12.** *The compositions ; and : are monotonic: If $F \Rightarrow F'$, $G \Rightarrow G'$, and for each $q \in Q$, $G_q \Rightarrow G'_q$, then $F; G \Rightarrow F'; G'$ and $F : (G_q)_{q \in Q} \Rightarrow F' : (G'_q)_{q \in Q}$.*

For the solution to be computable, we have to operate on a finite domain. Since we deal with formulas ordered by implication, we can factor them by logical equivalence. This yields a finite domain and takes care of the missing antisymmetry. The order and all operations will be

adapted to the domain $\mathrm{pBF}(Q_{A_P} \times P)_{/\Leftrightarrow}$ by applying them to arbitrary representatives. This makes $\Rightarrow$ a partial order on the equivalence classes, and all other operations are well-defined since they were monotonic with respect to implication.

We are now ready to define the system of equations induced by G and A_P. To simplify the notation, we will define $\Delta_{qa} = qa$ for $a \in T \cup \{\varepsilon\}$. We extend this to sentential forms by using composition: $\Delta_{q\alpha\beta} = \Delta_{q\alpha} : (\Delta_{p\beta})_{p\in Q}$. The following lemma states that this is well-defined and not dependent on the splitting of $\alpha\beta$.

▶ **Lemma 13.** *The composition of families is associative in the following sense:*

$$\Big(F : (G_q)_{q\in Q}\Big) : (H_p)_{p\in Q} = F : \Big(G_q : (H_p)_{p\in Q}\Big)_{q\in Q} \ .$$

For each non-terminal $X \in$ and each state $q \in Q$, we have one defining equation

$$\Delta_{qX} = \begin{cases} \bigwedge_{X\to\eta} \Delta_{q\eta} \ , & X \in N_\Box \ , \\ \bigvee_{X\to\eta} \Delta_{q\eta} \ , & X \in N_\bigcirc \ . \end{cases}$$

The resulting system of equations is solved by a standard fixed-point iteration, starting with the equivalence class of *false* for each component. We define σ_{qX} to be the value of Δ_{qX} in the least solution, and we extend this to sentential forms as above: $\sigma_{qa} = qa$ for $a \in T \cup \{\varepsilon\}$, $\sigma_{q\alpha\beta} = \sigma_{q\alpha} : (\sigma_{p\beta})_{p\in Q}$. To show that the formula $\sigma_{q\alpha}$ indeed describes the behavior of all finite plays from α, we construct strategies that are guided by the formula.

Strategies. We fix for each equivalence class of formulas $\sigma_{q\alpha}$ a representative in conjunctive normal form (CNF). (We prove that the development is independent from the choice of the representative.) The conjunctions correspond to the choices of prover during the play. The choices of refuter correspond to selecting one atomic proposition per clause. We formalize the selection process using the notion of choice functions. A *choice function* on a formula F is a function $c : F \to Q \times \Omega(Q)$ selecting an atomic proposition 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 having one of the chosen effects on the automaton. In particular, the strategy will only generate finite plays.

▶ **Proposition 14.** *(1) Let K be a clause of $\sigma_{q\alpha}$. There is a strategy s_K for prover such that all maximal plays starting in α that conform to s_K are either infinite or end in a terminal word w such that $q \xrightarrow{w}_i q'$ and $(q', i) \in K$. (2) Let c be a choice function on $\sigma_{q\alpha}$. There is a strategy s_c for refuter such that all maximal plays starting in α that conform to s_c end in a terminal word w with $q \xrightarrow{w}_i q'$ and $(q, i) \in c(\sigma_{q\alpha})$.*

The proof of Part (2) is a deterministic version of the analogue result in [16]. Since we do not have to guarantee termination of the play, Part (1) it is simpler.

Towards Infinite Games. The solution of the system of equations characterizes for each non-terminal X the terminal words w that can be derived from X. For the infinite game, we have to characterize the sentential forms wY that can be derived from X. Since there may be several different non-terminals Y such that a sentential form wY is reachable from X, we store the target non-terminal in the atomic propositions. For $F \in \mathrm{pBF}(Q \times \Omega(Q))$ and a non-terminal Y, we define $F.Y$ to be the formula in $\mathrm{pBF}(Q \times \Omega(Q) \times N)$ that is obtained by adding Y as a third component in every atomic proposition. With $* \in \{\vee, \wedge\}$, we set

$$(F * F').Y = F.Y * F'.Y \ , \qquad (q, i).Y = (q, i, Y) \ .$$

For each non-terminal X, we collect all rules $X \to \eta Y$ with a non-terminal Y as their rightmost symbol. We represent the behavior of η by the previously computed formulas $\sigma_{q\eta}$ and attach Y as described above. The resulting formulas are combined using disjunction or conjunction, depending on the owner of X. Given a non-terminal X and a state $q \in Q$, we define the extended solution for qX to be

$$\sigma^e_{qX} = \begin{cases} \bigwedge_{X \to \eta Y} \sigma_{q\eta}.Y, & X \in N_\Box, \\ \bigvee_{X \to \eta Y} \sigma_{q\eta}.Y, & X \in N_\bigcirc, \end{cases}$$

If no rule of the shape $X \to \eta Y$ exists, the formula is *false*. The latter is to model that prover wins in this case, independently of who owns X.

▶ **Proposition 15.** *(1) Let K be a clause of σ^e_{qX}. There is a strategy s^e_K for prover such that all maximal plays starting in X that conform to s^e_K are either infinite without visiting a sentential form of shape wY, or they visit a sentential form of shape wY such that $q \xrightarrow{w}_i q'$ and $(q', i, Y) \in K$. (2) Let c be a choice function on σ^e_{qX}. There is a strategy s^e_c for refuter such that all maximal plays starting in X that conform to s^e_c visit a sentential form of shape wY such that $q \xrightarrow{w}_i q'$ and $(q, i, Y) \in c(\sigma^e_{qX})$.*

4.2 From Formulas to a Parity Game

It remains to combine the formulas for finite plays to obtain a characterization of the infinite plays. We model the infinite plays as an infinite sequence of alternations: First, prover chooses a clause from the formula for X, which fixes her strategy for the following finite part. Second, refuter chooses an atomic proposition from the selected clause, which fixes the derived sentential form wY. Instead of storing the (unboundedly growing) prefixes w explicitly, we only store the target non-terminal, the state transition of A_P while processing w, and the highest priority occurring during the transition. Modeling the game like this leads to a parity game on a finite graph.

A *parity game* $\mathcal{P} = (V = V_\Box \cup V_\bigcirc, E, \Omega)$ is a directed graph with an ownership partitioning of the vertices and a function $\Omega : V \to \mathbb{N}$ that assigns to each vertex a priority. We will assume that the parity game is deadlock-free. A maximal play is an infinite path in the graph. It is won by player $\Box$ if the highest priority occurring infinitely often on the vertices in the play is even; won by player $\bigcirc$ otherwise.

▶ **Theorem 16** (Positional Determinacy of Parity Games, [30])**.** *Given a parity game $\mathcal{P}$, there is a decomposition of the vertices $V = W_\Box \cup W_\bigcirc$ and there are positional strategies $s_\Box : V_\Box \to V$, $s_\bigcirc : V_\bigcirc \to V$ such that $s_\Box$ is winning from all positions in $W_\Box$ and $s_\bigcirc$ is winning from all positions in $W_\bigcirc$.*

▶ **Definition 17.** The parity game $\mathcal{P}_{G,A_P}$ induced by the context-free grammar G and the DPA A_P is $(V = V_\Box \cup V_\bigcirc, E, \Omega)$. The vertices $V_\Box = \{qX \mid q \in Q, X \in N\}$ represent the formulas. They are owned by prover because prover is allowed to pick a clause. The vertices of refuter $V_\bigcirc = V_\bigcirc^{clause} \cup V_\bigcirc^{helper}$ are of two types. Since refuter should select an atomic proposition, she owns the vertices $V_\bigcirc^{clause} = \left\{qXK \;\middle|\; q \in Q, X \in N, K \in \sigma^e_{qX}\right\}$ representing the clauses. The helper vertices $V_\bigcirc^{helper} = \left\{(qXK, i, pY) \;\middle|\; q \in Q, X \in N, K \in \sigma^e_{qX}, (p, i, Y) \in K\right\}$ will be used to keep track of the priority that is seen while processing the terminal prefix w that is created by going from X to wY. The edges connect non-terminals to clauses, and clauses to the next formula via the helper vertices:

$$E = \big\{\ (qX, qXK) \bigm| q \in Q, X \in N, K \in \sigma^e_{qX} \big\}$$
$$\cup \big\{\ (qXK, (qXK, i, pY)),\ ((qXK, i, pY), pY) \bigm| q \in Q, X \in N, K \in \sigma^e_{qX}, (p, i, Y) \in K \big\}$$
$$\cup \big\{\ (qXK, qXK) \bigm| q \in Q, X \in N, K \in \sigma^e_{qX}, K = \emptyset \big\}.$$

The last part takes care of the empty clause which occurs iff the formula is equivalent to *false*. The priority function is zero but on the helper vertices, where it returns the priority given by the selected atomic proposition: $\Omega(qX) = \Omega(qXK) = 0$, $\Omega((qXK, i, pY)) = i$.

We are now able to state the correspondence between the ω-context-free game of interest and the constructed parity game.

▶ **Theorem 18** (Determinacy of ω-Context-Free Games)**.** *Prover resp. refuter has a winning strategy for the ω-regular inclusion game from S iff she wins the parity game from $q_0 S$.*

Proof Sketch. Using Theorem 16 on the positional determinacy of parity games, exactly one of the players wins the parity game from $q_0 S$, and she has a positional winning strategy. We use this positional winning strategy to construct a winning strategy for the ω-context-free game. To this end, we establish a correspondence between the play of the parity game and the run of A_P on an infinite word derived in the ω-context-free grammar by following the play. The key idea is that a winning strategy for the parity game for prover resp. refuter fixes clauses resp. choice functions. Using these clauses resp. choice functions, we can apply Proposition 15 to obtain a strategy for the finite part of the ω-context-free game that is played until the next sentential form represented in the parity game (by a vertex) is found. We make this precise in Section C.2. ◀

4.3 Complexity

We show that deciding whether refuter has a winning strategy for ω-regular non-inclusion from position S is a 2EXPTIME-complete problem. Moreover, the algorithm presented in this section achieves this optimal time complexity.

Our proof of the lower bound works by showing that the case of finite inclusion games can be seen a special case of the problem under consideration here. Solving finite context-free games has been shown to be a 2EXPTIME-complete problem in [20].

▶ **Theorem 19.** *Solving ω-context-free games is* 2EXPTIME*-hard.*

We summarize the algorithm outlined in this section: (1) Construct the deterministic parity automaton A_P. (2) Construct and solve the system of equations. (3) Extend the solution σ to obtain σ^e. (4) Construct the finite parity game $\mathcal{P}_G$. (5) Check which player wins $\mathcal{P}_G$ from $q_0 S$.

▶ **Theorem 20.** *Given an ω-context-free game and an initial position, the algorithm outlined above decides which player wins in time $\mathcal{O}\Big(2^{2^{|Q|^{c_1}}} \cdot 2^{|G|^{c_2}}\Big)$ for some constants $c_1, c_2 \in \mathbb{N}$.*

References

1 P. A. Abdulla, Y.-F. 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 6471 of *LNCS*, pages 132–147. Springer, 2010.

2 P. A. Abdulla, Y.-F. 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.

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

4 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.

5 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.

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

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

8 R. S. Cohen and A. Y. Gold. Theory of ω-languages (i and ii). *JCSS*, 15(2):169–184, 185–208, 1977.

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

10 A. Farzan, Z. Kincaid, and A. Podelski. Proof spaces for unbounded parallelism. In *POPL*, pages 407–420. ACM, 2015.

11 A. Farzan, Z. Kincaid, and A. Podelski. Proving liveness of parameterized programs. In *LICS*. IEEE, 2016.

12 Azadeh Farzan, Zachary Kincaid, and Andreas Podelski. Proofs that count. In *POPL*, pages 151–164. ACM, 2014.

13 O. Finkel. Topological properties of omega context-free languages. *Theor. Comp. Sci.*, 262(1):669–697, 2001.

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

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

16 L. Holík, R. Meyer, and S. Muskalla. Summaries for context-free games. In *FSTTCS*, LIPIcs. Dagstuhl, 2016. To appear, `https://arxiv.org/abs/1603.07256`.

17 Lukás Holík and Roland Meyer. Antichains for the verification of recursive programs. In *NETYS*, volume 9466 of *LNCS*, pages 322–336. Springer, 2015.

18 M. Linna. On ω-sets associated with context-free languages. *Inf. Cont.*, 31(3):272–293, 1976.

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

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

21 N. Piterman. From nondeterministic B$\breve{\text{u}}$chi and Streett automata to deterministic parity automata. *Log. Meth. Comput. Sci.*, 3(3), 2007.

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

23 S. Safra. On the complexity of ω-automata. In *FOCS*, SFCS, pages 319–327. IEEE, 1988.

24 Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

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

26 H. Seidl, R. Wilhelm, and S. Hack. *Compiler Design - Analysis and Transformation*. Springer, 2012.

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

28 A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for B$\ddot{\text{u}}$chi automata with applications to temporal logic. In *ICALP*, volume 194 of *LNCS*, pages 217–237. Springer, 1985.

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

30 W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comp. Sci.*, 200(1-2):135–183, 1998.

A Details on Section 2

One direction of the proof of Proposition 2 is immediate. We prove it in the form of the following proposition.

▶ **Proposition 21.** *Let $\mathcal{L} \subseteq T^\omega$ be a ω-context-free language. Then there is a CFG G such that $\mathcal{L} = \mathcal{L}^\omega(G)$.*

Proof. We may assume

$$\mathcal{L} = \bigcup_{i=1,\dots,n} V_i U_i^\omega$$

and there are CFGs

$$G_{V_i} = (N_{V_i}, T, P_{V_i}, S_{V_i}), \qquad G_{U_i} = (N_{U_i}, T, P_{U_i}, S_{U_i}),$$

with $V_i = \mathcal{L}(G_{V_i})$ and $U_i = \mathcal{L}(G_{U_i})$ for all i such that all sets of non-terminals are pairwise disjoint. We construct a new grammar $G = (N, \Sigma, P, S)$ with $N = \{S\} \uplus \{R_i \mid i = 1, \dots, m\} \uplus \bigcup_{i=1,\dots,n} N_{U_i} \uplus \bigcup_{i=1,\dots,n} N_{V_i}$ and $P = \{S \to S_{V_i} R_i \mid i = 1, \dots, m\} \uplus \{R_i \to S_{U_i} R_i \mid i = 1, \dots, m\} \uplus \bigcup_{i=1,\dots,n} P_{U_i} \uplus \bigcup_{i=1,\dots,n} P_{V_i}$. $\mathcal{L} = \mathcal{L}^\omega(G)$ is easy to see. ◀

Intermediary Infiniteness. A grammar is supposed to generate the infinite computations of a recursive program, and a rule $X \to aYZ$ should be understood as procedure X executing action a, calling procedure Y, and after Y has returned continuing with procedure Z. Our restriction to the right-infinite derivations allows procedure Z to run forever, but for Y we only consider finite executions. The reader may argue that we should also consider the infinite executions of Y. Interestingly, our restriction to the right-infinite derivations increases the expressiveness of the language class compared to a definition that closes the ω-language under intermediary infiniteness. The alternative definition yields a subclass of the ω-context-free languages as one can always add shortened rules to a given grammar that reflect intermediary infiniteness. In the example, one would just have to add the rule $X \to aY$ to also reflect the fact that the program may do its infinite computation without returning from procedure Y. To see that the inclusion is strict, consider the language $\mathcal{L} = (a^{n_i} b^{n_i})^\omega$ with $n_i \in \mathbb{N}$ for all i. The language containing $\mathcal{L}$ would also contain $a^\omega \notin \mathcal{L}$.

Proof of Lemma 4. For non-terminals $A, B \in N$ and a set $M \subseteq N$ of non-terminals, we define $\mathcal{P}(A, B)^M$ to be the set of all finite paths from A to B in the ω-graph such that all occurring intermediary vertices are in M. We show that the corresponding language

$$\mathcal{L}\Big(\mathcal{P}(A, B)^M\Big) = \bigcup_{P \in \mathcal{P}(A,B)^M} \mathcal{L}(p)$$

is context-free by induction on the size of M. This proves that $\bigcup_{p \in \mathcal{P}(X,Y)} \mathcal{L}(p)$ is context-free since $\mathcal{P}(X, Y) = \mathcal{P}(X, Y)^N$.

Case $M = \emptyset$: All paths in $\mathcal{P}(A, B)^\emptyset$ have length at most one. If $A = B$, the corresponding language contains ϵ and all elements of the context-free languages $\mathcal{L}(\alpha)$ for all self-loops (A, α, A). If $A \neq B$, the corresponding language contains all elements of the context-free languages $\mathcal{L}(\alpha)$ for all edges (A, α, B). Since there are only finitely many of those edges, and context-free languages are closed under finite unions, the language corresponding to $\mathcal{P}(A, B)^\emptyset$ is context-free.

Case $M \neq \emptyset$: Let us first consider the special case of cycles (i.e. $A = B$) and $A \in M$. Any cycle in which A occurs as intermediary vertex can be decomposed into several cycles, such that A does not occur as intermediary vertex in any of those. We can use this to obtain the representation

$$\mathcal{L}\Big(\mathcal{P}(A,A)^M\Big) = \Big(\cup_{c\in\mathcal{P}(A,A)^{M\setminus\{A\}}} L(c)\Big)^* = \mathcal{L}\Big(\mathcal{P}(A,A)^{M\setminus\{A\}}\Big)^*$$

which is context-free by induction and since context-free languages are closed under Kleene-iteration.

In the general case, any path p from A to B has either no repeating intermediary vertex, i.e. it is simple, or there is an intermediary vertex C occurring several times. In the latter case, it can be decomposed,

$$p = p_{AC}\ c_C\ p_{CB}$$

where p_{AC} is a path from A to C, p_C a cycle in C, and a p_{CB} a path from C to B. We can assume that C does not occur as intermediary vertex in p_{AC} and p_{CB}, and as before, we can decompose $c_C = c_1 c_2 \dots c_k$ into finitely many cycles such that C does not occur in any of them as intermediary vertex.

Altogether, we retrieve the representation

$$\left(\bigcup_{\substack{p\in\mathcal{P}(A,B)^M\\ p \text{ simple}}} \mathcal{L}(p)\right) \cup \left(\bigcup_{C\in N} \mathcal{L}\Big(\mathcal{P}(A,C)^{M'}\Big)\mathcal{L}\Big(\mathcal{P}(C,C)^{M'}\Big)^*\mathcal{L}\Big(\mathcal{P}(C,B)^{M'}\Big)\right)$$

where $M' = M \setminus \{C\}$.

The first part is context-free since there are only finitely many simple paths. The second part is a finite union of concatenations of context-free languages (by induction and closure under Kleene-iteration). Altogether, this shows that $\mathcal{L}\Big(\mathcal{P}(A,B)^M\Big)$ is context-free. ◀

B Details on Section 3

Proof of Lemma 7. For the proof of $\sigma_X = \rho_{\mathcal{L}(X)}$, we refer to the proof of Lemma 3 in [17].

It remains to prove $\sigma_{X,Y} = \rho_{\mathcal{L}(\mathcal{P}(X,Y))} = \{\rho_w \mid w \in \mathcal{L}(p), p \in \mathcal{P}(X,Y)\}$. Assume there is a word in $w \in \mathcal{L}(p), p \in \mathcal{P}(X,Y)$. Let $p = \alpha_0 \dots \alpha_k$ be a decomposition of the path into its edges. By plugging in the inequalities into each other along the path, one can see that

$$\sigma_{X,Y} \geq \sigma_{\alpha_0}; \dots; \sigma_{\alpha_k}; \{\mathrm{id}\} = \sigma_{\alpha_0}; \dots; \sigma_{\alpha_k}.$$

We can write $w = w_0 \dots w_m$ such that for each i, $w_i \in \mathcal{L}(\alpha_i)$. By the first part of the Lemma, we have $\sigma_{\alpha_i} = \{\rho_w \mid w \in \mathcal{L}(\alpha_i)\}$ for each i, in particular ρ_{w_i} in σ_{α_i}. By the definition of the composition of sets of boxes and the above inequality, we then also have $\rho_w \in \sigma_{X,Y}$.

Assume there is a box ρ in $\sigma_{X,Y}$. We prove using induction that all boxes in ${\sigma_{X,Y}}^j$ have corresponding words in $\mathcal{L}(\mathcal{P}(X,Y))$, where σ^j is the intermediary solution after the j^{th}-step of Kleene iteration. Since $\sigma_{X,Y} = {\sigma_{X,Y}}^{j_0}$ for some j_0, this proves the claim. If the ρ entered the solution in the first iteration we have $\rho = \mathrm{id}$ and we are done.

If it entered the solution in step $j > 0$, then there is some edge (X, α, Z) in the ω-graph such that $\rho \in \sigma_\alpha; {\sigma_{Z,Y}}^{j-1}$, where σ^{j-1} is the solution after the $(j-1)^{th}$ iteration. There are boxes $\tau_1 \in \sigma_\alpha, \tau_2 \in {\sigma_{Z,Y}}^{j-1}$ such that $\rho = \tau_1; \tau_2$. By the first part of the theorem, there is a word w_1 such that $w_1 \in \mathcal{L}(\alpha)$ with $\rho_{w_1} = \tau_1$. By induction, there is a word w_2 such that $w_2 \in \mathcal{L}(\mathcal{P}(Z,Y))$ with $\rho_{w_2} = \tau_2$. Then $w = w_1 w_2$ is a word in $\mathcal{L}(\mathcal{P}(X,Y))$ with $\rho_w = \rho_{w_1}; \rho_{w_2} = \tau_1; \tau_2 = \rho$. ◀

Proof of Lemma 9. Note that if $\rho = \mathrm{id}$, then $\tau\rho^\omega = \emptyset$.

Assume (τ, ρ) is a lasso, then there is an edge $(q_0, q, x) \in \tau$, a path p from q to some q' in ρ and a loop c from q' to q' in ρ such that at least one edge on the loop is labeled by one. Let k be the length of p and let m be the length of c.

Assume $w \in \mathcal{L}(\tau)\mathcal{L}(\rho)^\omega$, then there is a decomposition $w = w^{(0)}w^{(1)}\dots$ with $\tau = \rho_{w^{(0)}}$ and $\rho = \rho_{w^{(1)}} = \rho_{w^{(i)}}$ for all $i > 0$. Then the following sequence can be refined to a run by inserting intermediary states:

$$q_0 \xrightarrow{w^{(0)}} q \xrightarrow{w^{(1)}\dots w^{(1+k)}} q' \xrightarrow{w^{(1+k+1)}\dots w^{(1+k+c)}} q' \xrightarrow{w^{(1+k+c+1)}\dots w^{(1+k+2\cdot c)}} \dots$$

Since at least one edge occurring in the loop is labeled by 1, one can refine the sequence to an accepting run that visits infinitely many final states. This shows $w \in L^\omega(A)$.

Let us now assume $w \in \mathcal{L}(\tau)\mathcal{L}(\rho)^\omega, w \in L^\omega(A)$, i.e. there is an accepting run of w on A. Let $w = w^{(0)}w^{(1)}...$ with $\tau = \rho_{w^{(0)}}$ and $\rho = \rho_{w^{(1)}} = \rho_{w^{(i)}}$ for all $i > 0$. We fix an arbitrary accepting run of A on w. Let $q^{(i)}$ be the state of A in this run after processing $w^{(i)}$ for each i. We define $q = q^{(0)}$ and q' to be the first state which occurs infinitely often in the sequence of the $q^{(i)}$. Let p be the path from $q^{(0)}$ to q' in ρ. There has to be an occurrence of q', say $q^{(j)}$, such that there is a final state between the first occurrence of q' and $q^{(j)}$. This proves that ρ contains a loop c from and to q' in which at least one edge is labeled by 1. The membership of the words in the languages of the boxes guarantees the existence of p and c. The transition $(q_0, q^{(0)}, *) \in \tau$, the path p and the loop c prove that (τ, ρ) is a lasso. ◀

Proof of Theorem 10. For the implication from right to left, we show that whenever the inclusion fails, there is a non-terminal X, a box $\tau \in \sigma_{S,X}$, and a box $\rho \in \sigma_{X,X}$ such that (τ, ρ) is no lasso. Consider the word $w \in \mathcal{L}^\omega(G) \setminus \mathcal{L}^\omega(A)$. By definition of $\mathcal{L}^\omega(G)$, there is a decomposition

$$w = w^{(0)}w^{(1)}w^{(2)}...$$

and an infinite sequence of rules

$$S \to \alpha_0 X_1, X_1 \to \alpha_1 X_2, \dots$$

so that $w^{(j)} \in \mathcal{L}(\alpha_j)$ for all j. Let X be a non-terminal which occurs infinitely often in the sequence of the X_i. Such an X exists as there are only finitely many non-terminals. We create a new decomposition

$$w = v^{(0)}v^{(1)}v^{(2)}...$$

such that in the sequence of rules above, $v^{(0)}$ takes us from S to X for the first time, and each $v^{(j)}$ for $j > 0$ takes us from X to X.

To this decomposition, we apply Ramsey's theorem which states the following. Every (undirected) infinite complete graph that has a finite edge coloring contains an infinite complete monochromatic subgraph. For the application, define the labeled complete graph to have vertex set $\mathbb{N}$ and coloring (for all edges $\{i, j\}$ with $i < j$):

$$c(\{i, j\}) = \rho_{v_i}; \rho_{v_{i+1}}; ...; \rho_{v_{j-1}}$$

Ramsey's theorem yields an infinite complete subgraph such that all edges have the same color. Let $S = \{s_0, s_1, ...\}$ be the vertex set of this subgraph, with $s_0 < s_1 < \dots$ This vertex set yields a new decomposition of the word:

$$w = u^{(0)}u^{(1)}u^{(2)}...$$

with

$$u^{(0)} = v^{(0)}v^{(1)}...v^{(s_0-1)} \quad \text{and} \quad u^{(i)} = v^{(s_i)}v^{(s_i+1)}...v^{(s_{i+1}-1)} \quad \text{for all } i > 0\ .$$

Word $u^{(0)}$ takes us from S to X and all other $u^{(i)}$ take us from X to X. We define $\tau = \rho_{u^{(0)}}$ and $\rho = \rho_{u^{(1)}}$. Note that since all edges have the same color, we have $\rho = \rho_{u^{(i)}}$ for all $i > 0$.

By construction, $\tau \in \sigma_{S,X}$ and $\rho \in \sigma_{X,X}$. Since $w \notin \mathcal{L}^\omega(A)$, we know that (τ, ρ) is no lasso by Lemma 9.

For the implication from left to right, assume the inclusion holds, but there are $\tau \in \sigma_{S,X}$ and $\rho \in \sigma_{X,X}$ that do not form a lasso. By Lemma 7, all boxes in $\sigma_{X,Y}$ have non-empty equivalence classes. We also know that $\rho \neq \mathrm{id}$, since otherwise we would have considered the pair a lasso by definition. Hence, there is an infinite word $w \in \mathcal{L}(\tau)\mathcal{L}(\rho)^\omega$. By Lemma 7 and 3, this word is also in $\mathcal{L}^\omega(G)$. But by Lemma 9 and Lemma 5, $w \notin \mathcal{L}^\omega(A)$, which contradicts the assumption that the inclusion holds. ◀

C Details on Section 4

C.1 Details on Subsection 4.1

Proof of Lemma 12. We prove the part about : . The proof for the monotonicity of ; is analogous. The proof proceeds in phases (1) to (3) so that the claim in each phase is proven under the assumption of the claim proven in the previous phase. Let $\{*, \bar{*}\} = \{\wedge, \vee\}$. In the following, we will use $*$ and $\bar{*}$ as syntactic parts of formulas as well as to connect statements in the proof.

(1) First, we prove the lemma for the case when $F, F' \in Q \times P$. In this case, $F = F' = (p, i)$ and thus $F : (G_q)_{q\in Q} = (p, i); G_p = F' : (G'_q)_{q\in Q}$.

(2) Next, we assume that $F' \in Q \times \Omega(Q)$ and F is an arbitrary formula. We prove the statement by induction on F.

Base case: $F \in Q \times \Omega(Q)$, hence (1) proves the statement.

Induction step: Let $F = F_1 * F_2$. Note that the Boolean formulas $(a * b) \Rightarrow c$ and $(a \Rightarrow c) \ \bar{*}\ (b \Rightarrow c)$ are equivalent, called Equivalence (i) in the following. By the Equivalence (i), we get $(F_1 \Rightarrow F') \ \bar{*}\ (F_2 \Rightarrow F')$. Therefore, by the induction hypothesis, $(F_1 : (G_q)_{q\in Q} \Rightarrow F' : (G'_q)_{q\in Q}) \ \bar{*}\ (F_2 : (G_q)_{q\in Q} \Rightarrow F' : (G'_q)_{q\in Q})$. This is by (i) equivalent to $(F_1 : (G_q)_{q\in Q} * F_2 : (G_q)_{q\in Q}) \Rightarrow F' : (G'_q)_{q\in Q}$. By the definition of :, this shows $F : (G_q)_{q\in Q} \Rightarrow F' : (G'_q)_{q\in Q}$.

(3) We assume that both F, F' are arbitrary formulas. We prove the statement using induction on the structure of F'.

Base case: $F' \in Q \times \Omega(Q)$, hence the statement is proven by (2).

Induction step: Let $F' = F'_1 * F'_2$. By the general equivalence of the Boolean formulas $a \Rightarrow (b * c)$ and $(a \Rightarrow b) * (a \Rightarrow c)$, called Equivalence (ii) in the following, we get $(F \Rightarrow F'_1) * (F \Rightarrow F'_2)$. Therefore, by the induction hypothesis, $(F : (G_q)_{q\in Q} \Rightarrow F'_1 : (G'_q)_{q\in Q}) * (F : (G_q)_{q\in Q} \Rightarrow F'_2 : (G'_q)_{q\in Q})$ holds. Again by (ii), we get $F : (G_q)_{q\in Q} \Rightarrow (F'_1 : (G'_q)_{q\in Q} * F'_2 : (G'_q)_{q\in Q})$. This is $F : (G_q)_{q\in Q} \Rightarrow F' : (G'_q)_{q\in Q}$ by definition. ◀

Proof of Lemma 13. The proof proceeds in phases (1) to (3) so that the claim in each phase is proven under the assumption of the claim proven in the previous phase.

(1) We first show that $((q',i);(q,j)) : (H_p)_{p\in Q} = (q',i);((q,j) : (H_p)_{p\in Q})$.
To this end, note that:

$$((q',i);(q,j)) : (H_p)_{p\in Q} = (q, max(i,j)); H_q \ ,$$
$$(q',i);((q,j) : (H_p)_{p\in Q}) = (q',i);((q,j);H_q) \ .$$

We use structural induction on H_q to prove this equality.
Base case: Let $H_q = (p,k)$. Then we have

$$\begin{aligned}
&(q, max(i,j)); H_q \\
&= (q, max(i,j)); (p,k) \\
&= (p, max(i,j,k)) \\
&= (q',i); (p, max(j,k)) \\
&= (q',i); ((q,j); (p,k)) \\
&= (q',i); ((q,j); H_q) \ .
\end{aligned}$$

Induction step: Let $H_q = H_1 * H_2$, with $* \in \{\vee, \wedge\}$. Thus,

$$\begin{aligned}
&(q, max(i,j)); (H_1 * H_2) \\
&= (q, max(i,j)); H_1 * (q, max(i,j)); H_2 \\
&\overset{IH}{=} (q',i); ((q,j); H_1) * (q',i); ((q,j); H_2) \\
&= (q',i); ((q,j); H_1 * (q,j); H_2) \\
&= (q',i); (q,j)(H_1 * H_2) \ .
\end{aligned}$$

(2) We show that $((q',i) : (G_q)_{q\in Q}) : (H_p)_{p\in Q} = (q',i) : (G_q : (H_p)_{p\in Q})_{q\in Q}$. Note that

$$((q',i) : (G_q)_{q\in Q}) : (H_p)_{p\in Q} = ((q',i); G_{q'}) : (H_p)_{p\in Q} \ ,$$
$$(q',i) : ((G_q)_{q\in Q} : (H_p)_{p\in Q}) = (q',i); (G_{q'} : (H_p)_{p\in Q}) \ .$$

We proceed by structural induction on $G_{q'}$.
Base case: $G_{q'} = (q,j)$ holds and thus (1) proves the claim.
Induction step: Assume $G_q = G_1 * G_2$ for $* \in \{\vee, \wedge\}$. Then we can derive that

$$\begin{aligned}
&((q',i); (G_1 * G_2)) : (H_p)_{p\in Q} \\
&= ((q',i); G_1 * (q',i); G_2)) : (H_p)_{p\in Q} \\
&= ((q',i); G_1) : (H_p)_{p\in Q} * ((q',i); G_2)) : (H_p)_{p\in Q} \\
&\overset{IH}{=} (q',i); (G_1 : (H_p)_{p\in Q}) * (q',i); (G_2 : (H_p)_{p\in Q}) \\
&= (q',i); (G_1 : (H_p)_{p\in Q} * G_2 : (H_p)_{p\in Q}) \\
&= (q',i); ((G_1 * G_2) : (H_p)_{p\in Q}) \ .
\end{aligned}$$

(3) Let now $(G_q)_{q\in Q}$ and $(H_q)_{q\in Q}$ be arbitrary families. We prove the statement by induction on the structure of F.
Base case: $F = (q',i) \in Q \times \Omega(Q)$ and (2) proves the statement.

Induction step: Assume $F = F_1 * F_2$ for $* \in \{\vee, \wedge\}$. Then, we have

$$\begin{aligned}
&(F : (G_q)_{q\in Q}) : (H_p)_{p\in Q}\\
&= ((F_1 * F_2) : (G_q)_{q\in Q}) : (H_p)_{p\in Q}\\
&= (F_1 : (G_q)_{q\in Q} * F_2 : (G_q)_{q\in Q}) : (H_p)_{p\in Q}\\
&= (F_1 : (G_q)_{q\in Q}) : (H_p)_{p\in Q} * (F_2 : (G_q)_{q\in Q}) : (H_p)_{p\in Q}\\
&\overset{IH}{=} F_1 : ((G_q)_{q\in Q} : (H_p)_{p\in Q}) * F_2 : ((G_q)_{q\in Q} : (H_p)_{p\in Q})\\
&= (F_1 * F_2) : ((G_q)_{q\in Q} : (H_p)_{p\in Q})\\
&= F : ((G_q)_{q\in Q} : (H_p)_{p\in Q}), \text{ which proves the claim.}
\end{aligned}$$

◀

Conjunctive Normal Form. A formula in CNF is a conjunction of clauses, each clause being a disjunction of atomic propositions. We use set notation and write clauses as sets of atomic propositions and formulas as sets of clauses. Identify $\mathit{true} = \{\}$ and $\mathit{false} = \{\{\}\}$.

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

▶ **Lemma 22.** *$F \Rightarrow G$ if and only if there is $j : G \to F$ so that $j(H) \subseteq H$ for all $H \in G$.*

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

Disjunctions and compositions can be transformed to CNF by applying distributivity.

▶ **Lemma 23.**

$$(1)\ F \vee G \Leftrightarrow \{K \cup H \mid K \in F, H \in G\}, \quad (2)\ F \wedge G \Leftrightarrow F \cup G\ ,$$

$$(3)\ F : (G_q)_{q\in Q} \Leftrightarrow \bigcup_{K\in F} \bigcup_{\substack{z:K\to\bigcup_{q\in Q}G_q\\(q,i)\mapsto H\in G_q}} \Big\{ \bigcup_{(p,j)\in K} (p,j); z((p,j)) \Big\}$$

Proof. (1) is immediate, (2) follows from applying distributivity. We show (3) by structural induction on F.

Base case: Let $F = (q, i)$. Then

$$\begin{aligned}
&(q,i) : (G_q)_{q\in Q}\\
&= (q,i); G_q\\
&= \{(q,i); K | K \in G_q\}\\
&= \bigcup_{\substack{z:\{(q,i)\}\to G_q\\(q,i)\mapsto H\in G_q}} \{(q,i); z(q,i)\}\ .
\end{aligned}$$

Induction step:
We need to distinguish two cases. First assume $F = F_1 \wedge F_2$. Then we have

$$\begin{aligned}
&(F_1 \wedge F_2) : (G_q)_{q\in Q} \\
&= (F_1 : (G_q)_{q\in Q}) \wedge (F_2 : (G_q)_{q\in Q}) \\
&\overset{(*)}{=} \left(\bigcup_{K_1\in F_1} \bigcup_{\substack{z_1:K_1\to\cup_{q\in Q}G_q \\ (q,i)\mapsto H\in G_q}} \left\{ \bigcup_{(p,j)\in K_1} (p,j); z_1((p,j)) \right\} \right) \\
&\quad \cup \left(\bigcup_{K_2\in F_2} \bigcup_{\substack{z_2:K_2\to\cup_{q\in Q}G_q \\ (q,i)\mapsto H\in G_q}} \left\{ \bigcup_{(p,j)\in K_2} (p,j); z_2((p,j)) \right\} \right).
\end{aligned}$$

In step $(*)$, we used the induction hypothesis and part (2) of the Lemma. We define a function

$$z : K \to \cup_{q\in Q}G_q, (q,i) \mapsto \begin{cases} z_1(q,i), \text{ if } K \in F_1 \\ z_2(q,i), \text{ else.} \end{cases}$$

Using this definition, we can rewrite the last line of the equation to

$$\bigcup_{K\in F_1\cup F_2} \bigcup_{\substack{z:K\to\cup_{q\in Q}G_q \\ (q,i)\mapsto H\in G_q}} \left\{ \bigcup_{(p,j)\in K} (p,j); z((p,j)) \right\},$$

which proves the claim.

Assume now $F = F_1 \vee F_2$. Then,

$$\begin{aligned}
&(F_1 \vee F_2) : (G_q)_{q\in Q} \\
&= (F_1 : (G_q)_{q\in Q}) \vee (F_2 : (G_q)_{q\in Q}) \\
&\overset{(*)}{=} \{K \cup K' | K \in S_1, K' \in S_2\}, \text{ with} \\
&\qquad S_1 = \bigcup_{K_1\in F_1} \bigcup_{\substack{z_1:K_1\to\cup_{q\in Q}G_q \\ (q,i)\mapsto H\in G_q}} \left\{ \bigcup_{(p,j)\in K_1} (p,j); z_1((p,j)) \right\} \\
&\qquad S_2 = \bigcup_{K_2\in F_2} \bigcup_{\substack{z_2:K_2\to\cup_{q\in Q}G_q \\ (q,i)\mapsto H\in G_q}} \left\{ \bigcup_{(p,j)\in K_2} (p,j); z_2((p,j)) \right\}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\{K \cup K' | K \in S_1, K' \in S_2\} \\
&= \bigcup_{K_1\in F_1} \bigcup_{K_2\in F_2} \bigcup_{\substack{z_1:K_1\to\cup_{q\in Q}G_q \\ (q,i)\mapsto H\in G_q}} \bigcup_{\substack{z_2:K_2\to\cup_{q\in Q}G_q \\ (q,i)\mapsto H\in G_q}} \\
&\quad \left\{ \bigcup_{(p,j)\in K_1} (p,j); z_1((p,j)) \right\} \cup \left\{ \bigcup_{(p,j)\in K_2} (p,j); z_2((p,j)) \right\}
\end{aligned}$$

Using z as defined above, we can rewrite this as

$$\bigcup_{\substack{K_1 \cup K_2 \\ K_1 \in F_1, K_2 \in F_2}} \bigcup_{\substack{z : K_1 \cup K_2 \to \bigcup_{q \in Q} G_q \\ (q,i) \mapsto H \in G_q}} \left\{ \bigcup_{(p,j) \in K_1 \cup K_2} (p,j); z((p,j)) \right\},$$

which proves the claim. ◀

Towards a proof of the first part of Proposition 14, we prove the following Lemma.

▶ **Lemma 24.** *Let K be a clause of $\sigma_{q\alpha}$ for $\alpha = wX\beta$.*
(1) If $X \in N_\Box$, there is $X \to \eta$ and a clause K' of $\sigma_{qw\eta\beta}$ such that $K' \subseteq K$.
(2) If $X \in N_\bigcirc$, for all $X \to \eta$ there is a clause K' of $\sigma_{qw\eta\beta}$ such that $K' \subseteq K$.

Proof. Let $q \xrightarrow{w}_i p$, i.e. p is the unique state in which A_P is after processing w from q. Let $F = \sigma_{qwX\beta}$. We assume that $X \to \eta_1, \ldots, X \to \eta_k$ are rules with X as their left-hand side, and let $F_{\eta_i} = \sigma_{qw\eta_i\beta}$.

(1) By Lemma 23 (3) and associativity (Lemma 13), clause K of F is given by a clause $(p,i); \hat{K}$ of σ_{qwX} and a function z mapping this clause to $\bigcup_{q' \in Q} \sigma_{q'\beta}$. Since $X \in N_\Box$, we have $\sigma_{pX} = \bigwedge_{X \to \eta_j} \sigma_{p\eta_j}$. In particular, the clause $\hat{K}$ is already a clause in $\sigma_{p\eta_j}$ for some η_j, and $(p,i); \hat{K}$ is a clause of $\sigma_{qw\eta_j}$. Consequently, K is also a clause of $F_{q\eta_j} = \sigma_{qw\eta_j\beta}$. We may choose the rule $X \to \eta_j$ and $K' = K$.
(2) By Lemma 23 (3) and associativity (Lemma 13), clause K of F is given by a clause $(p,i); \hat{K}$ of σ_{qwX} and a function z mapping this clause to $\bigcup_{q' \in Q} \sigma_{q'\beta}$. Since $X \in N_\bigcirc$, we have $\sigma_{pX} = \bigvee_{X \to \eta_j} \sigma_{p\eta_j}$. In particular, $\hat{K} = K_1 \cup \ldots \cup K_k$, where K_j is a clause of $\sigma_{p\eta_j}$. Let $X \to \eta_j$ be some arbitrary move. Note that $(p,i); K_j$ is a clause of $\sigma_{qw\eta_j}$, and $(p,i); K_j \subseteq (p,i); \hat{K}$. Consider the clause K' of $F_{\eta_j} = \sigma_{qw\eta_j\beta}$ defined by $(p,i); K_j$ and the map z restricted to $(p,i); K_j$. Note that $K = \bigcup_{(q',i') \in (p,i);\hat{K}} (q',i'); z(q',i')$ and $K' = \bigcup_{(q',i') \in (p,i);K_j} (q',i'); z_{\restriction_{(p,i);K_j}}(q',i') = \bigcup_{(q',i') \in (p,i);K_j} (q',i'); z(q',i')$, so $K' \subseteq K$ holds. ◀

Proof of Proposition 14 (1). We consider the strategy s_K that keeps track of a clause K of the current formula. Initially, this clause is $K \in \sigma_{q\alpha}$. Whenever refuter makes a move $X \to \eta$, we track the clause K' of the formula of the new position as in Lemma 24 (2). Whenever it is our turn, we choose a rule $X \to \eta$ and the clause K' of the formula of the new position as in Lemma 24 (1). Note that since $K' \subseteq K$ in the Lemma, along a play $\alpha, \alpha^{(1)}, \alpha^{(2)}, \ldots$ that is conform to the strategy, we obtain a chain of clauses

$$K \supseteq K^{(1)} \supseteq K^{(2)} \supseteq \ldots .$$

In case the play is infinite, it has the desired property anyway. If it ends in a terminal word w, note that the formula σ_{qw} is the singleton formula $\sigma_{qw} = \{\{(p,i)\}\}$, where $q \xrightarrow{w}_i p$. Since we keep track of a clause of each occurring formula, the clause has to be $\{(p,i)\}$. The clauses of the chain form a descending chain, so we have $\{(p,i)\} \subseteq K$, and therefore $(p,i) \in K$. ◀

The following development aims to prove the second part of Proposition 14. It is mostly analogous to the development in [16], but some modifications have to be made since we consider the states of a deterministic parity automaton (plus priorities) instead of boxes for a non-deterministic automaton as atomic propositions.

The strategy s_c for a choice-function c is more involved, since we have to guarantee termination. To describe how far a sentential form is away from being a terminal word, we

use Kleene approximants. Define a *sequence of levels lvl* associated to a sentential form α to be a sequence of natural numbers of the same length. The formula $\sigma^{lvl}_{q\alpha}$ corresponding to α and lvl is defined by $\sigma^{i}_{qa} = \{\{qa\}\}$ for all $a \in T \cup \{\varepsilon\}$, σ^{i}_{qX} the solution to qX from the i^{th} Kleene iteration, and $\sigma^{lvl.lvl'}_{\alpha.\beta} = \sigma^{lvl}_{q\alpha} : (\sigma^{lvl'}_{q\beta})_{q\in Q}$.

A choice function for q, α and lvl is a choice function on $\sigma^{lvl}_{q\alpha}$. Note that σ^{i}_{qa} is independent of i for terminals qa. Moreover, there is an i_0 so that $\sigma^{i_0}_{qX} = \sigma_{qX}$ for all non-terminals X. This means a choice function on $\sigma_{q\alpha}$ can be understood as a choice function on $\sigma^{i_0}_{q\alpha}$. Here, we use a single number i_0 to represent a sequence $lvl = i_0 \dots i_0$ of the appropriate length.

By definition, σ^{0}_{qX} is *false* for all non-terminals, and *false* propagates through composition by definition. We combine this observation with the fact that choice functions do not exist on formulas that are equivalent to *false*, because they contain an empty clause.

▶ **Lemma 25.** *If there is a choice function for q, α and lvl, then lvl does not assign zero to any non-terminal X in α.*

The Lemma has an important consequence. Consider a sentential form α with an associated sequence $lvl \in 0^*$ and a choice function c for q, α and lvl. Then α has to be a terminal word, $\alpha = w \in T^*$, $\sigma^{lvl}_{q\alpha} = \{\{(p, i)\}\}$, where $q \xrightarrow{w}_i p$, and the choice function has to select (p, i). In particular, w itself forms a maximal play from w on, and indeed the play ends in a word whose effect is contained in the image of the choice function.

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

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

Proof. By Lemma 22, 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 $c'(G) \subseteq c(F)$. ◀

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

▶ **Lemma 27.** *Let c be a choice function for q, $\alpha = wX\beta$ and lvl.*
(1) If $X \in N_\Box$, for all $X \to \eta$ there is a choice function c_η for q, $w\eta\beta$ and lvl_η that refines c.
(2) If $X \in N_\bigcirc$, there is $X \to \eta$ and a choice function c_η for q, $w\eta\beta$ and lvl_η that refines c.

Proof. Let $q \xrightarrow{w}_i p$, i.e. p is the unique state in which A_P is after processing w from q. Let $F = \sigma^{lvl}_{qwX\beta}$, and for each rule $X \to \eta$, let $F_\eta = \sigma^{lvl_\eta}_{q\eta\beta}$.

(1) By Lemma 23 (3) and associativity (Lemma 13), the clauses of F are given by a clause $(p, i); K$ of $\sigma^{lvl'.\ell}_{qwX} = \sigma^{\ell}_{qwX}$ and a function mapping the atomic propositions in this clause to $\bigcup_{q'\in Q} \sigma^{lvl''}_{q'\beta}$. Similarly, the clauses of F_η are given by a clause of $\sigma^{\ell-1}_{qw\eta}$ and a mapping from the atomic propositions to $\bigcup_{q'\in Q} \sigma^{lvl''}_{q'\beta}$. We have $\sigma^{\ell}_{pX} = \bigwedge_{X\to\eta} \sigma^{\ell-1}_{p\eta}$. Since the

conjunction corresponds to a union of the clause sets, Lemma 23 (1), every clause of $\sigma^{\ell-1}_{qw\eta}$ is already a clause of σ^{ℓ}_{qwX}. Hence, the clauses of F_η form a subset of the clauses of F. Since c selects an atomic proposition from every clause of F, we can define the refinement c_η on F_η by restricting c.

(2) We show that there is a rule $X \to \eta$ and a choice function c_η on $\sigma^{lvl_\eta}_{qw\eta\beta}$ refining c. Towards a contradiction, assume this is not the case. Then for each rule $X \to \eta$, there is at least one clause K''_η of $\sigma^{lvl_\eta}_{qw\eta\beta}$ that does not contain an atomic proposition in the image of c. By Lemma 23 (3) and associativity (Lemma 13), K''_η is defined by a clause $(p,i);K'_\eta$ of $\sigma^{\ell-1}_{w\eta}$ and a function z_η mapping the atomic propositions from this clause to $\bigcup_{q'\in Q}\sigma^{lvl''}_{q'\beta}$. We have $\sigma^{\ell}_{pX} = \bigvee_{X\to\eta}\sigma^{\ell-1}_{p\eta}$. A clause of σ^{ℓ}_{qwX} is thus, Lemma 23 (2), of the form

$$K = (p,i);\Big(\bigcup_{X\to\eta} K_\eta\Big) = \bigcup_{X\to\eta}(p,i);K_\eta\ ,$$

where each K_η is a clause of $\sigma^{\ell-1}_{p\eta}$. We construct the clause $K' = (p,i);\big(\bigcup_{X\to\eta}K'_\eta\big)$ of σ^{ℓ}_{qwX} using the K'_η from above. On this clause, we define the map $z' = \bigcup_{X\to\eta} z_\eta$ that takes an atomic proposition $(p,i);(q',i) \in (p,i);K'_\eta$ and returns $z_\eta\big((p,i);(q',i)\big)$. (If an atomic proposition $(p,i);(q',i)$ is contained in $(p,i);K'_\eta$ for several η, pick an arbitrary η among those.) By Lemma 23 (3), K' and z' define a clause of $\sigma^{lvl}_{q\alpha}$. The choice function c selects an atomic proposition $(p,i);(q',i');(q'',i'')$ out of this clause, where there is a rule $X \to \eta$ such that $(q',i') \in K'_\eta$ and $(q'',i'') \in z'\big((p,i);(q',i')\big) = z_\eta\big((p,i);(q',i')\big)$. This atomic proposition is also contained in K''_η, a contradiction to the assumption that no atomic proposition from K''_η is in the image of c. ◀

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

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

▶ **Lemma 28.** *$\prec$ 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 ε. 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. $b = \max_{j=1,\dots,|v_0|} v_0$, 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. ◀

We have now gathered the ingredients to prove the second part of the Proposition.

Proof of Proposition 14 (2). We show the following stronger claim: Given any triple consisting of a sentential form α, an associated sequence of levels lvl, and a choice function c for α and lvl, there is a strategy s_c such that all maximal plays conform to it and starting in α end in a terminal word w with $(p,i) \in \{c(K) \mid K \in \sigma_{q\alpha}\}$, where $q \xrightarrow{w}_i p$. This proves the

proposition by choosing α and c as given and $lvl = i_0...i_0$, where $i_0 \in \mathbb{N}$ is a number such that $\sigma = \sigma^{i_0}$.

To show the claim, note that $\prec$ on $\mathbb{N}^*$ is well founded and the minimal elements are exactly 0^* by Lemma 28, and $lvl_\eta \prec lvl$. This means we can combine Lemma 25 and Lemma 27 (for the step case) into a Noetherian induction. The latter lemma does not state that lvl_η assigns a positive value to each non-terminal, which was a requirement on lvl. This follows from Lemma 25 and the fact that c_η is a choice function. The strategy s_c for refuter always selects the rule that affords a refinement of the initial choice function c. ◀

Proof of Proposition 15. We prove both parts of the Proposition by handling the first step $X \to \eta Y$, and then using Proposition 14.

To simplify the notation in this proof, we assume that for each η that occurs in a rule $X \to \eta Y$, the corresponding non-terminal Y is unique.

(1) We need to distinguish two cases.
Case $X \in N_\Box$**:** Note that $\sigma^e_{qX} = \bigwedge_{X\to\eta Y} \sigma_{q\eta}.Y$. A clause of σ^e_{qX} is of the shape $K'.Y$, where K' is a clause of some $\sigma_{q\eta}$ with $X \to \eta Y$. If we play according to $s_{K'}$, the strategy constructed for η and K' in Proposition 14 (1), from ηY on, we either end up in an infinite play that does not visit a sentential form of shape wY, or we end up in wY, with $(p, i) \in K'$, where $q \xrightarrow{w}_i p$. Consequently, we have $(p, i, Y) \in K'.Y$.
This means we can define s^e_K by picking the rule $X \to \eta Y$, and then playing according to $s_{K'}$.
Case $X \in N_\bigcirc$**:** Note that $\sigma^e_{qX} = \bigvee_{X\to\eta Y} \sigma_{q\eta}.Y$. A clause of σ^e_{qX} is of the shape $(p, i); K'.Y$, where $K' = \bigcup_{X\to\eta Y} K_\eta$ and K_η is a clause of $\sigma_{q\eta}$ for each rule $X \to \eta Y$. Assume prover picks a rule $X \to \eta Y$. Then we play according to the strategy s_{K_η}, the strategy constructed for η and K_η in Proposition 14 (1), from ηY on. We either end up in an infinite play that does not visit a sentential form of shape wY, or we end up in wvY, with $(p_\eta, i_\eta) \in K_\eta$, where $q \xrightarrow{w}_{i_\eta} p_\eta$. Consequently, we have $(p_\eta, i_\eta, Y) \in K_\eta.Y$.

(2) We again distinguish two cases, depending on who owns X.
Case $X \in N_\Box$**:** Note that $\sigma^e_{qX} = \bigwedge_{X\to\eta Y} \sigma_{q\eta}.Y$. A clause of σ^e_{qX} is of the shape $K_\eta.Y$, where K_η is a clause of some σ_η. If prover picks a rule $X \to \eta$, we can restrict the choice function c to $\sigma_{q\eta}.Y$. The restricted choice function on $\sigma_{q\eta}.Y$ in turn induces a choice function c_η on $\sigma_{q\eta}$ by ignoring the Y-component. Then we play according to s_{c_η}, the strategy constructed for η and c_η in Proposition 14 (2), from ηY. We end up in wY, with $(p_\eta, i_\eta) \in c_\eta(\sigma_{q\eta})$, where $q \xrightarrow{w}_{i_\eta} p_\eta$. Consequently, we have $(p_\eta, i_\eta, Y) \in c(\sigma_{qw\eta}.Y) \subseteq c(\sigma_{qwX}.Y)$.
Case $X \in N_\bigcirc$**:** We claim that there is a move $X \to \eta Y$ such that the given choice function induces a choice function c_η on $\sigma_{q\eta}$ that is refining c, i.e. for each $(p, i) \in c_\eta(\sigma_{q\eta})$, there is $(p, i, Y) \in c(\sigma^e_X)$. Assume this is not the case, then for each move $X \to \eta Y$ for every clause K_η of $\sigma_{q\eta}$, there is no $(p_\eta, i_\eta) \in K_\eta$ such that that $(p_\eta, i_\eta, Y) \in c(\sigma^e_{qX})$. Consider $K = \bigcup_{X\to\eta Y} K_\eta$. Note that since $\sigma^e_{qX} = \bigvee_{X\to\eta Y} \sigma_{q\eta}.Y$, $K.Y$ is a clause of σ^e_{qX}. Therefore, the choice function c selects an atomic proposition (p_η, i_η, Y) from it. By the definition of K, there is a rule $X \to \eta Y$ such that $(p_\eta, i_\eta) \in K_\eta$, a contradiction to the assumption.
Altogether, there is a move $X \to \eta Y$ such that c induces a refinement c_η on $\sigma_{q\eta}$. If we play according to s_{c_η}, the strategy constructed for η and c_η in Proposition 14 (2), from ηY, we end up in wY, with $(p_\eta, i_\eta) \in c_\eta(\sigma_{q\eta})$, where $q \xrightarrow{w}_{i_\eta} p_\eta$. Consequently, we have $(p_\eta, i_\eta, Y) \in c(\sigma^e_{qX})$. ◀

C.2 Details on Subsection 4.2

▶ **Lemma 29.** *Let w be an infinite word, $w = w^{(0)}w^{(1)}\dots$ a decomposition into finite words, $i^{(0)}, i^{(1)}, \dots$ a sequence of priorities and $q_0 = q^{(0)}, q^{(1)}, \dots$ a sequence of states such that $q^{(0)} \xrightarrow{w^{(0)}}_{i^{(0)}} q^{(1)} \xrightarrow{w^{(1)}}_{i^{(1)}} \dots$. Word w is accepted by A_P if and only if the highest priority occurring infinitely often among the $i^{(j)}$ is even.*

Proof. Since A_P is deterministic, there is a unique run of A_P on w, and it can be seen as a refinement of the sequence $q^{(0)} \xrightarrow{w^{(0)}}_{i^{(0)}} q^{(1)} \xrightarrow{w^{(1)}}_{i^{(1)}} \dots$. In particular, the highest priority i occurring infinitely often in the run is the highest priority occurring infinitely often among the $i^{(j)}$. Assume this would not be the case. Let j_0 such that after $q^{(j_0)}$, no state of priority $> i$ occurs any more. No larger priority can occur while processing each $w^{(j)}$ for $j > j_0$, so i occurs infinitely often among the $i^{(j)}$ for $j > j_0$, and no larger priority occurs at all. Together with the definition of the parity acceptance condition, this proves the claim. ◀

▶ **Lemma 30.** *If ○ has a winning strategy for the parity game from q_0S, then she has a winning strategy for the grammar game from S.*

Proof. Let $s_{\bigcirc}$ be the positional winning strategy for the parity game from q_0S. Since q_0S is owned by □, each successor of q_0S has to be in the winning region, i.e. each vertex q_0SK corresponding to a clause K of $\sigma^e_{q_0S}$. Since those vertices are owned by ○ and in her winning region, the strategy $s_{\bigcirc}$ selects a two-step successor pX (we ignore the intermediary helper vertex) that is also in her winning region. Let us consider the choice function c that selects the corresponding element (p, X, i) in each clause K. By Proposition 15 (2), there is a strategy for the grammar game that leads to a position wX after finitely many steps, where $q_0 \xrightarrow{w}_i p$.

This position in the grammar game corresponds to the position pX in the parity game, which is in refuter's winning region. We iterate this process to obtain both, an infinite play of the grammar game that visits positions of the shape $w'Y$ infinitely often, and a play of the parity game. Note that the play of the parity game is conforming to a winning strategy, and is therefore winning.

We need to argue that the word generated by the infinite play of the grammar game is infinite and not in $\mathcal{L}^\omega(A)$. The only case where the word is not infinite is that ε is the only word generated infinitely often as v when going from a position $w'Y$ to $w'vY'$. Note that by the definition of the functions q_ε in the system of equations, the intermediary helper vertex (qYK, i, pY') corresponding to the generation of ε has priority $i = 0$. All vertices of type qY and qYK have priority 0 as well, so 0 would be the highest priority occurring infinitely often in the parity game, which means that the play of the parity game is not winning for ○. This is a contradiction, since the play was conform to a winning strategy for ○.

This establishes that the word generated by the play is indeed infinite. Note that the word is $w_{play} = w^{(0)}w^{(1)}\dots$, where each $w^{(j)}$ is the finite word generated when going from wY to $ww^{(j)}Y'$. To argue $w_{play} \notin \mathcal{L}^\omega(A)$, we show that the unique run of A_P on w_{play} is not accepting. The run of the parity game visits infinitely many positions of type (qXK, i, pY), since the self-loops that handle empty clauses do not occur in a play won by refuter. Actually, the positions of this shape occurring in the play form an infinite sequence $(q^{(j)}X^{(j)}K^{(j)}, i^{(j)}, q^{(j+1)}X^{(j+1)})$. Note that $q_0 = q^{(0)} \xrightarrow{w^{(0)}}_{i^{(0)}} q^{(1)} \xrightarrow{w^{(1)}}_{i^{(1)}} \dots$ forms a sequence as in Lemma 29, and since the play was won by refuter, the largest $i^{(j)}$ occurring infinitely often has to be odd, so $w_{play} \notin \mathcal{L}^\omega(A)$. ◀

▶ **Lemma 31.** *If $\square$ has a winning strategy for the parity game form q_0S, then she has a winning strategy for the grammar game from S.*

Proof. Let $s_\square$ be the positional winning strategy for the parity game from q_0S. Let q_0SK be the successor vertex selected by $s_\square$. Since $s_\square$ is a winning strategy, q_0SK is in the winning region. Note that K is a clause of $\sigma^e_{q_0S}$.

By Proposition 15 (1), there is a strategy s_K for the grammar game that either leads to an infinite play that does not visit a position of the shape wX, or to a position wX after finitely many steps, where $(q, i, X) \in K$ and $q_0 \xrightarrow{w}_i q$. In the first case, the resulting play of the grammar game is won by prover by definition. In the second case, note that q_0SK has a corresponding two-step successor qX. Since q_0SK was in the winning region of prover, but owned by refuter, all its successors have to be in the winning region. In particular, qX is in the winning region of prover (and the deterministic step to the helper vertex does not matter).

We iterate this process to obtain both a play of the grammar game and a play of the parity game, where the play of the parity game is conforming to a winning strategy, and is therefore winning. As mentioned above, the play is trivially won by prover if we do not see infinitely many positions of the shape wX. (In particular, this occurs if any of the clauses K selected by $s_\square$ is empty. In this case, the corresponding play of the parity game takes the self-loop in the vertex for the clause.) Furthermore, plays that deadlock and plays that generate only a finite word (because ε is the only word generated infinitely often) are also won by prover.

Let us assume that the play of the grammar game is infinite, generates an infinite word w_{play} and visits positions of shape wX infinitely often. Note that $w_{play} = w^{(0)}w^{(1)}\ldots$, where each $w^{(j)}$ is the finite word generated when going from wY to $ww^{(j)}Y'$. To argue $w_{play} \in \mathcal{L}^\omega(A)$, we show that the unique run of A_P on w_{play} is accepting. The run of the parity game visits infinitely many positions of type (qXK, i, pY), since plays of the parity game that involve self-loops that handle empty clauses correspond to plays of the grammar gane that do not see positions of the shape wX infinitely often. Actually, the helper vertex occurring in the play form an infinite sequence $(q^{(j)}X^{(j)}K^{(j)}, i^{(j)}, q^{(j+1)}X^{(j+1)})$. Note that $q_0 = q^{(0)} \xrightarrow{w^{(0)}}_{i^{(0)}} q^{(1)} \xrightarrow{w^{(1)}}_{i^{(1)}} \ldots$ forms a sequence as in Lemma 29, and since the play was won by prover, the largest $i^{(j)}$ occurring infinitely often has to be even, so $w_{play} \in \mathcal{L}^\omega(A)$. ◀

C.3 Details on Subsection 4.3

Proof of Theorem 19. Given a context-free game G, A, we construct G_ω and A_ω as follows: We add a new symbol # to the terminal symbols. To obtain G_ω, we add a new initial symbol S_ω and a rule $S_\omega \to SH_\omega$, where S is the initial symbol of G, and H_ω is another fresh symbol that has a rule $H_\omega \to \#H_\omega$. To obtain A_ω, we add a new state q_ω, and for each transition that goes to a final state, we add a transition with the same input symbol to q_ω. We add a loop $q_\omega \xrightarrow{\#} q_\omega$, and we make q_ω the only final state of A_ω.

Note that $\mathcal{L}^\omega(G_\omega) = \mathcal{L}(G)\#^\omega$, and $\mathcal{L}^\omega(A) = \mathcal{L}(A)\#^\omega$.

Refuter wins the ω-regular non-inclusion game with respect to G_ω, A_ω from S_ω if and only if she wins the (finite) non-inclusion game G, A from S. Since the size of G_ω resp. A_ω is polynomial in the size of G resp. A, and solving (finite) non-inclusion games is 2EXPTIME-hard by [16], this proves the claim. ◀

Proof of Theorem 20. We analyze (1) the time needed to construct the deterministic parity automaton, (2) the number of iterations needed for the fixed point-iteration, (3) the

time consumption per iteration, (4) the time for extending the solution, (5) the time the construction of the parity game, and (6) the time needed to solve the parity game.

Steps (2) and (3) are analogous to the proof of the corresponding result in [16].

Here, Q is the set of states of the Büchi automaton A.

(1) As stated by Theorem 11, given a NBA with $|W|$ states, one can construct a deterministic parity automaton A_P with $m = |Q| = 2^{\mathcal{O}(|Q| \cdot \log |Q|)} \leq 2^{|Q|^{c_{qpa}}}$ many states and maximal priority $2 \cdot |Q| + 2 \leq |Q|^{c_{prio}}$ for some $c_{qpa}, c_{prio} \in \mathbb{N}$. The state set of this automaton consists of the Safra-trees for A, and it can be constructed in singly exponential time $2^{|Q|^{c_{det}}}$ for some $c_{det} \in \mathbb{N}$.

(2) 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. 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 $|N| \cdot m \cdot 2^k$, 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, i.e. 2^k times the number of variables $|N| \cdot m$.

Each atomic proposition consists of a state in Q and a priority in $\{0, \ldots, 2|Q| + 2\}$, i.e. $k = m \cdot (2|Q| + 2) \leq 2^{|Q|^{c_{qpa}}} \cdot |Q|^{c_{prio}} \leq 2^{|Q|^{c_{atom}}}$. Altogether, the iteration needs

$$|N| \cdot m \cdot 2^k \leq |N| \cdot 2^{|Q|^{c_{qpa}}} \cdot 2^{2^{|Q|^{c_{atom}}}} \leq |N| \cdot 2^{2^{|Q|^{c_{itr}}}}$$

many steps for some $c_{itr} \in \mathbb{N}$.

(3) Let $k \leq 2^{|Q|^{c_{atom}}}$ be again the number of atomic propositions. Every clause has at most size k and there are 2^k different clauses, so every formula has size at most $k \cdot 2^k \leq 2^{|Q|^{c_{atom}}} \cdot 2^{2^{|Q|^{c_{atom}}}} \leq 2^{2^{|Q|^{c_{form}}}}$ for some $c_{form} \in \mathbb{N}$.

Computing the operations used during the iteration according to Lemma 23 is polynomial in the size of the formulas. For disjunction and conjunction, this is clear. For composition, we need to iterate over the at most 2^k clauses and over the at most

$$\begin{aligned}(m \cdot 2^k)^k &\leq \left(2^{|Q|^{c_{qpa}}} \cdot 2^{2^{|Q|^{c_{atom}}}}\right)^{2^{|Q|^{c_{atom}}}} = \left(2^{|Q|^{c_{qpa}}}\right)^{2^{|Q|^{c_{atom}}}} \cdot \left(2^{2^{|Q|^{c_{atom}}}}\right)^{2^{|Q|^{c_{atom}}}} \\ &= 2^{|Q|^{c_{qpa}} \cdot 2^{|Q|^{c_{atom}}}} \cdot 2^{2^{|Q|^{c_{atom}}} 2^{|Q|^{c_{atom}}}} \leq 2^{2^{|Q|^{c_{fct}}}}\end{aligned}$$

functions mapping atomic propositions to clauses, for some $c_{fct} \in \mathbb{N}$. Each clause K and function z determines a clause of the composition. To obtain its atomic proposition, we need to iterate over the at most k atomic propositions (p, i) of K and compute $(p, i); z(p, i)$. To do this, we need to iterate over the at most k atomic propositions (q, j) of $z(p, i)$ and compute $(p, i); (q, j)$. Overall, to compute the relational composition of two formulas, we need

$$2^k \cdot 2^{2^{|Q|^{c_{fct}}}} \cdot k \cdot k \cdot (2^{|Q|^{c_{qpa}}} \cdot |Q|^{c_{prio}}) \leq 2^{2^{|Q|^{c_{comp}}}}$$

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

Per iteration, we need to carry out at most $m \cdot |G|$ 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 at most

$$m \cdot |G| \cdot \left(\left(2^{2^{|Q|^{c_{form}}}} \right)^{c_\wedge} + \left(2^{2^{|Q|^{c_{form}}}} \right)^{c_\vee} + 2^{2^{|Q|^{c_{comp}}}} \right) \leq |G| \cdot 2^{2^{|Q|^{c_{peritr}}}}$$

for some constants $c_\wedge, c_\vee, c_{peritr} \in \mathbb{N}$. Here, $2^{2^{|Q|^{c_{comp}}}}$ is the cost of computing the relational composition earlier, and $\left(2^{2^{|Q|^{c_{form}}}} \right)^{c_\wedge}$ and $\left(2^{2^{|Q|^{c_{form}}}} \right)^{c_\vee}$ are rough estimations for computing conjunction and disjunction.

(4) If we extend the solution of the equation system, we need compute the disjunction or conjunction over at most $|G|$ many finite solutions for each non-terminal X and each state $q \in Q$. Consequently, we need up to

$$|N| \cdot 2^{|Q|^{c_{qpa}}} \cdot |G| \cdot \left(\left(2^{2^{|Q|^{c_{form}}}} \right)^{c_\wedge} + \left(2^{2^{|Q|^{c_{form}}}} \right)^{c_\vee} \right) \leq |G| \cdot 2^{2^{|Q|^{c_e}}}$$

many steps for some constant $c_e \in \mathbb{N}$.

(5) The parity game has $m \cdot |N|$ many vertices owned by prover. Furthermore, we have $m \cdot |N| \cdot 2^{k'}$ many vertices for the clauses owned by refuter, where $k' = m \cdot (2|Q| + 2) \cdot |N|$ is the number of atomic propositions in the extended equation system. Additionally, there are up to $m \cdot |N| \cdot 2^{k'} \cdot k'$ many helper vertices also owned by refuter. The number of edges is polynomial in the number of vertices, and the highest occurring priority is again $2|Q| + 2$. Altogether, the size of the parity game is at most

$$m \cdot |N| + m \cdot |N| \cdot 2^{k'} + m \cdot |N| \cdot 2^{k'} \cdot k' \leq 2^{2^{|Q|^{c_{Q_{parity}}}}} \cdot 2^{|N|^{c_{N_{parity}}}}$$

for some constants $c_{Q_{parity}}, c_{N_{parity}} \in \mathbb{N}$. Furthermore, the parity game can be constructed in a time that is polynomial in its size. If we choose the constants to be sufficiently high, the constructing the parity game is possible within the same bound.

(6) In general, solving parity games is in $\mathsf{NP} \cap \mathsf{coNP}$, so we would not expect to have a deterministic 2EXPTIME-algorithm that solves our parity game of doubly-exponential size.

Fortunately, the number of nodes by prover is only singly exponential, namely $m \cdot |N| \leq 2^{|Q|^{c_{qpa}}} \cdot |N|$. By Theorem 16, prover wins the parity game if and only if she has a positional winning strategy $s_\Box$. A positional strategy is a function that takes a vertex pX and returns one of the $2^{k'}$-many clauses of σ^e_{pX}. Therefore, there are only up to

$$(2^{k'})^{2^{|Q|^{c_{qpa}}} \cdot |N|} \leq 2^{2^{|Q|^{c_{qpa}}} \cdot (2|Q|+2) \cdot |N| \cdot 2^{|Q|^{c_{qpa}}} \cdot |N|} \leq 2^{2^{|Q|^{c_{Q_{strat}}}}} \cdot 2^{|N|^{c_{N_{strat}}}}$$

many such strategies, for some constants $c_{Q_{strat}}, c_{N_{strat}} \in \mathbb{N}$.

After some strategy is fixed, checking whether it is winning is polynomial in the size of the game. We drop all edges originating in vertices owned by prover that were not selected by the winning strategy. In the resulting graph, we need to check whether refuter can win. In this case, the strategy would not be winning. To do this, we have to check whether there is a vertex v reachable from the initial vertex $q_0 S$ such that there is a cycle from/to v such that the highest priority occurring on an edges in the cycle is odd. If we choose the constants $c_{Q_{strat}}, c_{N_{strat}}$ to be sufficiently large, one can iterate over all strategies and check whether they are winning in $2^{2^{|Q|^{c_{Q_{strat}}}}} \cdot 2^{|N|^{c_{N_{strat}}}}$ many steps.

Altogether we need

$$2^{|Q|^{c_{det}}} + \left(|N| \cdot 2^{2^{|Q|^{c_{itr}}}}\right) \cdot \left(|G| \cdot 2^{2^{|Q|^{c_{peritr}}}}\right) + |G| \cdot 2^{2^{|Q|^{c_e}}} + 2^{2^{|Q|^{cQ_{parity}}}} \cdot 2^{|N|^{cN_{parity}}} + 2^{2^{|Q|^{cQ_{strat}}}} \cdot 2^{|N|^{cN_{strat}}} \leq 2^{2^{|Q|^{c_1}}} \cdot 2^{|G|^{c_2}}$$

many steps for some constants $c_1, c_2 \in \mathbb{N}$. Here, we used the rough estimation $|N| \leq |G|$. ◀