arXiv:2404.09862v1 [cs.FL] 15 Apr 2024

A Uniform Framework for Language Inclusion Problems

Kyveli Doveri[1,2][0000−0001−9403−2860], Pierre Ganty[1][0000−0002−3625−6003], and Chana Weil-Kennedy[1][0000−0002−1351−8824]

[1] IMDEA Software Institute, Madrid, Spain
firsname.lastname@imdea.org
[2] Universidad Politécnica de Madrid, Spain

Abstract. We present a uniform approach for solving language inclusion problems. Our approach relies on a least fixpoint characterization and a quasiorder to compare words of the "smaller" language, reducing the inclusion check to a finite number of membership queries in the "larger" language. We present our approach in detail on the case of inclusion of a context-free language given by a grammar into a regular language. We then explore other inclusion problems and discuss how to apply our approach.

Keywords: Formal Languages, Inclusion, Containment, Algorithm, Quasiorders

1 Introduction

We are interested in the classical problem of language inclusion, which given two language acceptors, asks whether the language of one acceptor is contained into the language of the other one. This problem is traditionally solved by complementing the "larger" language, intersecting with the "smaller" language and checking for emptiness. Here, we avoid explicit complementation. We present a simple and uniform approach for deciding language inclusion problems based on the use of quasiorder relations on words. In this paper we are interested in the decidable cases of this problem, where the underlying alphabet of the language is a finite set of symbols. Even though we focus mainly on words of finite length our approach also applies to languages of infinite words. At its core, our approach relies on two notions: a *fixpoint characterization* of the "smaller" language and a *quasiorder* to compare words. Intuitively, the language inclusion algorithms we derive leverage the fixpoint characterization to compute increasingly many words of the "smaller" language. After a finite amount of time the computation is stopped. The algorithm then tests whether each of the computed words of the "smaller" language also belong to the "larger" one. If one word fails the test, the inclusion does not hold and we have a counterexample. Whether we can correctly conclude that inclusion holds depends on whether we have computed "enough" words. The rôle of the quasiorder is precisely that of detecting when "enough" words have been computed. It must satisfy two properties: the first one ensures that it takes only a finite amount of time to compute "enough" words, and the second one ensures that if there exists a counterexample to inclusion, then it has been computed.

We present our approach in detail on the case of inclusion of a context-free language given by a grammar into a regular language, an EXPTIME-complete problem [20]. This case is simple yet non-trivial. After presenting our approach in Section 3, we give distinct quasiorders that can be used in the decision procedure in Section 4. Section 5 then dives into the algorithmic aspects related to using the so-called state-based quasiorders. The state-based quasiorders enable some modifications of the inclusion algorithm ultimately leading to the so-called antichains algorithms [6,17]. We also show how the saturation approach put forward by Esparza et al. [10] can be leveraged for the particular inclusion problem of a straight-line program[3] into a finite state automaton. In Section 6 we talk about the other language classes to which our framework can be applied. We revisit the case where the two languages are given by finite state automata. Next, we investigate the case asking whether the trace set of a finite process (which is a regular language) is contained into the trace set of a Petri net, a case that was first solved by Esparza et al. [19]. To demonstrate the generality of the approach, we survey how it can also be leveraged for the case of two languages of infinite words accepted by Büchi automata. We finish by briefly mentioning a few more cases that can be tackled using our approach.

This paper works as an overview of our previous work on the topic, providing a simplified explanation and pointers to the previous papers. In particular, the section presenting our approach greatly simplifies the framework put forward in [13].

2 Preliminaries

Well-Quasiorders, Complete Lattices and Kleene Iterates A *quasiorder* (qo) on a set E, is a binary relation $\ltimes \subseteq E \times E$ that is reflexive and transitive. A quasiorder $\ltimes$ is a *partial order* when $\ltimes$ is antisymetric ($x \ltimes y \wedge y \ltimes x \implies x = y$). A *complete lattice* is a set E and a partial order $\ltimes$ on E such that every subset $X \subseteq E$ has a least upper bound (the *supremum*) in E.

A sequence $\{s_n\}_{n\in\mathbb{N}} \in E^{\mathbb{N}}$ on quasiordered set $(E, \ltimes)$ is *increasing* if for every $n \in \mathbb{N}$ we have $s_n \ltimes s_{n+1}$. For a function $f : E \to E$ on a quasiordered set $(E, \ltimes)$ and for all $n \in \mathbb{N}$, we define the n-th iterate $f^n : E \to E$ of f inductively as follows: $f^0 \triangleq \lambda x.\, x$; $f^{n+1} \triangleq f \circ f^n$. The denumerable sequence of *Kleene iterates* of f starting from the bottom value $\bot \in E$ is given by $\{f^n(\bot)\}_{n\in\mathbb{N}}$. A *fixpoint* of f is x such that $f(x) = x$. The *least fixpoint* of f is the smallest fixpoint of f with respect to $\ltimes$, if f has at least one fixpoint; it is denoted $\operatorname{lfp} f$. Recall that when $(E, \ltimes)$ is a complete lattice and $f : E \to E$ is an monotone function (i.e. $d \ltimes d' \implies f(d) \ltimes f(d')$) then it follows from the Knaster–Tarski theorem

[3] Straight-line program are context-free grammars where at most one word is derived from each grammar variable.

that f has a least fixpoint $\operatorname{lfp} f$ which, provided f is continuous[4], is given by the supremum of the increasing sequence of Kleene iterates of f.

Given a quasiorder $\ltimes$ on a set E, and $X \subseteq E$ a subset, let $\uparrow_{\ltimes} X$ denote the upward closure of X with respect to $\ltimes$ given by $\{y \in E \mid \exists x \in X. x \ltimes y\}$. A quasiorder $\ltimes$ on E is a *well-quasiorder* (wqo) if for every infinite sequence $\{S_n\}_{n\in\mathbb{N}} \in \wp(E)^{\mathbb{N}}$ such that $\uparrow_{\ltimes} S_1 \subseteq \uparrow_{\ltimes} S_2 \subseteq \cdots$ we have that there exists $i \in \mathbb{N}$ such that $\uparrow_{\ltimes} S_i \supseteq \uparrow_{\ltimes} S_{i+1}$, namely, $\ltimes$ is a wqo iff there is no infinite strictly increasing chain of upward closed subsets in E.

There are other equivalent definitions for wqo but the one using upward closed sets is the most convenient for our purpose. One property of wqo we will leverage throughout the paper is that the component wise lifting of a wqo remains a wqo, namely, given $n \in \mathbb{N}$, if $\ltimes$ is a wqo then so is $\ltimes^n$.

Finally, we introduce the lifting of the qo $\ltimes$ to sets by defining $\sqsubseteq_{\ltimes} \subseteq \wp(E) \times \wp(E)$ as follows:

$$X \sqsubseteq_{\ltimes} Y \overset{\triangle}{\iff} \forall x \in X, \exists y \in Y, y \ltimes x \ .$$

It is routine to check that $\sqsubseteq_{\ltimes}$ is a quasiorder as well and also that $X \sqsubseteq_{\ltimes} Y$ iff $\uparrow_{\ltimes} X \subseteq \uparrow_{\ltimes} Y$. Given the previous equivalence, the rationale to introduce $\sqsubseteq_{\ltimes}$ is algorithmic since $\sqsubseteq_{\ltimes}$ is straightforward to implement for finite sets X, Y given a decision procedure for $\ltimes$; whereas $\uparrow_{\ltimes} X \subseteq \uparrow_{\ltimes} Y$ does not give a straightforward implementation even when X and Y are finite and $\ltimes$ is decidable.

Alphabets, Words and Languages An *alphabet* is a nonempty finite set of symbols, generally denoted by Σ. A *word* is a sequence of symbols over the alphabet Σ. The set of finite words and the set of infinite words over Σ are denoted by Σ^* and Σ^ω respectively. We denote by ε the *empty word* and define $\Sigma^+ \triangleq \Sigma^* \setminus \{\varepsilon\}$. A *language of finite words* over Σ is a subset of Σ^*. A *language of infinite words* or *ω-language* over Σ is a subset of Σ^ω.

Finite Automata A *finite automaton* (FA) on an alphabet Σ is a tuple $\mathcal{A} = (Q, \delta, q_I, F)$ where Q is a finite set of states including an initial state $q_I \in Q$, $\delta \colon Q \times \Sigma \to \wp(Q)$ is a transition function, and $F \subseteq Q$ is a subset of final states. Let $q \xrightarrow{a} q'$ denote a transition $q' \in \delta(q, a)$ that we lift to finite words by transitive and reflexive closure, thus writing $q \xrightarrow{u}{}^{*} q'$ with $u \in \Sigma^*$. The language of finite words accepted by $\mathcal{A}$ is $L^*(\mathcal{A}) \triangleq \{u \in \Sigma^* \mid \exists q \in F. q_I \xrightarrow{u}{}^{*} q\}$. An *accepting trace* of $\mathcal{A}$ on an infinite word $w = a_0 a_1 \cdots \in \Sigma^\omega$ is an infinite sequence $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \cdots$ such that $q_0 = q_I$ and $q_j \in F$ for infinitely many j's. The ω-language accepted by $\mathcal{A}$ is $L^\omega(\mathcal{A}) \triangleq \{\xi \in \Sigma^\omega \mid$ there is an accepting trace of $\mathcal{A}$ on $\xi\}$. We call $\mathcal{A}$ a Büchi automaton (BA) when we consider it as an acceptor of infinite words. A language $L \subseteq \Sigma^\omega$ is *ω-regular* if $L = L^\omega(\mathcal{A})$ for some BA $\mathcal{A}$.

Context-free Grammars A *context-free grammar* (CFG) or simply *grammar* on Σ is a tuple $\mathcal{G} = (V, P)$ where $V = \{X_1, \ldots, X_n\}$ is the finite set of variables,

[4] f is continuous iff f preserves least upper bounds of nonempty increasing chains.

and P is the finite set of production rules $X_i \to \beta$ where $\beta \in (V \cup \Sigma)^*$. Given $\alpha, \alpha' \in (V \cup \Sigma)^*$, we write $\alpha \Rightarrow \alpha'$ if there exists $\gamma, \delta \in (V \cup \Sigma)^*$ and $X_i \to \beta \in P$ such that $\alpha = \gamma X_i \delta$ and $\alpha' = \gamma\beta\delta$. The reflexive and transitive closure of $\Rightarrow$ is written $\Rightarrow^*$. For $i \in [1, n]$, let $L_i(\mathcal{G}) = \{w \in \Sigma^* \mid X_i \Rightarrow^* w\}$. When we omit the subscript i it is meant to be 1, hence we define the *language accepted* by $\mathcal{G}$ as $L(\mathcal{G})$, that is $L_1(\mathcal{G})$ given by $\{w \in \Sigma^* \mid X_1 \Rightarrow^* w\}$. A CFG is in Chomsky normal form (CNF) if all of its production rules are of the form $X_j \to X_k\, X_l$, $X_j \to a$, or $X_1 \to \varepsilon$ where $X_j, X_k, X_l \in V$ and $a \in \Sigma$.

Example 1. Let $\mathcal{G}' = (V', P')$ be the CFG on $\Sigma = \{a, b\}$ given by $V' = \{X_1\}$ and $P' = \{X_1 \to \varepsilon, X_1 \to a\, X_1\, b\}$. Notice that this grammar is not in CNF. The language $L(\mathcal{G}')$ is $\{a^n b^n \mid n \geq 0\}$. We can define a grammar $\mathcal{G} = (V, P)$ in CNF such that $L(\mathcal{G}) = L(\mathcal{G}')$. Let $V = \{X_1, X_2, X_3, X_4\}$ and P be the rules

$$\begin{array}{lll} X_1 \to \varepsilon & X_2 \to a & X_1 \to X_2\, X_3 \\ & X_4 \to b & X_3 \to X_1\, X_4 \ . \end{array}$$

3 Algorithm

In this section we outline the quasiorder-based approach to solving the inclusion problem of a context-free language into a regular language, which reduces the problem to finitely many membership queries. A membership query is a check of whether a given word belongs to a given language. More precisely, we consider the inclusion problem $L(\mathcal{G}) \subseteq M$ where $\mathcal{G}$ is a CFG and M is a regular language. To solve the inclusion problem, we select a subset S of words of $L(\mathcal{G})$ such that (i) S is finite, (ii) S is effectively computable, and (iii) S contains a counterexample to $L(\mathcal{G}) \subseteq M$ if the inclusion does not hold. Upon computing such a set S the inclusion check $L(\mathcal{G})$ into M reduces to finitely many membership queries of the words of S into M.

More concretely, the computation of S will be guided by a quasiorder $\ltimes$ on words. The set S of words is such that, as per the quasiorder $\ltimes$, the following holds: $S \subseteq L(\mathcal{G})$ and $L(\mathcal{G}) \sqsubseteq_{\ltimes} S$. [5] In that setting, for S to satisfy (i), we will require (1) $\ltimes$ to be a well-quasi order. Moreover for S to comply with (ii), we will require (2) $\ltimes$ to be decidable but also "monotonic" (it will become clear what it means and why we need this condition later). Finally, for S to comply with (iii), we first formalize the requirement:

$$L(\mathcal{G}) \subseteq M \iff \forall u \in S,\ u \in M \ . \qquad (\star)$$

Intuitively, it means that whatever set S we select needs to be included in M if the inclusion $L(\mathcal{G}) \subseteq M$ holds and otherwise the set S needs to contain at least one counterexample to the inclusion. To achieve this, we will require (3) $\ltimes$ to be M-preserving: A quasiorder $\ltimes \subseteq \Sigma^* \times \Sigma^*$ is said to be *M-preserving* when for every $u, v \in \Sigma^*$ if $u \in M$ and $u \ltimes v$ then $v \in M$[6]. To see how $\ltimes$

[5] We can even relax the inclusion $S \subseteq L(\mathcal{G})$ to the weaker condition $S \sqsubseteq_{\ltimes} L(\mathcal{G})$.
[6] This is equivalent to saying that M is upward-closed w.r.t. the quasiorder $\ltimes$.

being M-preserving enforces ($\star$) let us first assume that $L(\mathcal{G}) \subseteq M$ holds. Since $S \subseteq L(\mathcal{G})$ and $L(\mathcal{G}) \subseteq M$, the direction $\Rightarrow$ of Equation ($\star$) holds. This still holds if we have the weaker condition $S \sqsubseteq_{\ltimes} L(\mathcal{G})$. In this case, for every $s \in S$ there exists $u \in L(\mathcal{G})$ such that $u \ltimes s$, hence the assumption $L(\mathcal{G}) \subseteq M$ together with the M-preservation show that $s \in M$. Now assume that the right-hand side of ($\star$) holds and let $u \in L(\mathcal{G})$. Since $L(\mathcal{G}) \sqsubseteq_{\ltimes} S$ there is $s \in S$ such that $s \ltimes u$. Since $s \in M$, $s \ltimes u$ and $\ltimes$ is M-preserving we have $u \in M$.

To compute S we leverage a fixpoint characterization of $L(\mathcal{G})$. Then we effectively compute the set S by computing finitely many Kleene iterates of the fixpoint characterization of $L(\mathcal{G})$ so as to obtain a set S such that $L(\mathcal{G}) \sqsubseteq_{\ltimes} S$. In order to decide we have computed enough Kleene iterates we rely on $\sqsubseteq_{\ltimes}$ (which is why we need $\ltimes$ to be decidable). We also need $\ltimes$ to satisfy a so-called monotonicity condition for otherwise we cannot guarantee that S satisfies $L(\mathcal{G}) \sqsubseteq_{\ltimes} S$.

Let us now turn to the characterization of $L(\mathcal{G})$ as the least fixpoint of a function. Fix a grammar $\mathcal{G}$ in CNF and M a regular language. Our algorithms also work when the grammar $\mathcal{G}$ is not given in CNF, we assume CNF for the simplicity of the presentation.

Least Fixpoint Characterization. To compute our finite set S we use a characterization of $L(\mathcal{G})$ as the least fixpoint of a function, and show that we can iteratively compute the function's Kleene iterates until we compute a set S such that $L(\mathcal{G}) \sqsubseteq_{\ltimes} S$.

Example 2. Take the grammar $\mathcal{G}$ of Example 1 such that $L(\mathcal{G}) = \{a^n b^n \mid n \geq 0\}$. We define the function $F_{\mathcal{G}} \colon \wp(\Sigma^*)^4 \to \wp(\Sigma^*)^4$ where $(L_1, L_2, L_3, L_4) \in \wp(\Sigma^*)^4$ is a vector of languages of finite words as follows:

$$F_{\mathcal{G}} \colon (L_1, L_2, L_3, L_4) \mapsto (L_2 L_3 \cup \{\varepsilon\}, \{a\}, L_1 L_4, \{b\}) \enspace .$$

If we apply $F_{\mathcal{G}}$ to the empty vector $(\emptyset, \emptyset, \emptyset, \emptyset) \in \wp(\Sigma^*)^4$, we get the vector of languages $(\{\varepsilon\}, \{a\}, \emptyset, \{b\})$ (recall that $L\emptyset = \emptyset$ and $\emptyset L = \emptyset$ for any language L). Applying $F_{\mathcal{G}}$ to this last vector we obtain $(\{\varepsilon\}, \{a\}, \{b\}, \{b\})$, i.e. $F^2_{\mathcal{G}}(\emptyset, \emptyset, \emptyset, \emptyset) = (\{\varepsilon\}, \{a\}, \{b\}, \{b\})$. We give a list of the first few repeated applications of $F_{\mathcal{G}}$ to the empty vector.

$$\begin{aligned}
F_{\mathcal{G}}(\boldsymbol{\emptyset}) &= (\{\varepsilon\}, \{a\}, \emptyset, \{b\}) \\
F^2_{\mathcal{G}}(\boldsymbol{\emptyset}) &= (\{\varepsilon\}, \{a\}, \{b\}, \{b\}) \\
F^3_{\mathcal{G}}(\boldsymbol{\emptyset}) &= (\{ab, \varepsilon\}, \{a\}, \{b\}, \{b\}) \\
F^4_{\mathcal{G}}(\boldsymbol{\emptyset}) &= (\{ab, \varepsilon\}, \{a\}, \{ab^2, b\}, \{b\}) \\
F^5_{\mathcal{G}}(\boldsymbol{\emptyset}) &= (\{a^2b^2, ab, \varepsilon\}, \{a\}, \{ab^2, b\}, \{b\}) \\
F^6_{\mathcal{G}}(\boldsymbol{\emptyset}) &= (\{a^2b^2, ab, \varepsilon\}, \{a\}, \{a^2b^3, ab^2, b\}, \{b\}) \\
F^7_{\mathcal{G}}(\boldsymbol{\emptyset}) &= (\{a^3b^3, a^2b^2, ab, \varepsilon\}, \{a\}, \{a^2b^3, ab^2, b\}, \{b\}) \enspace .
\end{aligned}$$

It is not hard to see that for all words $a^n b^n \in L(\mathcal{G})$, there exists an i such that $a^n b^n$ appears in the first component of $F^i_{\mathcal{G}}(\boldsymbol{\emptyset})$. In fact $i = 2n + 1$.

We now formally define the function $F_{\mathcal{G}}$ over $\wp(\Sigma^*)^n$, i.e. over n-vectors of languages of finite words over Σ, where n is the number of variables of our grammar $\mathcal{G}$. Let $\mathcal{L} = (L_1, \ldots, L_n) \in \wp(\Sigma^*)^n$ be an n-vector of languages $L_1, \ldots, L_n$. For each $j \in [1, n]$, the j-th component of $F_{\mathcal{G}}(\mathcal{L})$, denoted $F_{\mathcal{G}}(\mathcal{L})_j$, is defined as:

$$F_{\mathcal{G}}(\mathcal{L})_j \triangleq \bigcup_{X_j \to X_k\, X_{k'} \in P} L_k\, L_{k'} \cup \bigcup_{a \in \Sigma \cup \{\varepsilon\},\ X_j \to a \in P} \{a\} \ .$$

We have the following least fixpoint characterization for $\mathcal{G}$.

Proposition 3 (Least fixpoint characterization). *For all $i \in \{1, \ldots, n\}$ we have $L_i(\mathcal{G}) = (\operatorname{lfp} F_{\mathcal{G}})_i$.*

Proof. The function $F_{\mathcal{G}}$ is increasing and the supremum of the increasing sequence of its Kleene iterates starting at the bottom value $(\emptyset, \ldots, \emptyset) \in \wp(\Sigma^*)^n$ is the vector equal to $\{u \in \Sigma^* \mid X_j \Rightarrow^* u\}$ for each $j \in [1, n]$. Therefore, by the Knaster–Tarski theorem applied to the complete lattice $(\wp(\Sigma^*)^n, \subseteq^n)$ and $F_{\mathcal{G}}$, $\operatorname{lfp} F_{\mathcal{G}} = (\{u \in \Sigma^* \mid X_j \Rightarrow^* u\})_{j \in [1,n]}$. Thus, $(\operatorname{lfp} F_{\mathcal{G}})_i = L_i(\mathcal{G})$. □

Remark 4. The function definition uses the fact that $\mathcal{G}$ is given in CNF, notably that its rules are of the form $X_j \to X_k X_{k'}$. However a function $F'_{\mathcal{G}}$ can be defined for a grammar $\mathcal{G}$ in any form such that $(\operatorname{lfp} F'_{\mathcal{G}}) = (L_1(\mathcal{G}), \ldots, L_n(\mathcal{G}))$ and if $\mathcal{G}$ is in CNF then $F'_{\mathcal{G}}$ and $F_{\mathcal{G}}$ coincide, see [12, Definition 2.9.1 and Theorem 2.9.3].

The ordering $\sqsubseteq_{\ltimes^n}$ is used to compare the Kleene iterates of the function $F_{\mathcal{G}}$. For it to be apt to detect convergence, hence when the algorithm should stop, the quasiorder $\ltimes$ needs to be monotonic. A quasiorder is *monotonic* if it is right-monotonic and left-monotonic. A quasiorder $\ltimes$ on Σ^* is *right-monotonic* (respectively *left-monotonic*) if for all $u, v \in \Sigma^*$, for all $a \in \Sigma$, $u \ltimes v$ implies $ua \ltimes va$ (respectively $u \ltimes v$ implies $au \ltimes av$).

Proposition 5 ($\sqsubseteq_{\ltimes}$ monotonicity). *Let $\ltimes$ be a monotonic quasiorder on Σ^*. If $Y \sqsubseteq_{\ltimes^n} S$ then $F_{\mathcal{G}}(Y) \sqsubseteq_{\ltimes^n} F_{\mathcal{G}}(S)$.*

Proof. Fix $j \in [1, n]$. We want to show that for all $y \in F_{\mathcal{G}}(Y)_j$ there exists $s \in F_{\mathcal{G}}(S)_j$ such that $s \ltimes y$. Let $y \in F_{\mathcal{G}}(Y)_j$. If there is an $a \in \Sigma \cup \{\varepsilon\}$ such that $X_j \to a \in P$ and $y = a$, then y is also in $F_{\mathcal{G}}(S)_j$ by definition. Since $\ltimes$ is a quasiorder it is reflexive, so $y \ltimes y$. Otherwise, by definition of $F_{\mathcal{G}}(Y)_j$, there exist $y_k \in Y_k, y_{k'} \in Y_{k'}$ such that $X_j \to X_k X_{k'} \in P$ and $y = y_k y_{k'}$. Since $Y \sqsubseteq_{\ltimes^n} S$, there exist $s_k \in S_k, s_{k'} \in S_{k'}$ such that $s_k \ltimes y_k$ and $s_{k'} \ltimes y_{k'}$. Using $X_j \to X_k X_{k'} \in P$ and the definition of $F_{\mathcal{G}}(S)_j$, we have $s_k s_{k'} \in F_{\mathcal{G}}(S)_j$. By right-monotonicity of our quasiorder, $s_k s_{k'} \ltimes y_k s_{k'}$. By left-monotonicity, $y_k s_{k'} \ltimes y_k y_{k'}$. Since quasiorders are transitive, we obtain $s_k s_{k'} \ltimes y_k y_{k'}$. We thus conclude from the above that $F_{\mathcal{G}}(S)_j \sqsubseteq_{\ltimes} F_{\mathcal{G}}(Y)_j$, hence it is easy to see that $F_{\mathcal{G}}(S) \sqsubseteq_{\ltimes^n} F_{\mathcal{G}}(Y)$. □

Under the assumption of monotonicity on our M-preserving well-quasiorder we can iteratively compute a finite set S such that $L(\mathcal{G}) \sqsubseteq_{\ltimes} S$ using the function $F_{\mathcal{G}}$ as shown by the next proposition.

Proposition 6 (Stabilization). *Let $\ltimes$ be a wqo on Σ^*. There is an $m \geq 0$ such that ${F_\mathcal{G}}^{m+1}(\boldsymbol{\emptyset}) \sqsubseteq_{\ltimes^n} {F_\mathcal{G}}^{m}(\boldsymbol{\emptyset})$; and if $\ltimes$ is monotonic then* $\operatorname{lfp} F_\mathcal{G} \sqsubseteq_{\ltimes^n} {F_\mathcal{G}}^{m}(\boldsymbol{\emptyset})$.

Proof. For simplicity, we write this proof for the case where n, the number of variables in $\mathcal{G}$, is equal to 1. It holds also for $n > 1$ by reasoning componentwise. Since $\ltimes$ is a wqo on Σ^*, the set of upward closed sets of $\wp(\Sigma^*)$ ordered by inclusion has the ascending chain condition [5, Theorem 1.1], meaning there is no infinite strictly increasing sequence of upward closed sets. It is routine to check by induction using Proposition 5 that ${F_\mathcal{G}}^{i}(\emptyset) \sqsubseteq_{\ltimes} {F_\mathcal{G}}^{i+1}(\emptyset)$ for all $i \geq 0$. Therefore also $\uparrow_{\ltimes} {F_\mathcal{G}}^{i}(\emptyset) \subseteq \uparrow_{\ltimes} {F_\mathcal{G}}^{i+1}(\emptyset)$ for all $i \geq 0$. By the ascending chain condition, there exists a positive integer m such that $\uparrow_{\ltimes} {F_\mathcal{G}}^{m+1}(\emptyset) \subseteq \uparrow_{\ltimes} {F_\mathcal{G}}^{m}(\emptyset)$, which is equivalent to ${F_\mathcal{G}}^{m+1}(\emptyset) \sqsubseteq_{\ltimes} {F_\mathcal{G}}^{m}(\emptyset)$.

Assume that $\ltimes$ is monotonic. An induction using Proposition 5 and the above shows that for every $k \geq m$, ${F_\mathcal{G}}^{k+1}(\emptyset) \sqsubseteq_{\ltimes} {F_\mathcal{G}}^{k}(\emptyset)$. Hence, by transitivity of $\sqsubseteq_{\ltimes}$ we deduce that for every $k \geq m$, ${F_\mathcal{G}}^{k}(\emptyset) \sqsubseteq_{\ltimes} {F_\mathcal{G}}^{m}(\emptyset)$. Recall that $\operatorname{lfp} F_\mathcal{G}$ is the supremum of the sequence of Kleene iterates of $F_\mathcal{G}$, i.e. of $\{F_\mathcal{G}^n(\emptyset)\}_{n\in\mathbb{N}}$. In the complete lattice $(\wp(\Sigma^*)^n, \subseteq^n)$, the supremum of $\{F_\mathcal{G}^n(\emptyset)\}_{n\in\mathbb{N}}$ is equal to $\bigcup_{n\in\mathbb{N}} F_\mathcal{G}^n(\emptyset)$. Thus $\operatorname{lfp} F_\mathcal{G} \sqsubseteq_{\ltimes} {F_\mathcal{G}}^{m}(\emptyset)$. $\square$

Algorithm. Given a regular language $M \subseteq \Sigma^*$ we say that a quasiorder $\ltimes \subseteq \Sigma^* \times \Sigma^*$ is M-*suitable* if it is 1) a wqo, 2) M-preserving, 3) monotonic and 4) decidable i.e., given two words u and v we can decide whether $u \ltimes v$. Intuitively a quasiorder is M-suitable if it can be used in our quasiorder-based framework to decide the inclusion problem $L(\mathcal{G}) \subseteq M$.

Algorithm 1: Algorithm for deciding $L(\mathcal{G}) \subseteq M$

```
Data: $\mathcal{G} = (V, P)$ CFG with $n$ variables.
Data: $M$-suitable quasiorder $\ltimes$.
Data: Procedure deciding $u \in M$ given $u$.
1 cur := $\boldsymbol{\emptyset}$;
2 repeat
3     prev := cur;
4     cur := $F_\mathcal{G}$(prev);
5 until cur $\sqsubseteq_{\ltimes^n}$ prev;
6 foreach $u \in (\text{prev})_1$ do
7     if $u \notin M$ then return false;
8 return true;
```

Theorem 7. *Algorithm 1 decides the inclusion problem $L(\mathcal{G}) \subseteq M$.*

Proof. As established by Proposition 6, given a monotonic decidable wqo $\ltimes$, lines 2 to 5 of Algorithm 1 compute, in finite time, a finite set equal to ${F_\mathcal{G}}^{m}(\boldsymbol{\emptyset})$ for some m such that $\operatorname{lfp} F_\mathcal{G} \sqsubseteq_{\ltimes^n} {F_\mathcal{G}}^{m}(\boldsymbol{\emptyset})$. Moreover, since ${F_\mathcal{G}}^{m}(\boldsymbol{\emptyset}) \subseteq \operatorname{lfp} F_\mathcal{G}$, we find that ${F_{\mathcal{G}}}^m(\emptyset) \sqsubseteq_{\ltimes} \operatorname{lfp} F_{\mathcal{G}}$. Since $\ltimes$ is M-preserving, Equation ($\star$) holds for ${F_{\mathcal{G}}}^m(\emptyset)_1$, i.e.

$$L(\mathcal{G}) \subseteq M \iff \forall u \in {F_{\mathcal{G}}}^m(\emptyset)_1,\ u \in M \ .$$

Thus the algorithm is correct, and it terminates because there are only finitely many membership checks $u \in M$ in the for-loop at lines 6 and 7. $\square$

Remark 8. The loop at lines 2 to 5 of the algorithm iteratively computes a vector of sets of words cur, updating its value to $F_{\mathcal{G}}(\text{prev})$ at line 4. The algorithm remains correct if we remove some words from prev before each new update, as long as the resulting vector prev is included in cur with respect to $\sqsubseteq_{\ltimes}$ (intuitively this ensures we do not lose any counter-examples). That is, we can replace the assignment line 3 by prev := cur' for any cur' $\subseteq$ cur such that cur $\sqsubseteq_{\ltimes}$ cur'; the correctness follows from Proposition 5.

4 Quasiorder Instantiation

We instantiate the algorithm with a quasiorder and give an example run. We then discuss other quasiorders with which the algorithm can be instantiated.

4.1 A State-based Quasiorder

We present an M-suitable quasiorder to instantiate Algorithm 1. The quasiorder is derived from a finite automaton with language M. Given an automaton $\mathcal{A} = (Q, q_I, \delta, F)$ with $L(\mathcal{A}) = M$ and a word $u \in \Sigma^*$, we define the *context* set of u

$$\operatorname{ctx}^{\mathcal{A}}(u) \triangleq \{(q, q') \in Q^2 \mid q \xrightarrow{u}{}^{*} q'\} \ .$$

We derive the following quasiorder on Σ^*:

$$u \leq^{\mathcal{A}}_{\text{ctx}} v \overset{\triangle}{\iff} \operatorname{ctx}^{\mathcal{A}}(u) \subseteq \operatorname{ctx}^{\mathcal{A}}(v) \ .$$

Proposition 9. *The quasiorder $\leq^{\mathcal{A}}_{ctx}$ is M-suitable.*

Proof. The result [13, Lemma 7.8] shows that $\leq^{\mathcal{A}}_{\text{ctx}}$ is a M-preserving, monotonic wqo. Since for every $u \in \Sigma^*$ we can compute the finite set $\operatorname{ctx}^{\mathcal{A}}(u)$, we deduce that given two words u and v we can decide $u \leq^{\mathcal{A}}_{\text{ctx}} v$. Thus, $\leq^{\mathcal{A}}_{\text{ctx}}$ is M-suitable. $\square$

Let M be the regular language a^*b^*. Consider the automaton $\mathcal{A}$ depicted in Fig. 1. It has two states p, q and accepts M, i.e. $L(\mathcal{A}) = M$. We give an execution of our algorithm instantiated with $\leq^{\mathcal{A}}_{\text{ctx}}$ to decide $L \subseteq M$, where L is the language recognized by the grammar $\mathcal{G}$ of Example 1, that is $L = L(\mathcal{G}) = \{a^n b^n \mid n \geq 0\}$.

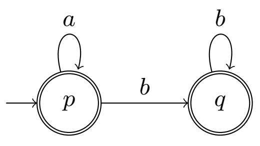

Fig. 1. Finite automaton $\mathcal{A}$ recognizing language a^*b^*.

We compute the contexts of the words appearing in the first iterations of $F_{\mathcal{G}}$, as computed in Example 2.

$$\begin{aligned} &\mathrm{ctx}^{\mathcal{A}}(\varepsilon) = \{(p,p),(q,q)\} && \mathrm{ctx}^{\mathcal{A}}(b) = \{(p,q),(q,q)\} \\ &\mathrm{ctx}^{\mathcal{A}}(a) = \{(p,p)\} && \mathrm{ctx}^{\mathcal{A}}(w) = \{(p,q)\}, \qquad \forall w \in a^+b^+ \ . \end{aligned}$$

The algorithm computes the first four iterations of $F_{\mathcal{G}}$ and then stops. We have $F^5_{\mathcal{G}}(\emptyset,\emptyset,\emptyset,\emptyset)_1 = \{a^2b^2, ab, \varepsilon\}$, $F^4_{\mathcal{G}}(\emptyset,\emptyset,\emptyset,\emptyset)_1 = \{ab,\varepsilon\}$, and the languages of the other component are the same. Since $ab \leq^{\mathcal{A}}_{\mathrm{ctx}} a^2b^2$, we have that $F^5_{\mathcal{G}}(\emptyset,\emptyset,\emptyset,\emptyset) \sqsubseteq_{(\leq^{\mathcal{A}}_{\mathrm{ctx}})^4} F^4_{\mathcal{G}}(\emptyset,\emptyset,\emptyset,\emptyset)$, hence $\mathrm{lfp}\, F_{\mathcal{G}} \sqsubseteq_{(\leq^{\mathcal{A}}_{\mathrm{ctx}})^4} F^4_{\mathcal{G}}(\emptyset,\emptyset,\emptyset,\emptyset)$. For each $u \in F^4_{\mathcal{G}}(\emptyset,\emptyset,\emptyset,\emptyset)_1$ we check if $u \in a^*b^*$. This is the case, and the algorithm returns `true`.

4.2 Other Quasiorders

The first kind of quasiorder we presented was a *state-based* quasiorder derived from a finite automaton for the language M. Here we present a *syntactic* quasiorder based on the syntactic structure of the language M. Given a word $u \in \Sigma^*$, we define the set

$$\mathrm{ctx}^M(u) \triangleq \{(w,w') \in \Sigma^* \times \Sigma^* \mid wuw' \in M\} \ .$$

We derive the following quasiorder on Σ^*:

$$u \leq^M_{\mathrm{ctx}} v \overset{\triangle}{\iff} \mathrm{ctx}^M(u) \subseteq \mathrm{ctx}^M(v) \ .$$

Following [5], we call this the Myhill order.

Proposition 10. *The quasiorder $\leq^M_{ctx}$ is M-suitable. Moreover, it is the coarsest among the M-suitable quasiorders.*

Proof. Since M is a regular language, by Lemma 7.6 (a) in [13] $\leq^{\mathcal{A}}_{\mathrm{ctx}}$ is M-suitable. By Lemma 7.6 (b) if $\ltimes$ is a M-suitable quasiorder then for every $u \in \Sigma^*$ we have $\uparrow_{\ltimes}\{u\} \subseteq \uparrow_{\leq^M_{\mathrm{ctx}}}\{u\}$, in other words $u \ltimes v$ implies $u \leq^M_{\mathrm{ctx}} v$. Hence, $\leq^M_{\mathrm{ctx}}$ is coarser than any M-suitable quasiorder. □

Notice that given two words u and v deciding $u \leq^M_{\mathrm{ctx}} v$ reduces to an inclusion between two regular languages, it is therefore PSPACE-complete to decide it. On the other hand deciding $u \leq^{\mathcal{A}}_{\mathrm{ctx}} v$ is PTIME.

5 Algorithmic Aspects

In Section 4.1 we considered the so-called state-based quasiorders for Algorithm 1. We can leverage state-based quasiorders even further and modify the algorithm to drop words entirely and replace them by their context. As we will see, contexts (that is sets of pairs of states) carry sufficient information to perform all the tasks required by the algorithm (namely comparison between words, updates via word concatenation and word membership tests).

5.1 A State-Based Variant

As a first step, consider a variant of Algorithm 1 where words are stored together with their context: instead of w we will store $(w, \mathrm{ctx}^{\mathcal{A}}(w))$. This change preserves the logic of the algorithm, hence its correctness. Observe that because the contexts are readily available they can be used directly for comparisons: instead of computing contexts each time two words need to be compared, the algorithm compares the already-computed contexts. Furthermore, and this is key, contexts can be updated during computations because they can be characterized inductively. Indeed, given $\mathrm{ctx}^{\mathcal{A}}(w)$ and $\mathrm{ctx}^{\mathcal{A}}(w')$, $\mathrm{ctx}^{\mathcal{A}}(ww')$ can be computed as follows:

$$\mathrm{ctx}^{\mathcal{A}}(ww') = \{(p,q) \mid \exists p' \colon (p,p') \in \mathrm{ctx}^{\mathcal{A}}(w) \wedge (p',q) \in \mathrm{ctx}^{\mathcal{A}}(w')\} \enspace .$$

In addition, membership queries in $M = L(\mathcal{A})$ have an equivalent counterpart on contexts: $w \in M$ iff $\mathrm{ctx}^{\mathcal{A}}(w) \cap (\{q_I\} \times F) \neq \emptyset$. As a consequence of the above, Algorithm 1 can drop words entirely and focus exclusively on contexts. We call this algorithm the antichains algorithm following a prolific history of such algorithms like [17] (for the CFG into REG case) and starting with [6] (for the REG into REG case).

Next, we will see a data-structure for contexts tailored to the case where $L(\mathcal{G})$ is a singleton. This data-structure, which leverages the automaton structure of regular languages, has been studied by Esparza, Rossmanith and Schwoon in a paper from 2000 [10]. Their paper list several potential use of the data-structure but not the one we are giving next.

5.2 A Data-Structure for the Case of Straight Line Programs

In 2000 Esparza, Rossmanith and Schwoon published a paper [10] in the EATCS bulletin where they give an algorithm which allows to solve a number of elementary problems on context-free grammars including identifying useless variables, and deciding emptiness or finiteness. The algorithm is a "saturation" algorithm that takes as input a finite state automaton $\mathcal{A} = (Q, \delta, q_I, F)$ and a context-free grammar $\mathcal{G} = (V, P)$ on an alphabet Σ. It adds transitions to $\mathcal{A}$ so that, upon saturation, its resulting language corresponds to the following set of words on alphabet $(V \cup \Sigma)$:

$$\{\alpha \in (V \cup \Sigma)^* \mid \exists w \in L(\mathcal{A}) \colon \alpha \Rightarrow^* w\} \enspace .$$

This set, which they denote as $\mathrm{pre}^*(L(\mathcal{A}))$, comprises the sentences of the grammar $\mathcal{G}$ which, upon application of zero or more production rules, becomes a word in $L(\mathcal{A})$. At the logical level, the algorithm consists of one saturation rule and nothing more which states that if $X \to \beta \in P$ and $q \xrightarrow{\beta}{}^* q'$ then add a transition $q \xrightarrow{X} q'$ to $\mathcal{A}$.

We show that this approach is relevant to our problem. Let us consider the inclusion problem CFG into REG where we have the restriction that for every $i \in [1, n]$ we have that $L(\mathcal{G})_i$ is either a singleton word or it is empty: $L(\mathcal{G})_i \neq \emptyset \implies L(\mathcal{G})_i \in \Sigma^*$. Such grammars are called straight line programs (SLP) in the literature. As we have shown in the past [14], the inclusion problem for straight line programs has direct applications in regular expression matching for text compressed using grammar-based techniques. Intuitively, the regular expression is translated into $L(\mathcal{A})$ and the compressed text is given by $L(\mathcal{G})$. Then we have that the regular expression has a match in the decompressed text iff $L(\mathcal{G}) \subseteq L(\mathcal{A})$.

In the context of our inclusion algorithm, the assumption that $\mathcal{G}$ is a straight line program ensures that, for every m, each entry in the vector $F^m_{\mathcal{G}}(\vec{\emptyset})$ is either the empty set or a singleton word. Now consider the state-based variant of Algorithm 1 discussed above where words are replaced by their contexts. Because of the SLP assumption we have that each entry of the cur and prev vectors in Algorithm 1 stores a set of pairs of states: each entry is given by a set $P \subseteq Q^2$ as opposed to $P \subseteq \wp(Q^2)$ when $\mathcal{G}$ need not be a SLP.

A possible data-structure to store such a set P is a graph where nodes are given by Q and whose edges coincides with the pairs of states of P. Add labels on edges and you can encode a vector of sets like P, namely an edge from q to q' with label m encodes that the pair (q, q') belongs to the mth component of the vector. Next, observe that the updates to be carried out on the vectors cur and prev via $F_{\mathcal{G}}$ as prescribed by Algorithm 1 can be implemented via the saturation rule described above. For instance, assume $F_{\mathcal{G}}$ requires to update the m-th entry of cur following the production rule $X_m \to X_k\, X_{k'}$. The corresponding update to the graph-based data structure adds an edge labelled m between states (q, q') if there exists a node q'' such that (q, q'') and (q'', q') are two edges respectively labelled by k and k'. It is worth pointing that this update rule coincides with the above saturation rule for the rule $X_m \to X_k\, X_{k'}$. Therefore the saturation rule together with the underlying graph data-structure leveraging the automaton $\mathcal{A}$ enable a state-based implementation of Algorithm 1 tailored to the case SLP into REG. It is worth pointing out that Esparza et al. [10] work out the precise time and space complexity of implementing the saturation rule. Ultimately, we obtain an algorithm in PTIME for the SLP into REG inclusion problem. A more detailed complexity analysis can be found in Valero's PhD thesis [22, Chapter 5].

6 Other Language Inclusion Problems

In this section we talk about the other language classes to which our framework can be applied.

6.1 REG $\subseteq$ REG

We consider the case where $\mathcal{G}$ is a right regular grammar i.e. a grammar in which all production rules are of the form $X_j \to X_k\, a$, $X_j \to a$, or $X_j \to \varepsilon$ where X_j and X_k are variables and $a \in \Sigma$. The function $F_{\mathcal{G}}$ of the least fixpoint characterization of $L(\mathcal{G})$ is now defined as follows. Let $\mathcal{L} = (L_1, \ldots, L_n) \in \wp(\Sigma^*)^n$ define:

$$F_{\mathcal{G}}(\mathcal{L})_j \triangleq \bigcup_{X_j \to X_k\, a \in P} L_k\{a\} \cup \bigcup_{a \in \Sigma \cup \{\varepsilon\},\ X_j \to a \in P} \{a\} \ .$$

The same algorithm with this new $F_{\mathcal{G}}$ is also correct. Notice however that for Proposition 5 and Proposition 6 to be correct it is now sufficient for $\ltimes$ to only be right-monotonic. This is because the new $F_{\mathcal{G}}$ only uses right-concatenations in its production rules. As a consequence of this, we can relax $\leq^{\mathcal{A}}_{\text{ctx}}$ into the coarser quasiorder $\leq^{\mathcal{A}}_{\text{post}}$ where, we just replace the context ctx of a word u by the set post containing all the states reachable with the word u from the initial state of $\mathcal{A}$, that is, $\{q \mid q_I \xrightarrow{u}{}^* q'\}$. Similarly, we can relax $\leq^{M}_{\text{ctx}}$ into the Nerode quasiorder of M. In the Nerode quasiorder, a word u subsumes v if, for every $w \in \Sigma^*$, $uw \in M$ implies $vw \in M$.

6.2 REG $\subseteq$ Petri Net Traces

In 1999 Petr Jančar, Javier Esparza and Faron Moller [19] published a paper which solved several language inclusion problems including the problem asking whether the trace set of a finite-state labelled transition system is included in the trace set of a Petri net (which defines a labelled transition system with possibly infinitely many states). In this section we give a decision procedure following the framework we put in place in the previous sections. As we will see our decision procedure is very close yet slightly different from the one they proposed.

It is worth pointing that their exposition favors a process theory viewpoint rather than an automata theory viewpoint which is why they talk about trace sets and labelled transition systems (or processes) instead of languages and acceptors. With this view in mind, the notion of alphabet Σ becomes a set of *actions* in which they include a distinguished *silent* action τ. The τ resembles the ε symbol used in the automata theoretic setting. In their paper, they define and prove correct an algorithm to decide whether the trace set given by a finite-state labelled transition system is included into the trace set given by a Petri net.

Our first step is to extract from their paper an ordering on words that is M-suitable. In our setting, M is the trace set given by a Petri net. The ordering is a "state-based" ordering defined upon the underlying Petri net. The definition relies on the notion of ω-markings which, intuitively, are vectors with as many entries as there are places in the Petri net and such that each entry takes its value in $\mathbb{N} \cup \{\omega\}$. Let P be the set of places of the Petri net, then $\mathbb{N}^P_\omega$ denote the set of ω-markings thereof. Given two ω-markings $\hat{M}$ and $\hat{M}'$ they define (p. 481) an ordering $\leqslant$ between them such that $\hat{M} \leqslant \hat{M}'$ if $\hat{M}(p) \leqslant \hat{M}'(p)$ for all $p \in P$

such that ω is greater than every natural. They then lift the ordering $\leqslant$ to sets of ω-markings as follows. Given two sets $\mathcal{M}, \mathcal{M}' \subseteq \mathbb{N}^P_\omega$ define $\mathcal{M} \leqslant \mathcal{M}'$ iff for each $M \in \mathcal{M}$ there exists $M' \in \mathcal{M}'$ such that $M \leqslant M'$. This relation is a quasiorder on sets of ω-marking which becomes a well-quasiorder when restricted to the finite sets of ω-markings. That is $\leqslant$ is a wqo on $\wp_{\text{fin}}(\mathbb{N}^P_\omega)$ (their Corollary 2.17).

We are now in position to define our ordering on words (what they call traces). Let $u, v \in (\Sigma \setminus \{\tau\})^*$ define

$$u \lessdot v \text{ iff } \max(\mathcal{C}(\overset{u}{\Rightarrow}(\{M_0\}))) \leqslant \max(\mathcal{C}(\overset{v}{\Rightarrow}(\{M_0\}))) \enspace .$$

Let us discuss this definition starting with the expressions $\max(\mathcal{C}(\overset{u}{\Rightarrow}(\{M_0\})))$ which denote finite sets of ω-markings (i.e. $\max(\mathcal{C}(\overset{u}{\Rightarrow}(\{M_0\}))) \in \wp_{\text{fin}}(\mathbb{N}^P_\omega)$ so that the ordering $\leqslant$ is the well-quasiorder described above). The details of the definition $\max(\mathcal{C}(\overset{u}{\Rightarrow}(\{M_0\})))$ would take too much space to include here but intuitively the finite set of ω-markings is a finite description of the possibly infinitely many markings (which are ω-markings with no entry set to ω) the Petri net can reach guided by the sequence of actions in u from its initial marking M_0 (the $\mathcal{C}$ and the max operator turn a possibly infinite sets of markings into a finite set of ω-markings).

Let us now turn to the M-suitability of $\lessdot$. As we have already shown the ordering $\lessdot$ is a wqo. Second it is routine to check that monotonicity follows from their Lemma 2.5(a) and Lemma 2.9 [19]. Indeed Lemma 2.5(a) states that if $\mathcal{M} \leqslant \mathcal{M}'$ then $\overset{a}{\Rightarrow}(\mathcal{M}) \leqslant \overset{a}{\Rightarrow}(\mathcal{M}')$ for every $a \in \Sigma \setminus \{\tau\}$; while Lemma 2.9 states that $\mathcal{M} \leqslant \mathcal{M}'$ then $\max(\mathcal{C}(\mathcal{M})) \leqslant \max(\mathcal{C}(\mathcal{M}'))$.

Next we show that M-preservation holds by simply observing that w is a trace of the Petri net iff $\overset{w}{\Rightarrow}(\{M_0\}) \neq \emptyset$, hence that if $u \lessdot v$ and u is a trace then v is a trace by definition of $\lessdot$, their Lemma 2.7(a) showing $\mathcal{M} \subseteq \mathcal{C}(\mathcal{M})$, and the definition of max (p. 482).

It remains to show that $\lessdot$ is decidable which is easily established by first using the result of their Lemma 2.15 stating that we can effectively construct $\max(\mathcal{C}(\overset{a}{\Rightarrow}(\{\mathcal{M}\})))$ given a finite set $\mathcal{M}$ of ω-markings and an action $a \in \Sigma$; and second by using the result of their Lemma 2.14 to lift (inductively) their effectivity result from actions to finite sequences of actions. Notice that this effectivity result for finite sequences of actions also gives a decision procedure for the membership in the trace set of the Petri net.

With all the ingredients in place, we obtain an algorithm deciding whether the trace set of a finite-state labelled transition system is included into the trace set of a Petri net. It is worth pointing out that their algorithm [19, Theorem 3.2] performs less comparisons of finite sets of ω-markings than ours. A tree structure restricts the comparisons they do during the exploration to pairs of sets of ω-markings, provided one is the ancestor (w.r.t. tree structure) of the other. Since our algorithm does not have such a restriction, we claim that we are making at least as many comparisons as their algorithm. This potentially results in a shorter execution time and less membership checks in our algorithm.

6.3 Languages of Infinite Words

In this section we give the idea on how we extended the framework to the inclusion problem between languages of infinite words [9]. In particular we studied the inclusion between two ω-regular languages and the inclusion of an ω-context-free language into an ω-regular language.

Consider the inclusion problem $L^\omega(\mathcal{A}) \subseteq M$ where $\mathcal{A}$ is a BA and M is an ω-regular language. The first step is to reduce the inclusion check to the ultimately periodic words of the languages, as they suffice to decide the inclusion [3]. These are infinite words of the form uv^ω for $u \in \Sigma^*$ a finite prefix and $v \in \Sigma^+$ a finite period. We compare two ultimately periodic words using a pair of quasiorders $\ltimes_1$ and $\ltimes_2$ on finite words. The first quasiorder, $\ltimes_1$, compares the prefixes of ultimately periodic words, while the second quasiorder, $\ltimes_2$, compares their periods. As expected the quasiorders need to be M-preserving right-monotonic decidable wqos. In this setting M-preservation means that if $uv^\omega \in M$ and $(u, v) \ltimes_1 \times \ltimes_2 (u', v')$ then $u'v'^\omega \in M$. For example, by taking $\ltimes_1$ to be $\leq^{\mathcal{B}}_{post}$ and $\ltimes_2$ to be $\leq^{\mathcal{B}}_{\text{ctx}}$, where $\mathcal{B}$ is a BA such that $L^\omega(\mathcal{B}) = M$, we have an M-preserving pair of right-monotonic decidable wqos.

An M-preserving pair of wqos ensures the existence of a finite subset $S = S_1 \times S_2$ of ultimately periodic words that is sufficient to decide the inclusion. In order to compute such a subset S we establish a fixpoint characterization for the sets of prefixes and periods of L. Finally, we decide the inclusion by checking membership in M for every ultimately periodic word uv^ω such that $(u, v) \in S$.

In the case where $\mathcal{A}$ is a Büchi Pushdown (and therefore where $L^\omega(\mathcal{A})$ is an ω-context-free language), the only difference lies in the fixpoint functions which now use both right and left concatenations. Thus the pair of quasiorders in this case should be monotonic. For example we can take both orders to be $\leq^{\mathcal{B}}_{\text{ctx}}$.

6.4 Pointers to Other Cases

In our paper [7], the approach is adapted to the inclusion problem between Visibly Pushdown Languages (VPL) (of finite words) and ω-VPL (of infinite words), classes of languages defined in [1]. In these cases, defining monotonicity conditions is challenging.

In the work by Henzinger et al. [16], the authors adapt our framework to solve their inclusion problem between operator-precedence languages [11]. These languages fall within a class that is strictly contained in deterministic context-free languages and, in turn, strictly contains VPL [4].

In [8], we further extend the framework for solving the inclusion $L^\omega(\mathcal{A}) \subseteq M$, for $\mathcal{A}$ an BA and M an ω-regular language, using a family of quasiorders instead of a pair of quasiorders. A family of quasiorders allows more pruning when searching for a counterexample, thus lesser membership queries at the end.

As we show in [13] it is also possible to define an M-suitable quasiorder leveraging pre-computed simulation relations on the states of an automaton for M. This quasiorder might be coarser than the state-based one presented in Section 4.1.

Some directions for future work are the inclusion of a context-free language into a superdeterministic context-free language [15], and the inclusion between tree automata. It is worth noting that least fixpoint characterizations exist for a very large number of language classes [18]. Eventually, we want to explore whether our approach can be adapted to the emptiness problem of alternating automata for finite and infinite words. The PhD thesis of Nicolas Maquet [21] suggests it can be done. In [2], Bonchi and Pous compare antichains and bisimulation up-to techniques. We want to extend this comparison, given the novel insights of our approach.

Acknowledgments. Pierre visited Javier during his first year of PhD, a visit that turned out to be a milestone in Pierre's career and has had influence up to this day. This visit also got Pierre a new colleague, a mentor and, most importantly, a friend. Pierre wishes to thank Javier from the bottom of his heart for all the good memories throughout the years. Chana is grateful to have had Javier as a PhD advisor and as an academic role model. She learned a lot from him, and benefited from the kind and studious atmosphere that he has established in his Chair at the Technical University of Munich. We also are thankful to the reviewers for their valuable feedback. This publication is part of the grant PID2022-138072OB-I00, funded by MCIN, FEDER, UE and has been partially supported by PRODIGY Project (TED2021-132464B-I00) funded by MCIN and the European Union NextGeneration.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC'04: Proc. 36th Ann. ACM Symp. on Theory of Computing. ACM (2004). https://doi.org/10.1145/1007352.1007390
2. Bonchi, F., Pous, D.: Checking NFA equivalence with bisimulations up to congruence. In: POPL'13: Proc. of the 40th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. Association for Computing Machinery (2013). https://doi.org/10.1145/2429069.2429124
3. Calbrix, H., Nivat, M., Podelski, A.: Ultimately periodic words of rational ω-languages. In: MFPS'94: Proc. Int. Symp. on Mathematical Foundations of Programming Semantics. LNCS, Springer (1994). https://doi.org/10.1007/3-540-58027-1_27
4. Crespi Reghizzi, S., Mandrioli, D.: Operator precedence and the visibly pushdown property. Journal of Computer and System Sciences **78**(6) (2012). https://doi.org/10.1016/j.jcss.2011.12.006
5. de Luca, A., Varricchio, S.: Well quasi-orders and regular languages. Acta Informatica **31**(6) (1994). https://doi.org/10.1007/BF01213206
6. De Wulf, M., Doyen, L., Henzinger, T.A., Raskin, J.F.: Antichains: A new algorithm for checking universality of finite automata. In: Computer Aided Verification. LNCS, Springer (2006). https://doi.org/10.1007/11817963_5
7. Doveri, K., Ganty, P., Hadži-Đokić, L.: Antichains Algorithms for the Inclusion Problem Between ω-VPL. In: TACAS'23: Proc. 29th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, Part I. LNCS, Springer (2023). https://doi.org/978-3-031-30823-9_15

8. Doveri, K., Ganty, P., Mazzocchi, N.: FORQ-Based Language Inclusion Formal Testing. In: CAV'22: Proc. 32nd Int. Conf. on Computer Aided Verification. Springer (2022). https://doi.org/10.1007/978-3-031-13188-2_6
9. Doveri, K., Ganty, P., Parolini, F., Ranzato, F.: Inclusion Testing of Büchi Automata Based on Well-Quasiorders. In: CONCUR'21: Proc. 32nd Int. Conf. on Concurrency Theory. LIPIcs, Schloss Dagstuhl (2021). https://doi.org/10.4230/LIPIcs.CONCUR.2021.3
10. Esparza, J., Rossmanith, P., Schwoon, S.: A Uniform Framework for Problems on Context-Free Grammars. EATCS BULLETIN **72** (2000), can be found at https://archive.model.in.tum.de/um/bibdb/esparza/ufpcfg.pdf
11. Floyd, R.W.: Syntactic analysis and operator precedence. J. ACM **10**(3) (1963). https://doi.org/10.1145/321172.321179
12. Gallier, J.: Languages, automata, theory of computation, preprint on webpage at https://www.cis.upenn.edu/~jean/gbooks/toc.pdf
13. Ganty, P., Ranzato, F., Valero, P.: Complete abstractions for checking language inclusion. ACM Trans. Comput. Logic **22**(4) (2021). https://doi.org/10.1145/3462673
14. Ganty, P., Valero, P.: Regular Expression Search on Compressed Text. In: 2019 Data Compression Conference (DCC). IEEE (2019). https://doi.org/10.1109/DCC.2019.00061
15. Greibach, S.A., Friedman, E.P.: Superdeterministic PDAs: A Subcase with a Decidable Inclusion problem. Journal of the ACM **27**(4) (1980). https://doi.org/10.1145/322217.322224
16. Henzinger, T.A., Kebis, P., Mazzocchi, N., Saraç, N.E.: Regular Methods for Operator Precedence Languages. In: ICALP'23: Proc. 50th Int. Coll. on Automata, Languages, and Programming. LIPIcs, Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPIcs.ICALP.2023.129
17. Holík, L., Meyer, R.: Antichains for the Verification of Recursive Programs. In: NETYS'15: Networked Systems. LNCS, Springer (2015). https://doi.org/10.1007/978-3-319-26850-7_22
18. Istrail, S.: Generalization of the ginsburg-rice schützenberger fixed-point theorem for context-sensitive and recursive-enumerable languages. Theoretical Computer Science **18**(3) (1982)
19. Jančar, P., Esparza, J., Moller, F.: Petri Nets and Regular Processes. Journal of Computer and System Sciences **59**(3) (Dec 1999). https://doi.org/10.1006/jcss.1999.1643
20. Kasai, T., Iwata, S.: Some problems in formal language theory known as decidable are proved EXPTIME complete (1992), https://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/0796-02.pdf
21. Maquet, N.: New algorithms and data structures for the emptiness problem of alternating automata. Ph.D. thesis, Université Libre de Bruxelles, Belgium (2011), http://hdl.handle.net/2013/ULB-DIPOT:oai:dipot.ulb.ac.be:2013/209961
22. Valero Mejía, P.: On the use of quasiorders in formal language theory. Ph.D. thesis, Universidad Politecnica de Madrid - University Library (2020). https://doi.org/10.20868/upm.thesis.64477