Inclusion Testing of Büchi Automata Based on Well-Quasiorders

Kyveli Doveri
IMDEA Software Institute, Madrid, Spain
Universidad Politécnica de Madrid, Spain

Pierre Ganty
IMDEA Software Institute, Madrid, Spain

Francesco Parolini
Sorbonne Université, Paris, France

Francesco Ranzato
University of Padova, Italy

Abstract

We introduce an algorithmic framework to decide whether inclusion holds between languages of infinite words over a finite alphabet. Our approach falls within the class of Ramsey-based methods and relies on a least fixpoint characterization of ω-languages leveraging ultimately periodic infinite words of type uv^{ω}, with u a finite prefix and v a finite period of an infinite word. We put forward an inclusion checking algorithm between Büchi automata, called BAInc, designed as a complete abstract interpretation using a pair of well-quasiorders on finite words. BAInc is quite simple: it consists of two least fixpoint computations (one for prefixes and the other for periods) manipulating finite sets (of pairs) of states compared by set inclusion, so that language inclusion holds when the sets (of pairs) of states of the fixpoints satisfy some basic conditions. We implemented BAInc in a tool called BAIT that we experimentally evaluated against the state-of-the-art. We gathered, in addition to existing benchmarks, a large number of new case studies stemming from program verification and word combinatorics, thereby significantly expanding both the scope and size of the available benchmark set. Our experimental results show that BAIT advances the state-of-the-art on an overwhelming majority of these benchmarks. Finally, we demonstrate the generality of our algorithmic framework by instantiating it to the inclusion problem of Büchi pushdown automata into Büchi automata.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Büchi (Pushdown) Automata, ω-Language Inclusion, Well-quasiorders

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.3

Supplementary Material *Software (Source Code)*: `https://github.com/parof/bait`
archived at `swh:1:dir:0085d9456cf1300c44386339a996c9453dca17c2`

Funding *Pierre Ganty*: Partially supported by the Madrid regional project S2018/TCS-4339 BLOQUES, the Spanish project PGC2018-102210-B-I00 BOSCO and the Ramón y Cajal fellowship RYC-2016-20281.
Francesco Ranzato: Partially funded by *University of Padova*, under the SID2018 project "Analysis of STatic Analyses (ASTA)"; *Italian Ministry of University and Research*, under the PRIN2017 project no. 201784YSZ5 "AnalysiS of PRogram Analyses (ASPRA)"; *Facebook Research*, under a "Probability and Programming Research Award".

Acknowledgements We thank: Richard Mayr, Lorenzo Clemente, Parosh Abdulla and Lukáš Holík for their insights and help with RABIT; Ming-Hsien Tsai for his help with GOAL; Reed Oei for the Pecan benchmarks; Matthias Heizmann and Daniel Dietsch for the Ultimate Automizer benchmarks.

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 3; pp. 3:1–3:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Deciding whether a formal language contains another one is a fundamental problem with diverse applications ranging from automata-based verification to compiler construction [6, 13, 25, 42]. In this work, we deal with the inclusion problem for ω-languages, namely languages of words of infinite length (ω-words) over a finite alphabet. In particular, we are interested in the case of ω-regular languages, which is known to be PSPACE-complete [26], and in the inclusion of ω-context-free languages into ω-regular, which is known to be EXPTIME-complete [21, 32].

1.1 Main Contributions

We put forward a number of language inclusion algorithms that are systematically designed from an abstraction-based perspective of the inclusion problem. Our starting point was a recent abstract interpretation-based algorithmic framework for the inclusion problem for languages of finite words [15, 16]. Extending this framework to ω-words raises several challenges. First, the finite word case crucially relies on least fixpoint characterizations of languages which we are not aware of for languages of ω-words (while greatest fixpoint characterizations exist). The second challenge is to define suitable abstractions for languages of ω-words and effective representations thereof.

We overcome the first challenge by reducing the inclusion problem for ω-languages to an equivalent inclusion problem between their so-called *ultimately periodic* subsets. The ultimately periodic subset of an ω-language L consists of those ω-words of the form $uv^\omega \in L$, where u and v are finite words referred to as, resp., a *prefix* and a *period* of an ω-word. It turns out that an underlying Büchi (pushdown) automaton accepting L enables a least fixpoint characterization of the ultimately periodic subset of L. To guarantee convergence in finitely many Kleene iterations of such a least fixpoint, we resort to a conceptually simple approach based on abstract interpretation [8]. Roughly speaking, we define over-approximating abstractions of sets of finite words which "enlarge" these sets with new words picked according to a quasiorder relation on finite words. Our abstractions rely on two distinct quasiorder relations which, resp., enlarge the sets of prefixes and periods of an ultimately periodic set representing an ω-language. The quasiorders inducing our abstractions have to satisfy two basic properties: (1) to be well-quasiorders to guarantee finite convergence of fixpoint computations; (2) some monotonicity conditions w.r.t. word concatenation in order to yield a sound and complete inclusion algorithm (soundness holds for mere quasiorders). Once the abstract least fixpoint has been computed, an inclusion check $L \subseteq^? M$ reduces to a finite number of tests $uv^\omega \in^? M$ for finitely many prefixes u and periods v taken from the abstract least fixpoint representing L. We introduce different well-quasiorders to be used in our inclusion algorithm and we show that using distinct well-quasiorder-based abstractions for prefixes and periods pays off.

For a language inclusion check $L \subseteq M$, where L and M are accepted by Büchi automata, some quasiorders enable a further abstraction step where finite words are abstracted by states relating these words in the underlying Büchi automaton accepting M, and this correspondingly defines a purely "state-based" inclusion algorithm that operates on automaton states only. We further demonstrate the generality of our algorithmic framework by instantiating it to the inclusion problem of Büchi pushdown automata into Büchi automata.

We implemented our language inclusion algorithm in a tool called BAIT (Büchi Automata Inclusion Tester) [10]. We put together an extensive suite of benchmarks [11], notably verification tasks as defined by the RABIT tool [1, 2], logical implication tasks in word combinatorics as defined by the Pecan theorem prover [34], and termination tasks as defined by Ultimate Automizer [18]. We conducted an experimental comparison of BAIT against some state-of-the-art language inclusion checking tools: GOAL [41], HKCω [24], RABIT [37, 7] and ROLL [27]. The experimental results show that BAIT advances the state-of-the-art of the tools for checking inclusion of ω-languages on an overwhelming majority of benchmarks.

1.2 Related Works

Due to space constraints, we limit our discussion to Ramsey-based algorithms, as our inclusion procedure, and to methods based on automata complementation. Kuperberg et al. [24] also reduce the language equivalence problem over Büchi automata to that of their ultimately periodic subsets. A further commonality is that the algorithm of [24] handles prefixes and periods differently: for the prefixes they leverage a state-of-the-art up-to congruence algorithm [3], while up-to congruences are not used for the periods[1]. Fogarty and Vardi [14] for the universality problem, and later Abdulla et al. [1, 2] for the inclusion problem between languages accepted by Büchi automata, all reduce their decision problems to the ultimately periodic subsets. Their approach is based on a partition of nonempty words whose blocks are represented and manipulated through so-called supergraphs. The equivalence relation underlying their partition can be obtained from one of our quasiorders. Moreover, by equipping their supergraphs with a subsumption order [2, Def. 6], they define a relation which coincides with one of our quasiorders. Hofmann and Chen [20], whose approach based on abstract interpretation inspired our work, also tackle the inclusion problem for ω-languages. They construct an abstract (finite) lattice using the same equivalence relation which is derived from a given Büchi atomaton, and define a Galois connection between it and the (infinite) lattice of languages of infinite words. However, they do not relax this relation into a quasiorder. Finally, the complementation-based approaches reduce language inclusion to a language emptiness check by using intersection and an explicit complementation of a Büchi automaton. Despite that there are Büchi automata of size n whose complement cannot be represented with less than $n!$ states [33], algorithms to complement Büchi automata have been defined, implemented and are effective in practice [40]. In our approach, explicit complementation is avoided altogether.

2 Overview

We assume familiarity with the basics of language theory (see, e.g., [22, 35]). Throughout the paper, we fix Σ to be a finite nonempty alphabet. Furthermore, let ϵ denote the empty word, Σ^* the set of finite words over Σ, $\Sigma^+ \triangleq \Sigma^* \setminus \{\epsilon\}$, Σ^ω the set of infinite words (or ω-words) over Σ, $|w| \in \mathbb{N}$ denote the length of $w \in \Sigma^*$. The *ultimately periodic words* are the words $\xi \in \Sigma^\omega$ such that $\xi = uv^\omega$ for some finite *prefix* $u \in \Sigma^*$ and some finite *period* $v \in \Sigma^+$. Given $L \subseteq \Sigma^\omega$, we associate pairs of finite words to ultimately periodic words and define

$$I_L \triangleq \{(u, v) \in \Sigma^* \times \Sigma^+ \mid uv^\omega \in L\} \ .$$

In the following we give an outline of our approach. Given two ω-languages L and M such that the inclusion check reduces to that of their ultimately periodic words, i.e. $L \subseteq M \Leftrightarrow I_L \subseteq I_M$ holds, we reduce the inclusion problem $L \subseteq M$ to finitely many membership queries in the candidate "larger" language M.

[1] In the technical report thereof, the authors work out up-to union and up-to equivalence reasoning for periods but not their combination (up-to congruence).

A quasiorder (qo) relation on a set S is a reflexive and transitive binary relation on S. Any qo $\leq \subseteq S \times S$ induces a map $\rho_{\leq} : \wp(S) \to \wp(S)$ defined by $\rho_{\leq}(X) \triangleq \{y \in S \mid \exists x \in X,\, x \leq y\}$, which turns out to be a closure operator on the complete lattice $\langle \wp(S), \subseteq \rangle$. Let us recall that a closure operator is a monotone ($X \subseteq X' \Rightarrow \rho(X) \subseteq \rho(X')$), idempotent ($\rho(X) = \rho(\rho(X))$), and increasing ($X \subseteq \rho(X)$) map. Given $X \in \wp(S)$, the set $\rho_{\leq}(X)$ is called the *upward closure* of X w.r.t. $\leq$. We say that a qo relation $\preceq$ on $\Sigma^* \times \Sigma^+$ *preserves* I_M if $\rho_{\preceq}(I_M) = I_M$ holds. Given a qo $\preceq$ that preserves I_M, since $\rho_{\preceq}$ is monotone and increasing, we have that:

$$L \subseteq M \iff I_L \subseteq I_M \iff \rho_{\preceq}(I_L) \subseteq I_M \quad . \tag{1}$$

A qo $\leq$ is a well-quasiorder (wqo) if for any upward closure $\rho_{\leq}(X)$ there is a finite subset $X' \subseteq_{\text{fin}} X$ such that $\rho_{\leq}(X) = \rho_{\leq}(X')$. Hence, if a relation $\preceq$ on $\Sigma^* \times \Sigma^+$ is a wqo then there exists a finite subset $T \subseteq_{\text{fin}} I_L$ such that $\rho_{\preceq}(T) = \rho_{\preceq}(I_L)$. By exploiting the properties of closures, this reduces the inclusion check to finitely many membership queries in M:

$$L \subseteq M \iff \rho_{\preceq}(T) \subseteq I_M \iff T \subseteq I_M \iff \forall (u,v) \in T,\, uv^\omega \in M \quad . \tag{2}$$

Following this approach, we design inclusion algorithms in the cases where both languages L and M are ω-regular and where the "left" language L is ω-context-free and the "right" language M is ω-regular. In Section 3, we define wqos that preserve I_M as required by (1). Section 4 gives a detailed account of each step so as to end up designing our inclusion algorithms. Section 5 shows how to obtain algorithms deciding $L(\mathcal{A}) \subseteq L(\mathcal{B})$ and $L(\mathcal{P}) \subseteq L(\mathcal{B})$, where $\mathcal{A}$ and $\mathcal{B}$ are Büchi automata and $\mathcal{P}$ is a Büchi pushdown automaton, by reasoning exclusively on the automata states/configurations. Section 6 describes the experimental results of our implementation BAIT.

3 Well-Quasiorders for ω-Regular Languages

The equivalence (1) holds because the qo $\preceq$ on $\Sigma^* \times \Sigma^+$ is such that $\rho_{\preceq}(I_M) = I_M$. In the following, we focus on pairs of qos $\leq$ and $\preccurlyeq$ on, resp., Σ^* and Σ^+, such that their product relation $\leq \times \preccurlyeq$ on $\Sigma^* \times \Sigma^+$ preserves I_M, i.e., $\rho_{\leq\times\preccurlyeq}(I_M) = I_M$ holds. We define different pairs of qos preserving I_M and show how they compare. All these qos are *well-quasiorders* and *right-monotonic*. The first property guarantees the existence of a finite representation for I_L and the convergence after finitely many steps of the fixpoint computations, while the second property ultimately yields a (sound and) complete inclusion algorithm.

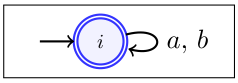
A qo $\leq$ on Σ^* is *left-monotonic* (*right-monotonic*) if

$$\forall u, v, w \in \Sigma^*,\, u \leq v \;\Rightarrow\; wu \leq wv \;\; (uw \leq vw) \enspace ,$$

while $\leq$ is monotonic if it is both left- and right-monotonic. Given any relation $R \subseteq X \times X$, $R^* \triangleq \bigcup_{n \in \mathbb{N}} R^n$ denotes its reflexive and transitive closure.

A Büchi automaton (BA) on an alphabet Σ is a tuple $\mathcal{A} = (Q, \delta, i, F)$ where Q is a finite set of states including a unique initial state $i \in Q$, $\delta \colon Q \times \Sigma \to \wp(Q)$ is a transition function, and $F \subseteq Q$ is a subset of final states. We write a transition $q \xrightarrow{a} q'$ when $q' \in \delta(q, a)$ and lift this relation to finite words by transitive and reflexive closure, thus writing $q \xrightarrow{u}{}^{*} q'$ with $u \in \Sigma^*$. We write $q \xrightarrow{u}_{F}{}^{*} q'$ if there exists $q_f \in F$ and $u_1, u_2 \in \Sigma^*$ such that $q \xrightarrow{u_1}{}^{*} q_f$, $q_f \xrightarrow{u_2}{}^{*} q'$ and $u = u_1u_2$. The language of finite words accepted by $\mathcal{A}$ is $L^*(\mathcal{A}) \triangleq \{u \in \Sigma^* \mid i \xrightarrow{u}{}^{*} q,\, q \in F\}$. A trace of $\mathcal{A}$ on an ω-word $w = a_0a_1 \cdots \in \Sigma^\omega$ is an infinite sequence $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \cdots$, which is called initial when $q_0 = i$ and fair when $q_j \in F$ for infinitely many j's. The ω-language accepted by $\mathcal{A}$ is $L^\omega(\mathcal{A}) \triangleq \{w \in \Sigma^\omega \mid \text{there exists an initial and fair trace on } w\}$. An ω-language $L \subseteq \Sigma^\omega$ is *ω-regular* if $L = L^\omega(\mathcal{A})$ for some BA $\mathcal{A}$.

(a) $\mathcal{C}$



(b) $\mathcal{D}$

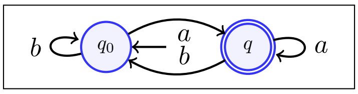


Figure 1 Büchi automata $\mathcal{C}$ and $\mathcal{D}$ over the alphabet $\Sigma = \{a, b\}$.

State-Based Quasiorders. We define quasiorders that compare words in Σ^* based on the states of a BA $\mathcal{A} = (Q, \delta, i, F)$. To do so, we associate with each word $u \in \Sigma^*$ its *context* $c^{\mathcal{A}}[u] \subseteq Q^2$ and *final context* $f^{\mathcal{A}}[u] \subseteq Q^2$ in $\mathcal{A}$ as follows:

$$c^{\mathcal{A}}[u] \triangleq \{(q, q') \in Q^2 \mid q \xrightarrow{u}{}^{*} q'\} \ ,$$
$$f^{\mathcal{A}}[u] \triangleq \{(q, q') \in Q^2 \mid q \xrightarrow{u}_F{}^{*} q'\} \ .$$

We also define the *successor set* $s^{\mathcal{A}}[u] \subseteq Q$ in $\mathcal{A}$ through a word $u \in \Sigma^*$ as follows:

$$s^{\mathcal{A}}[u] \triangleq \{q \in Q \mid i \xrightarrow{u}{}^{*} q\} \ .$$

Based on this, we define the following qos on words in Σ^*:

$$u \leq^{\mathcal{A}} v \stackrel{\triangle}{\Leftrightarrow} s^{\mathcal{A}}[u] \subseteq s^{\mathcal{A}}[v] \ ,$$
$$u \trianglelefteq^{\mathcal{A}} v \stackrel{\triangle}{\Leftrightarrow} c^{\mathcal{A}}[u] \subseteq c^{\mathcal{A}}[v] \ ,$$
$$u \preccurlyeq^{\mathcal{A}} v \stackrel{\triangle}{\Leftrightarrow} u \trianglelefteq^{\mathcal{A}} v \wedge f^{\mathcal{A}}[u] \subseteq f^{\mathcal{A}}[v] \ .$$

▶ **Example 3.1.** Consider the BA $\mathcal{D}$ in Fig. 1 (b). Since for all $u \in \Sigma^*$, $s^{\mathcal{D}}[ua] = \{q\}$ and $s^{\mathcal{D}}[ub] = s^{\mathcal{D}}[\epsilon] = \{q_0\}$, we have that $u \leq^{\mathcal{D}} v$ iff either $u, v \in \Sigma^* a$ or $u, v \in \Sigma^* b \cup \{\epsilon\}$. Similarly, we find that $u \trianglelefteq^{\mathcal{D}} v$ iff either $u, v \notin \{\epsilon\}$ and $u \leq^{\mathcal{D}} v$, or $u, v \in \{\epsilon\}$. For $u \in \Sigma^*$ we have $f^{\mathcal{D}}[ua] = \{(q_0, q), (q, q)\}$. For $u \in \Sigma^* \setminus b^*$ we have $f^{\mathcal{D}}[ub] = \{(q_0, q_0), (q, q_0)\}$ and $f^{\mathcal{D}}[b^k] = \{(q, q_0)\}$, for any $k \geq 1$. As for the empty word, $f^{\mathcal{D}}[\epsilon] = \{(q, q)\}$. Hence, for all $u, v \in \Sigma^*$, it turns out that $u \preccurlyeq^{\mathcal{C}} v$ holds iff one of the following four cases holds: (i) $u, v \in \Sigma^* a$; (ii) $u \in \Sigma^* b$ and $v \in \Sigma^* b \setminus b^*$; (iii) $u, v \in b^+$; (iv) $u, v \in \{\epsilon\}$. ⌟

The qos $\trianglelefteq^{\mathcal{A}}$ and $\leq^{\mathcal{A}}$ appeared in [15] while $\preccurlyeq^{\mathcal{A}}$ was obtained by relaxing an equivalence defined in [20]. By definition, we have that $\preccurlyeq^{\mathcal{A}} \subseteq \trianglelefteq^{\mathcal{A}}$, and since $q \in s^{\mathcal{A}}[u]$ iff $(i, q) \in c^{\mathcal{A}}[u]$, we deduce that $\preccurlyeq^{\mathcal{A}} \subseteq \trianglelefteq^{\mathcal{A}} \subseteq \leq^{\mathcal{A}}$ holds. Since Q is a finite set, it turns out that all these three state-based qos are indeed wqos. It is easily seen that $\trianglelefteq^{\mathcal{A}}$ and $\preccurlyeq^{\mathcal{A}}$ are monotonic and that $\leq^{\mathcal{A}}$ is right-monotonic. Finally, we turn to the preservation property with respect to $I_{L^\omega(\mathcal{A})}$. Let $(u, v) \in I_{L^\omega(\mathcal{A})}$. Then, $uv^\omega \in L^\omega(\mathcal{A})$ and there is an initial fair trace of $\mathcal{A}$ on uv^ω. Hence, we can find two states $p, q \in Q$ and two integers $n, m \geq 1$ such that $i \xrightarrow{u}{}^{*} p$, $p \xrightarrow{v^n}{}^{*} q$ and $q \xrightarrow{v^m}{}^{*}_{F} q$. Let $(s, t) \in \Sigma^* \times \Sigma^+$ be such that $u \leq^{\mathcal{A}} s$ and $v \preccurlyeq^{\mathcal{A}} t$. By monotonicity of $\preccurlyeq^{\mathcal{A}}$, we deduce that $v^k \preccurlyeq^{\mathcal{A}} t^k$ holds for all $k \in \mathbb{N}$. Hence, by definition of the state-based qos, we also have $i \xrightarrow{s}{}^{*} p$, $p \xrightarrow{t^n}{}^{*} q$ and $q \xrightarrow{t^m}{}^{*}_{F} q$. Therefore, $(s, t) \in I_{L^\omega(\mathcal{A})}$ holds. The argument remains the same if $u \trianglelefteq^{\mathcal{A}} s$ or $u \preccurlyeq^{\mathcal{A}} s$, so that we conclude that the pair $\leq^{\mathcal{A}}, \preccurlyeq^{\mathcal{A}}$, as well as the pairs $\trianglelefteq^{\mathcal{A}}, \preccurlyeq^{\mathcal{A}}$ and $\preccurlyeq^{\mathcal{A}}, \preccurlyeq^{\mathcal{A}}$ are pairs of wqos preserving $I_{L^\omega(\mathcal{A})}$.

4 An Algorithmic Framework for Checking Inclusion

We start with the ω-regular $\subseteq$ ω-regular case and then leverage the generality of our algorithmic framework to tackle the ω-context-free $\subseteq$ ω-regular case in Section 4.2.

4.1 Language Inclusion ω-regular $\subseteq$ ω-regular

Let us first recall the following fundamental theorem for languages of ω-words.

▶ **Theorem 4.1** ([5]). *The equivalence $L \subseteq M \iff I_L \subseteq I_M$ holds for all ω-regular languages $L, M \subseteq \Sigma^\omega$.*

Fix an ω-regular language M, a pair $\leq, \preccurlyeq$ of right-monotonic wqos on, resp., Σ^* and Σ^+, with $\rho_{\leq\times\preccurlyeq}(I_M) = I_M$, as given in Section 3, and a BA $\mathcal{A} = (Q, \delta, i_{\mathcal{A}}, F)$ such that $L = L^\omega(\mathcal{A})$.

A Representation for the Ultimately Periodic Words of L. We slightly generalize the approach presented in Section 2 and represent the ultimately periodic words of L by a subset $S \subseteq I_L$ such that $\{uv^\omega \mid (u,v) \in S\} = \{uv^\omega \mid (u,v) \in I_L\}$ holds, so that $L \subseteq M \Leftrightarrow S \subseteq I_M$ holds. The definition of such a subset S representing I_L relies on the following result.

▶ **Lemma 4.2.** *Let $\mathcal{A} = (Q, \delta, i_{\mathcal{A}}, F)$ be a BA. Then, $uv^\omega \in L^\omega(\mathcal{A})$ iff there exist $p \in F$, $u' \in \Sigma^*$, $v' \in \Sigma^+$ such that $uv^\omega = u'v'^\omega$, $i_{\mathcal{A}} \xrightarrow{u'}{}^* p$ and $p \xrightarrow{v'}{}^* p$.*

For each pair of states $q, q' \in Q$, we define the automaton $\mathcal{A}^q_{q'} \triangleq (Q, \delta, q, \{q'\})$. By Lemma 4.2, it turns out that the ultimately periodic words generated by the pairs of finite words in

$$S_{\mathcal{A}} \triangleq \bigcup_{p\in F} L^*(\mathcal{A}^{i_{\mathcal{A}}}_p) \times (L^*(\mathcal{A}^p_p)\setminus\{\epsilon\})$$

coincide with the ultimately periodic words of $L^\omega(\mathcal{A})$. Hence, by reasoning as in Section 2, it turns out that:

$$L^\omega(\mathcal{A}) \subseteq M \iff S_{\mathcal{A}} \subseteq I_M \iff \rho_{\leq\times\preccurlyeq}(S_{\mathcal{A}}) \subseteq I_M \ . \tag{1'}$$

Fixpoint Characterization of $S_{\mathcal{A}}$. For a function $f : X \to X$ over a set X and for all $n \in \mathbb{N}$, the n-th power $f^n : X \to X$ of f is inductively defined as usual: $f^0 \triangleq \lambda x.x$; $f^{n+1} \triangleq f \circ f^n$. The denumerable sequence of *Kleene iterates* of f starting from an initial value $a \in X$ is given by $\{f^n(a)\}_{n\in\mathbb{N}}$. This sequence finitely converges to some $f^k(a)$, with $k \in \mathbb{N}$, when for all $n \geq k$, $f^n(a) = f^k(a)$. Let us recall that when X is a directed-complete partial order with bottom $\perp$ and f is monotone, if the Kleene iterates starting from the bottom $\{f^n(\perp)\}_{n\in\mathbb{N}}$ finitely converge to some $f^k(\perp)$ then $f^k(\perp)$ is the least fixpoint of f, denoted by $\operatorname{lfp} f$.

Given $X \in \wp(\Sigma^*)^Q$, we define

$$\mathrm{Post}_{\mathcal{A}}(X) \triangleq \langle \textstyle\bigcup_{a\in\Sigma, q\in\delta(q',a)} X_{q'}a\rangle_{q\in Q} \in \wp(\Sigma^*)^Q \ ,$$

where, for all $q \in Q$, X_q denotes the q-indexed component of the vector X. In turn, for each $p \in F$, we define the maps

$$P_{\mathcal{A}} \triangleq \lambda X.\langle\{\epsilon \mid q = i_{\mathcal{A}}\} \cup\ (\mathrm{Post}_{\mathcal{A}}(X))_q\rangle_{q\in Q} \ ,$$
$$R_{\mathcal{A},p} \triangleq \lambda X.\langle\{a \in \Sigma \mid q \in \delta(p,a)\} \cup\ (\mathrm{Post}_{\mathcal{A}}(X))_q\rangle_{q\in Q} \ ,$$

which allows us to give the following least fixpoint characterization of $S_{\mathcal{A}}$.

▶ **Lemma 4.3.** $S_{\mathcal{A}} = \bigcup_{p\in F}(\operatorname{lfp} P_{\mathcal{A}})_p \times (\operatorname{lfp} R_{\mathcal{A},p})_p$.

▶ **Example 4.4.** Consider the BA $\mathcal{C}$ in Fig. 1. Since $L^*(\mathcal{C}^i_i) = \{a,b\}^*$, we have that $S_{\mathcal{C}} = \{a,b\}^* \times \{a,b\}^+$. Since $\mathcal{C}$ has only one state, vectors have dimension one. We have that $P_{\mathcal{C}} = \lambda X.\{\epsilon\} \cup Xa \cup Xb$ and $R_{\mathcal{C}} = \lambda X.\{a,b\} \cup Xa \cup Xb$, so that their Kleene iterates are $P^n_{\mathcal{C}}(\emptyset) = \{u \in \{a,b\}^* \mid |u| \leq n-1\}$ and $R^n_{\mathcal{C}}(\emptyset) = \{v \in \{a,b\}^+ \mid |v| \leq n\}$, for $n \in \mathbb{N}$. ⌟

A Finite Representation of $S_{\mathcal{A}}$. Given two vectors $X, X' \in \wp(\Sigma^*)^k$, we abuse notations and write $X \cup X'$ for the vector $\langle X_j \cup X'_j \rangle_{j\in[1,k]}$, and we write $X \subseteq X'$ when $X_j \subseteq X'_j$ for all $j \in [1, k]$. Given two functions $f : \wp(\Sigma^*)^k \to \wp(\Sigma^*)^k$ and $\rho : \wp(\Sigma^*) \to \wp(\Sigma^*)$ we write $f^n(\emptyset)$ for $f^n(\emptyset, \cdots, \emptyset)$, and $\rho \circ f$ for the function $\lambda X.\langle \rho((f(X))_j)\rangle_{j\in[1,k]}$.

Since $\leq$ is a wqo, $\rho_{\leq}(\operatorname{lfp} P_{\mathcal{A}}) = \rho_{\leq}(D)$ for some finite subset $D \subseteq_{\text{fin}} \operatorname{lfp} P_{\mathcal{A}}$. Since $\operatorname{lfp} P_{\mathcal{A}} = \bigcup_{n\in\mathbb{N}} P^n_{\mathcal{A}}(\emptyset)$, there exists some index $N_1 \in \mathbb{N}$ such that $D \subseteq P^{N_1}_{\mathcal{A}}(\emptyset)$. Hence, $\rho_{\leq}(P^{N_1}_{\mathcal{A}}(\emptyset)) = \rho_{\leq}(\operatorname{lfp} P_{\mathcal{A}})$ holds. This also applies to $\preccurlyeq$ and $R_{\mathcal{A},p}$, for each $p \in F$, so that there exists an index $N_2 \in \mathbb{N}$ such that $\rho_{\preccurlyeq}(R^{N_2}_{\mathcal{A},p}(\emptyset)) = \rho_{\preccurlyeq}(\operatorname{lfp} R_{\mathcal{A},p})$. Thus, by taking $T_p \triangleq P^{N_1}_{\mathcal{A}}(\emptyset) \times R^{N_2}_{\mathcal{A},p}(\emptyset)$, for each $p \in F$, we obtain a finite representation of $S_{\mathcal{A}}$, as required by step (2). By plugging the least fixpoint characterisation of $S_{\mathcal{A}}$ of Lemma 4.3 inside (1′), by observing that the closures preserve unions, and that $\rho_{\leq\times\preccurlyeq}$ and $\rho_{\leq} \times \rho_{\preccurlyeq}$ coincide on Cartesian products, we derive the following equivalences as in Section 2:

$$\begin{aligned} L^\omega(\mathcal{A}) \subseteq M &\iff \forall p \in F,\ \rho_{\leq\times\preccurlyeq}((\operatorname{lfp} P_{\mathcal{A}})_p \times (\operatorname{lfp} R_{\mathcal{A},p})_p) \subseteq I_M \\ &\iff \forall p \in F,\ \rho_{\leq\times\preccurlyeq}(T_p) \subseteq I_M \iff \forall p \in F,\ T_p \subseteq I_M \ . \end{aligned} \tag{2′}$$

▶ Remark 4.5. Assume that $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are two sequences of vectors in $\wp(\Sigma^*)^Q$ such that for each $n \in \mathbb{N}$: (i) the Q-indexed components of X_n and Y_n, for all n, are finite sets; (ii) for each $n \in \mathbb{N}$, $\rho_{\leq}(X_n) = \rho_{\leq}(P^n_{\mathcal{A}}(\emptyset))$ and $\rho_{\preccurlyeq}(Y_n) = \rho_{\preccurlyeq}(R^n_{\mathcal{A},p}(\emptyset))$. We call such a sequence $\{X_n\}_{n\in\mathbb{N}}$ (resp. $\{Y_n\}_{n\in\mathbb{N}}$) a sequence of *correct pruning steps* w.r.t. $\leq$ (resp. $\preccurlyeq$). Then, the vector $X_{N_1} \times Y_{N_2}$ can be used to achieve (2′), likewise T_p was used above. ⌟

Convergence Check. Let us now turn to the definition of a procedure for deciding when to stop the computations of $\rho_{\leq}(P^n_{\mathcal{A}}(\emptyset))$ and $\rho_{\preccurlyeq}(R^n_{\mathcal{A},p}(\emptyset))$. Here, we exploit a completeness property of the closures $\rho_{\leq}$ and $\rho_{\preccurlyeq}$, commonly used in abstract interpretation [8, 9]: a closure $\rho : C \to C$ is called *complete* for a function $f : C \to C$ when $\rho \circ f = \rho \circ f \circ \rho$ holds. Completeness is often used in abstract interpretation because it transfers to fixpoints, meaning that if ρ is complete for f then $\rho(\operatorname{lfp} f) = \operatorname{lfp}(\rho \circ f)$ holds [9, Theorem 7.1.0.4]. The following result provides a sufficient condition on a qo on Σ^* so as the induced closure operator turns out to be complete for the functions $P_{\mathcal{A}}$ and $R_{\mathcal{A},p}$, for each $p \in F$.

▶ **Lemma 4.6.** *Let $\mathcal{A} = (Q, \delta, i_{\mathcal{A}}, F)$ be a BA on Σ and $\leq$ be a right-monotonic qo on Σ^*. Then, $\rho_{\leq}$ is complete for $P_{\mathcal{A}}$ and $R_{\mathcal{A},p}$, for each $p \in F$.*

We are now in position to show that if the qos $\leq$ and $\preccurlyeq$ are right-monotonic and decidable, then a finite representation of $S_{\mathcal{A}}$ can be computed. First, observe that for all $n \geq 0$, $P^n_{\mathcal{A}}(\emptyset)$ is finite and computable (an easy induction can prove this). Let us also notice that $P_{\mathcal{A}}$ is a monotone function, hence $\rho_{\leq} \circ P_{\mathcal{A}}$ is monotone as well. Suppose that $\rho_{\leq}(P^{N_1+1}_{\mathcal{A}}(\emptyset)) \subseteq \rho_{\leq}(P^{N_1}_{\mathcal{A}}(\emptyset))$ holds for some $N_1 \in \mathbb{N}$. Thus, by monotonicity of $\rho_{\leq} \circ P_{\mathcal{A}}$, it turns out that $\rho_{\leq} \circ P_{\mathcal{A}} \circ \rho_{\leq}(P^{N_1+1}_{\mathcal{A}}(\emptyset)) \subseteq \rho_{\leq} \circ P_{\mathcal{A}} \circ \rho_{\leq}(P^{N_1}_{\mathcal{A}}(\emptyset))$. By Lemma 4.6, $\rho_{\leq}$ is complete for $P_{\mathcal{A}}$, hence this latter inclusion is equivalent to $\rho_{\leq}(P^{N_1+2}_{\mathcal{A}}(\emptyset)) \subseteq \rho_{\leq}(P^{N_1+1}_{\mathcal{A}}(\emptyset))$. A simple induction based on this argument proves that for all $k \geq N_1$, $\rho_{\leq}(P^k_{\mathcal{A}}(\emptyset)) \subseteq \rho_{\leq}(P^{N_1}_{\mathcal{A}}(\emptyset))$ holds, so that we obtain that $\{\rho_{\leq}(P^n_{\mathcal{A}}(\emptyset))\}_{n\in\mathbb{N}}$ finitely converges at iteration N_1. Hence, to detect convergence of the iterates we check whether $\rho_{\leq}(P^{n+1}_{\mathcal{A}}(\emptyset)) \subseteq \rho_{\leq}(P^n_{\mathcal{A}}(\emptyset))$ holds or not. When the qo $\leq$ is decidable, this test boils down to check if for each $x \in P^{n+1}_{\mathcal{A}}(\emptyset)$, there exists $y \in P^n_{\mathcal{A}}(\emptyset)$ such that $y \leq x$. This same reasoning also applies to $\preccurlyeq$ and $R_{\mathcal{A},p}$.

Word-based Inclusion Algorithms. Our "word-based" algorithm BAIncW for checking $L^\omega(\mathcal{A}) \subseteq M$ is parameterized by a pair of right-monotonic wqos $\leq, \preccurlyeq$ (on, resp., Σ^*, Σ^+) preserving I_M. It computes the Kleene iterates $P^n_{\mathcal{A}}(\emptyset)$ and $R^n_{\mathcal{A},p}(\emptyset)$, for each final state

$p \in F$, until $\rho_{\leq}((P_{\mathcal{A}}^{N_1+1}(\emptyset))_q) \subseteq \rho_{\leq}((P_{\mathcal{A}}^{N_1}(\emptyset))_q)$ and $\rho_{\preccurlyeq}((R_{\mathcal{A},p}^{N_2+1}(\emptyset))_q) \subseteq \rho_{\preccurlyeq}((R_{\mathcal{A},p}^{N_2}(\emptyset))_q)$ hold for each $q \in Q$ and some $N_1, N_2 \in \mathbb{N}$. The resulting finite sets of words $(P_{\mathcal{A}}^{N_1}(\emptyset))_p$ and $(R_{\mathcal{A},p}^{N_2}(\emptyset))_p$, for each final state $p \in F$, are used by the membership check procedure enabled by (2′):

$$L^\omega(\mathcal{A}) \subseteq M \iff \forall p \in F, \forall u \in (P_{\mathcal{A}}^{N_1}(\emptyset))_p, \forall v \in (R_{\mathcal{A},p}^{N_2}(\emptyset))_p,\ uv^\omega \in M.$$

BAIncW Word-based algorithm for checking $L^\omega(\mathcal{A}) \subseteq M$.

Data: Büchi automaton $\mathcal{A} = (Q, \delta, i_{\mathcal{A}}, F)$
Data: Procedure deciding $uv^\omega \overset{?}{\in} M$ given $(u, v) \in \Sigma^* \times \Sigma^+$
Data: Decidable right-monotonic wqos $\leq, \preccurlyeq$ s.t. $\rho_{\leq\times\preccurlyeq}(I_M) = I_M$

1 Compute $P_{\mathcal{A}}^{N_1}(\emptyset)$ with least N_1 s.t. $\forall q \in Q$, $\rho_{\leq}((P_{\mathcal{A}}^{N_1+1}(\emptyset))_q) \subseteq \rho_{\leq}((P_{\mathcal{A}}^{N_1}(\emptyset))_q)$;
2 **foreach** $p \in F$ **do**
3   Compute $R_{\mathcal{A},p}^{N_2}(\emptyset)$ with least N_2 s.t. $\forall q \in Q$, $\rho_{\preccurlyeq}((R_{\mathcal{A},p}^{N_2+1}(\emptyset))_q) \subseteq \rho_{\preccurlyeq}((R_{\mathcal{A},p}^{N_2}(\emptyset))_q)$;
4   **foreach** $u \in (P_{\mathcal{A}}^{N_1}(\emptyset))_p$, $v \in (R_{\mathcal{A},p}^{N_2}(\emptyset))_p$ **do**
5   **if** $uv^\omega \notin M$ **then return false**;
6 **return true**;

▶ **Theorem 4.7.** *Given all the required input data,* BAIncW *decides* $L^\omega(\mathcal{A}) \subseteq M$.

▶ Remark 4.8. The for-loop at lines 2-5 of BAIncW is restricted to the final states $p \in F$ of the BA $\mathcal{A}$. Thus, in general, the less they are the better is for BAIncW. ⌟

▶ **Example 4.9.** Consider the BAs $\mathcal{C}$ and $\mathcal{D}$ in Fig. 1. From Example 4.4 we have that $P_{\mathcal{C}}(\emptyset) = \{\epsilon\}$, $P_{\mathcal{C}}^2(\emptyset) = \{\epsilon, a, b\}$ and $P_{\mathcal{C}}^3(\emptyset) = \{\epsilon, a, b, aa, ab, ba, bb\}$. From Example 3.1, for $u \in \{aa, ba\}$ and $v \in \{ab, bb\}$, we have that $a \leq^{\mathcal{D}} u$ and $b \leq^{\mathcal{D}} v$, while a and ϵ are incomparable for $\leq^{\mathcal{D}}$. Hence, $\rho_{\leq^{\mathcal{D}}}(P_{\mathcal{C}}(\emptyset)) \neq \rho_{\leq^{\mathcal{D}}}(P_{\mathcal{C}}^2(\emptyset))$ and $\rho_{\leq^{\mathcal{D}}}(P_{\mathcal{C}}^2(\emptyset)) = \rho_{\leq^{\mathcal{D}}}(P_{\mathcal{C}}^3(\emptyset))$ hold, so that a finite representation of lfp $P_{\mathcal{C}}$ is achieved by $P_{\mathcal{C}}^2(\emptyset)$. Since $\rho_{\preccurlyeq^{\mathcal{D}}}(R_{\mathcal{C}}^2(\emptyset)) = \rho_{\preccurlyeq^{\mathcal{D}}}(R_{\mathcal{C}}^1(\emptyset))$, the membership check is performed on the elements of $P_{\mathcal{C}}^2(\emptyset) \times R_{\mathcal{C}}^1(\emptyset) = \{\epsilon, a, b\} \times \{a, b\}$, and for $(a, b) \in P_{\mathcal{C}}^2(\emptyset) \times R_{\mathcal{C}}^1(\emptyset)$, the word ab^ω is a witness that $L^\omega(\mathcal{C}) \nsubseteq L^\omega(\mathcal{D})$. ⌟

As explained by Remark 4.5, any sequence of correct pruning steps for the Kleene iterates can be safely exploited to compute a finite representation of $S_{\mathcal{A}}$. This is formalized by the algorithm BAIncW given in App. A.

The pairs of qos derived from M as defined in Section 3, are all pairs of decidable right-monotonic wqos that verify the preservation property w.r.t. M. Each of them yields a slightly different algorithm deciding whether $L^\omega(\mathcal{A}) \subseteq M$ holds (see the discussion in Section 4.3).

4.2 Language Inclusion ω-context-free $\subseteq$ ω-regular

A (*Büchi*) *pushdown automaton* ((B)PDA) on Σ is a tuple $\mathcal{P} = (Q, \Gamma, \delta, i, F)$ where Q is a finite set of states including an initial state i, Γ is the stack alphabet including an initial stack symbol $\bot$, $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \times Q \times \Gamma^*$ is the finite set of transitions, and $F \subseteq Q$ is a subset of accepting states. Configurations of the PDA $\mathcal{P}$ are pairs in $Q \times \Gamma^*$ and, for each $a \in \Sigma$, the transition relation $\vdash^a$ between configurations is defined by $(q, \gamma w) \vdash^a (p, \beta w)$, for some $w \in \Gamma^*$, when $(q, a, \gamma, p, \beta) \in \delta$, and it is lifted to words by reflexivity and transitivity, that is, for all $u \in \Sigma^*$, $(q, w) \vdash^{*u} (p, w')$ when the configurations (q, w) and (p, w') are related by a sequence of transitions such that the concatenation of the corresponding labels is the word u. We write $(q, w) \vdash^{*u}_F (p, w')$ when such a sequence includes a configuration whose state is final. The language of finite words accepted by a

PDA $\mathcal{P}$ is $L^*(\mathcal{P}) \triangleq \{u \in \Sigma^* \mid (i, \bot) \vdash^{*u} (p, w),\ p \in F,\ w \in \Gamma^*\}$. A natural extension from finite to infinite words relies on infinite sequences of configurations as follows. A trace of $\mathcal{P}$ for an ω-word $\xi = a_0 a_1 \cdots \in \Sigma^\omega$ is an infinite sequence $(q_0, w_0) \vdash^{*a_0} (q_1, w_1) \vdash^{*a_1} \cdots$, which is initial when $(q_0, w_0) = (i, \bot)$ and fair when $q_j \in F$ for infinitely many j's. The ω-language accepted by $\mathcal{P}$ is $L^\omega(\mathcal{P}) \triangleq \{\xi \in \Sigma^\omega \mid$ there exists an initial and fair trace of $\mathcal{P}$ for $\xi\}$. An ω-language $L \subseteq \Sigma^\omega$ is ω-context-free if $L = L^\omega(\mathcal{P})$ for some BPDA $\mathcal{P}$ on Σ.

We fix an ω-regular language M, a pair $\leq, \preccurlyeq$ of monotonic wqos on Σ^*, Σ^+ such that $\rho_{\leq \times \preccurlyeq}(I_M) = I_M$ holds, and a BPDA $\mathcal{P}$ such that $L = L^\omega(\mathcal{P})$. Theorem 4.1 still holds when the "left" language L is ω-context-free, so that $L \subseteq M \iff I_L \subseteq I_M$ holds. The following result generalises Lemma 4.2 to BPDAs.

▶ **Lemma 4.10.** *Let $\mathcal{P} = (Q, \Gamma, \delta, i, F)$ be a BPDA. Then, $uv^\omega \in L^\omega(\mathcal{P})$ iff there exist $(q, \gamma) \in Q \times \Gamma$, $u' \in \Sigma^*$, $v' \in \Sigma^+$ such that $uv^\omega = u'v'^\omega$, $(i, \bot) \vdash^{*u'} (q, \gamma s)$ and $(q, \gamma) \vdash^{*v'}_F (q, \gamma w)$, for some $s, w \in \Gamma^*$.*

Similarly to the ω-regular case described in Section 4.1, Lemma 4.10 allows us to define two PDAs $\mathcal{P}^1_{q\gamma}$ and $\mathcal{P}^2_{q\gamma}$, where for each $(q, \gamma) \in Q \times \Gamma$, $\mathcal{P}^1_{q\gamma}$ deals with the prefixes, $\mathcal{P}^2_{q\gamma}$ deals with the periods, and are such that the ultimately periodic words generated by the pairs in $S_{\mathcal{P}} \triangleq \bigcup_{(q,\gamma) \in Q \times \Gamma} L^*(\mathcal{P}^1_{q\gamma}) \times L^*(\mathcal{P}^2_{q\gamma})$ coincide with those of $L^\omega(\mathcal{P})$. Hence, similarly to (1′) for the ω-regular case, it turns out that:

$$L^\omega(\mathcal{P}) \subseteq M \iff S_{\mathcal{P}} \subseteq I_M \iff \rho_{\leq \times \preccurlyeq}(S_{\mathcal{P}}) \subseteq I_M \ . \tag{1''}$$

Moreover, analogously to Lemma 4.3 for the ω-regular case, $S_{\mathcal{P}}$ admits a least fixpoint characterisation.

▶ **Lemma 4.11.** *Any PDA $\mathcal{P}$ induces a monotone map $F_{\mathcal{P}} : \wp(\Sigma^*)^m \to \wp(\Sigma^*)^m$, for some $m \in \mathbb{N}$, such that $L^*(\mathcal{P}) = (\operatorname{lfp} F_{\mathcal{P}})_0$.*

Let us mention that the definition of $F_{\mathcal{P}}$ relies on the production rules of a context-free grammar (CFG) accepting $L^*(\mathcal{P})$ and that $(\operatorname{lfp} F_{\mathcal{P}})_0$ denotes the first vector component corresponding to the start variable of the CFG. Let $P_{q\gamma}$ and $R_{q\gamma}$ be the functions provided by Lemma 4.11 for the two PDAs $\mathcal{P}^1_{q\gamma}$ and $\mathcal{P}^2_{q\gamma}$ defined above for each $(q, \gamma) \in Q \times \Gamma$. By (1″) and Lemma 4.11, it turns out that:

$$L^\omega(\mathcal{P}) \subseteq M \iff \forall (q, \gamma) \in Q \times \Gamma,\ \rho_{\leq}((\operatorname{lfp} P_{q\gamma})_0) \times \rho_{\preccurlyeq}((\operatorname{lfp} R_{q\gamma})_0)) \subseteq I_M \ . \tag{2''}$$

Since both $\leq$ and $\preccurlyeq$ are wqos, the corresponding upward-closed sets in (2″) can be obtained as upward closure of some finite subsets. In particular, by reasoning as for the ω-regular case, we have that for each $(q, \gamma) \in Q \times \Gamma$ there exist $N_1, N_2 \in \mathbb{N}$ such that $\rho_{\leq}((\operatorname{lfp} P_{q\gamma})_0) = \rho_{\leq}((P^{N_1}_{q\gamma}(\emptyset))_0)$ and $\rho_{\preccurlyeq}((\operatorname{lfp} R_{q\gamma})_0) = \rho_{\preccurlyeq}((R^{N_2}_{q\gamma}(\emptyset))_0)$ hold.

Let us now turn to the convergence of the sequences of Kleene iterates. Being induced by the rules of a CFG, the function $F_{\mathcal{P}}(\langle X_1, ..., X_m \rangle)$ of Lemma 4.11 may rely on nonlinear concatenations of type $X_i X_j$ for some $i, j \in [1, m]$, so that prefixes and periods in $S_{\mathcal{P}}$ can be obtained both by left- and right-concatenations. This is different from the ω-regular case, where only right-concatenations were needed. Thus, in contrast to the ω-regular case of Lemma 4.6, we need stronger monotonicity conditions on the qos $\leq$ and $\preccurlyeq$ in order to ensure the completeness of the closures $\rho_{\leq}$ and $\rho_{\preccurlyeq}$ for, resp., $P_{q\gamma}$ and $R_{q\gamma}$: both qos need to be (left- and right-) monotonic.

▶ **Lemma 4.12.** *Let $\mathcal{P}$ be a BPDA on Σ and $\leq$ be a monotonic qo on Σ^*. Then, $\rho_{\leq}$ is complete for all the functions $P_{q\gamma}$ and $R_{q\gamma}$ induced by $\mathcal{P}$.*

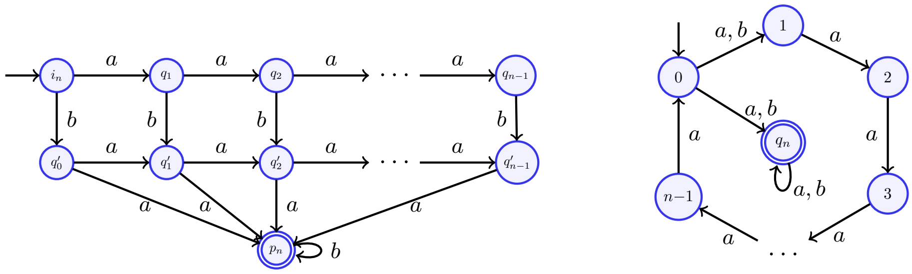


Figure 2 The families $\{\mathcal{A}_n\}_{n\geq 2}$ (left) and $\{\mathcal{B}_n\}_{n\geq 2}$ (right) s.t. $L^\omega(\mathcal{A}_n) \subseteq L^\omega(\mathcal{B}_n)$ for all n.

Hence, by Lemma 4.12, the same arguments used for the ω-regular case entail that the convergence of the Kleene iterates of $P_{q\gamma}$ and $R_{q\gamma}$ boils down to check, resp., the conditions: $\rho_\leq(P^{n+1}_{q\gamma}(\emptyset)) \subseteq \rho_\leq(P^n_{q\gamma}(\emptyset))$ and $\rho_\preccurlyeq(R^{n+1}_{q\gamma}(\emptyset)) \subseteq \rho_\preccurlyeq(R^n_{q\gamma}(\emptyset))$, for some $n \in \mathbb{N}$.

Summing up, our "word-based" algorithm for $L^\omega(\mathcal{P}) \subseteq M$ follows the same template of BAIncW for the ω-regular case. First, it computes the iterates of $P_{q\gamma}$ and $R_{q\gamma}$ for, resp., the prefix and period languages, until finite convergence is reached. Then, the resulting finite sets of words $(P^{N_1}_{q\gamma}(\emptyset))_0$ and $(R^{N_2}_{q\gamma}(\emptyset))_0$ are used by the following membership check:

$$L^\omega(\mathcal{P}) \subseteq M \iff \forall(q,\gamma) \in Q \times \Gamma, \forall u \in (P^{N_1}_{q\gamma}(\emptyset))_0, \forall v \in (R^{N_1}_{q\gamma}(\emptyset))_0,\ uv^\omega \in M \ .$$

The pairs of state-based wqos that can be used to decide the inclusion $L^\omega(\mathcal{P}) \subseteq M$ are $\trianglelefteq^{\mathcal{B}}, \preccurlyeq^{\mathcal{B}}$ and $\preccurlyeq^{\mathcal{B}}, \preccurlyeq^{\mathcal{B}}$, where $\mathcal{B}$ is a BA recognising M, as defined in Section 3.

4.3 Discussion

Let us discuss how the inclusion algorithms provided by pairs of qos defined in Section 3 can be related to each other. Consider two wqos $\leq, \leq' \subseteq \Sigma^* \times \Sigma^+$ such that $\leq$ is coarser than $\leq'$, i.e., $\leq' \subseteq \leq$ holds. It turns out that $\rho_{\leq'}(X) \subseteq \rho_{\leq'}(Y)$ implies $\rho_\leq(X) \subseteq \rho_\leq(Y)$, so that if some Kleene iterates of BAIncW converge in N' steps w.r.t. $\leq'$, then the same Kleene iterates converge in $N \leq N'$ steps w.r.t. $\leq$, namely, convergence can be "faster" with a coarser qo. Also, given a wqo $\leq$ and a nonempty set $X \in \wp(\Sigma^*)$, consider the set $\mathcal{C}_X \triangleq \{Y \subseteq_{\text{fin}} X \mid \rho_\leq(Y) = \rho_\leq(X)\}$ of finite subsets of X inducing the same $\leq$-upward closure as X, which is not empty because $\leq$ is a wqo. An element of $\mathcal{C}_X$ of minimal size is called a *minor* of X and denoted by $\lfloor X \rfloor_\leq$. If $\leq$ is coarser than $\leq'$ then any minor $\lfloor X \rfloor_\leq$ w.r.t. $\leq$ has at most as many elements as any minor $\lfloor X \rfloor_{\leq'}$ w.r.t. $\leq'$. Thus, a coarser pair of wqos may achieve a smaller minimal representation on which to perform the membership queries of BAIncW. The following example shows the benefits of using the coarsest state-based pair of wqos on the family of inclusion problems between the BAs depicted in Fig. 2.

▶ **Example 4.13.** Consider the families of BAs $\{\mathcal{A}_n\}_{n\geq 2}$ and $\{\mathcal{B}_n\}_{n\geq 2}$ in Fig. 2. Let $X_n \triangleq \{a^iba^{j+1} \in \Sigma^* \mid i,j \geq 0,\ i+j \leq n-1\}$ such that $L^*({\mathcal{A}_n}^{i_n}_{p_n}) = X_n\{b\}^*$ and $L^*({\mathcal{A}_n}^{p_n}_{p_n}) \setminus \{\epsilon\} = b^+$. For any $w \in L^*({\mathcal{A}_n}^{i_n}_{p_n})$ we have that $q_n \in s^{\mathcal{B}_n}[w]$, and, since $s^{\mathcal{B}_n}[aba] = \{q_n\}$, it holds that $aba \leq^{\mathcal{B}_n} w$. Since $aba \in L^*({\mathcal{A}_n}^{i_n}_{p_n})$, we deduce that any minor $\lfloor L^*({\mathcal{A}_n}^{i_n}_{p_n}) \rfloor_{\leq^{\mathcal{B}_n}}$ has size one. Similarly, any minor $\lfloor L^*({\mathcal{A}_n}^{p_n}_{p_n}) \setminus \{\epsilon\} \rfloor_{\preccurlyeq^{\mathcal{B}_n}}$ has size one. We also have that $c^{\mathcal{B}_n}[a^iba^{j+1}] = \{(n-i, j+2), (0, q_n), (q_n, q_n)\}$. Hence, if $w \preccurlyeq^{\mathcal{B}_n} w'$, for $w, w' \in X_n$, then $w = w'$. Since X_n has size $\frac{n(n+1)}{2}$, all the minors $\lfloor L^*({\mathcal{A}_n}^{i_n}_{p_n}) \setminus \{\epsilon\} \rfloor_{\preccurlyeq^{\mathcal{B}_n}}$ and $\lfloor L^*({\mathcal{A}_n}^{i_n}_{p_n}) \setminus \{\epsilon\} \rfloor_{\trianglelefteq^{\mathcal{B}_n}}$ have at least $\frac{n(n+1)}{2}$ elements. Hence, using the pair of qos $\leq^{\mathcal{B}_n}, \preccurlyeq^{\mathcal{B}_n}$, a single membership query (i.e., $uv^\omega \in L^\omega(\mathcal{B}_n)$) is needed to decide the inclusion $L^\omega(\mathcal{A}_n) \subseteq L^\omega(\mathcal{B}_n)$, as opposed to no less than $\frac{n(n+1)}{2}$ membership queries for the other pairs of qos. ⌟

▶ Remark 4.14. The supergraphs of [2, Def. 6] endowed with their subsumption orders coincide with our qo $\preccurlyeq$. Without the subsumption order they coincide with $\preccurlyeq \cap \preccurlyeq^{-1}$. ⌟

5 State-Based Inclusion Algorithms

In this section, we show how to derive *state-based* inclusion algorithms, namely, algorithms that, given two BAs $\mathcal{A} = (Q_\mathcal{A}, \delta_\mathcal{A}, i_\mathcal{A}, F_\mathcal{A})$ and $\mathcal{B} = (Q_\mathcal{B}, \delta_\mathcal{B}, i_\mathcal{B}, F_\mathcal{B})$, decide whether $L^\omega(\mathcal{A}) \subseteq L^\omega(\mathcal{B})$ by operating on the states of $\mathcal{A}$ and $\mathcal{B}$ only. The intuition is that words are abstracted into states and, correspondingly, operations/tests on words are abstracted into operations/tests on states. Of course, the key to enable such abstractions are the state-based qos defined in Section 3, whose definitions rely just on the states of a BA representing an ω-language. Due to lack of space, we focus on the ω-regular case, while a state-based algorithm for the context-free case is given in App. B and is designed by following an analogous pattern.

We focus on the pair of qos $\leq^\mathcal{B}, \preccurlyeq^\mathcal{B}$ defined in Section 3. The state-based algorithms for different pairs of qos can be analogously derived. Given an ultimately periodic word uv^ω, the prefix $u \in \Sigma^*$ is abstracted by the set of its successor states in $\mathcal{B}$ given by $s^\mathcal{B}[u] \in \wp(Q_\mathcal{B})$, while the period v is abstracted by the pair $(c^\mathcal{B}[v], f^\mathcal{B}[v]) \in \wp(Q_\mathcal{B}^2) \times \wp(Q_\mathcal{B}^2)$ providing its context and final context in $\mathcal{B}$. Thus, the state abstraction of $S_\mathcal{A}$, as given in Lemma 4.3, is:

$$S_{\mathcal{A},\mathcal{B}} \triangleq \bigcup_{p\in F_\mathcal{A}} \{s^\mathcal{B}[u] \mid u \in (\operatorname{lfp} P_\mathcal{A})_p\} \times \{(c^\mathcal{B}[v], f^\mathcal{B}[v]) \mid v \in (\operatorname{lfp} R_{\mathcal{A},p})_p\} \ .$$

We give a fixpoint characterisation of $S_{\mathcal{A},\mathcal{B}}$ using the state abstractions of the functions $P_\mathcal{A}$ and $R_{\mathcal{A},p}$ w.r.t., resp., the qos $\leq^\mathcal{B}$ and $\preccurlyeq^\mathcal{B}$.

Let us define the maps $\mathrm{Post}_\mathcal{A}^{\leq^\mathcal{B}} : \wp(\wp(Q_\mathcal{B}))^{Q_\mathcal{A}} \to \wp(\wp(Q_\mathcal{B}))^{Q_\mathcal{A}}$ and $\mathrm{Post}_\mathcal{A}^{\preccurlyeq^\mathcal{B}} : \wp(\wp(Q_\mathcal{B}^2) \times \wp(Q_\mathcal{B}^2))^{Q_\mathcal{A}} \to \wp(\wp(Q_\mathcal{B}^2) \times \wp(Q_\mathcal{B}^2))^{Q_\mathcal{A}}$ as follows:

$$\mathrm{Post}_\mathcal{A}^{\leq^\mathcal{B}}(X) \triangleq \langle \textstyle\bigcup_{a\in\Sigma, q\in\delta_\mathcal{A}(q',a)} \{y \star a \mid y \in X_{q'}\}\rangle_{q\in Q_\mathcal{A}}$$
$$\mathrm{Post}_\mathcal{A}^{\preccurlyeq^\mathcal{B}}(Y) \triangleq \langle \textstyle\bigcup_{a\in\Sigma, q\in\delta_\mathcal{A}(q',a)} \{\big(y_1 \circ c^\mathcal{B}[a], y_1 \circ f^\mathcal{B}[a] \cup y_2 \circ c^\mathcal{B}[a]\big) \mid (y_1, y_2) \in Y_{q'}\}\rangle_{q\in Q_\mathcal{A}}$$

where $y \star a \triangleq \bigcup_{q'\in y}\{q \in Q_\mathcal{B} \mid (q', q) \in c^\mathcal{B}[a]\}$, for $y \in \wp(Q_\mathcal{B})$ and $a \in \Sigma$. The intuition for this latter definition is the following: if $y = s^\mathcal{B}[u]$, for some $u \in \Sigma^*$, then $y \star a = s^\mathcal{B}[ua]$. Also, given two binary relations $y_1, y_2 \in \wp(Q_\mathcal{B}^2)$ on states of $\mathcal{B}$, the notation $y_1 \circ y_2$ denotes their composition. Here, the intuition is similar: if $y_1 = c^\mathcal{B}[u]$ and $y_2 = f^\mathcal{B}[u]$, for some $u \in \Sigma^*$, then $y_1 \circ c^\mathcal{B}[a] = c^\mathcal{B}[ua]$ and $y_1 \circ f^\mathcal{B}[a] \cup y_2 \circ c^\mathcal{B}[a] = f^\mathcal{B}[ua]$. In turn, the functions:

$$P_{\mathcal{A},\mathcal{B}} \triangleq \lambda X \in \wp(\wp(Q_\mathcal{B}))^{Q_\mathcal{A}}.\ \langle\{\{i_\mathcal{B}\} \mid q = i_\mathcal{A}\}\rangle_{q\in Q_\mathcal{A}} \cup \mathrm{Post}_\mathcal{A}^{\leq^\mathcal{B}}(X)$$
$$R_{\mathcal{A},\mathcal{B},p} \triangleq \lambda Y \in \wp(\wp(Q_\mathcal{B}^2) \times \wp(Q_\mathcal{B}^2))^{Q_\mathcal{A}}.\ \langle\{(c^\mathcal{B}[a], f^\mathcal{B}[a]) \mid q \in \delta_\mathcal{A}(p,a)\}\rangle_{q\in Q_\mathcal{A}} \cup \mathrm{Post}_\mathcal{A}^{\preccurlyeq^\mathcal{B}}(Y)$$

with $p \in Q_\mathcal{A}$, give us the following least fixpoint characterization:

▶ **Lemma 5.1.** $S_{\mathcal{A},\mathcal{B}} = \bigcup_{p\in F_\mathcal{A}} (\operatorname{lfp} P_{\mathcal{A},\mathcal{B}})_p \times (\operatorname{lfp} R_{\mathcal{A},\mathcal{B},p})_p$.

Let us now turn to the convergence check for the Kleene iterates of $P_{\mathcal{A},\mathcal{B}}$ and $R_{\mathcal{A},\mathcal{B},p}$. The qo $\leq^\mathcal{B}$ on words translates into the inclusion order $\subseteq$ on $\wp(Q_\mathcal{B})$ and, analogously, $\preccurlyeq^\mathcal{B}$ translates into the componentwise inclusion order $\subseteq^2 \triangleq \subseteq \times \subseteq$ on $\wp(Q_\mathcal{B}^2) \times \wp(Q_\mathcal{B}^2)$. Hence, the convergence of the iterates $P_{\mathcal{A},\mathcal{B}}^n(\emptyset)$ is checked by $\rho_\subseteq(P_{\mathcal{A},\mathcal{B}}^{n+1}(\emptyset)) \subseteq \rho_\subseteq(P_{\mathcal{A},\mathcal{B}}^n(\emptyset))$ (where $\subseteq$ is componentwise on vectors). Similarly, for the iterates $R_{\mathcal{A},\mathcal{B},p}^n(\emptyset)$ w.r.t. $\subseteq^2$. Let us remark that since the inclusion $\subseteq$ is a partial order (rather than a mere qo), each set $X \in \wp(\wp(Q_\mathcal{B}))$ admits a unique minor $\lfloor X \rfloor$ w.r.t. $\subseteq$, and similarly for $\subseteq^2$. Hence, the sequences of minors $\{\lfloor P_{\mathcal{A},\mathcal{B}}^n(\emptyset)\rfloor\}_{n\in\mathbb{N}}$ and $\{\lfloor R_{\mathcal{A},\mathcal{B},p}^n(\emptyset)\rfloor\}_{n\in\mathbb{N}}$ w.r.t., resp., $\subseteq$ and $\subseteq^2$, are uniquely defined. Since these are sequences of correct pruning steps according to Remark 4.5, they can be exploited to achieve a smaller representation of $S_{\mathcal{A},\mathcal{B}}$. Hence, the clear rationale to use these uniquely defined minors is to keep at each iteration the minimum number of elements of the Kleene iterates for representing them.

Finally, let us discuss the state abstraction of the membership check $uv^\omega \in L^\omega(\mathcal{B})$. For $x \in \wp(Q_\mathcal{B})$ and $(y_1, y_2) \in \wp(Q^2_\mathcal{B}) \times \wp(Q^2_\mathcal{B})$, define the following state-based inclusion predicate:

$$\mathrm{Inc}^\mathcal{B}(x, (y_1, y_2)) \triangleq \exists q, q' \in Q_\mathcal{B},\ q \in x \wedge (q, q') \in y_1^* \wedge (q', q') \in y_1^* \circ y_2 \circ y_1^* \ .$$

This is the correct state-based membership check because for all $u \in \Sigma^*$, $v \in \Sigma^+$, it turns out that $uv^\omega \in L^\omega(\mathcal{B}) \Leftrightarrow \mathrm{Inc}^\mathcal{B}(s^\mathcal{B}[u], (c^\mathcal{B}[v], f^\mathcal{B}[v]))$.

Summing up, we are now in a position to put forward our state-based algorithm BAIncS for checking $L^\omega(\mathcal{A}) \subseteq L^\omega(\mathcal{B})$. An illustrative run on the example of Fig. 1 is given in Section 5.1.

BAIncS State-based algorithm for checking $L^\omega(\mathcal{A}) \subseteq L^\omega(\mathcal{B})$.

Data: Büchi automata $\mathcal{A} = (Q_\mathcal{A}, \delta_\mathcal{A}, i_\mathcal{A}, F_\mathcal{A})$ and $\mathcal{B} = (Q_\mathcal{B}, \delta_\mathcal{B}, i_\mathcal{B}, F_\mathcal{B})$

Compute $\lfloor P^{N_1}_{\mathcal{A},\mathcal{B}}(\emptyset)\rfloor$ with least N_1 s.t. $\forall q \in Q_\mathcal{A}$, $\rho_\subseteq((P^{N_1+1}_{\mathcal{A},\mathcal{B}}(\emptyset))_q) \subseteq \rho_\subseteq((P^{N_1}_{\mathcal{A},\mathcal{B}}(\emptyset))_q)$;
foreach $p \in F_\mathcal{A}$ **do**
  Compute $\lfloor R^{N_2}_{\mathcal{A},\mathcal{B},p}(\emptyset)\rfloor$ with least N_2 s.t. $\forall q \in Q_\mathcal{A}$, $\rho_{\subseteq^2}((R^{N_2+1}_{\mathcal{A},\mathcal{B},p}(\emptyset))_q) \subseteq \rho_{\subseteq^2}((R^{N_2}_{\mathcal{A},\mathcal{B},p}(\emptyset))_q)$;
  foreach $x \in (\lfloor P^{N_1}_{\mathcal{A},\mathcal{B}}(\emptyset)\rfloor)_p$, $(y_1, y_2) \in (\lfloor R^{N_2}_{\mathcal{A},\mathcal{B},p}(\emptyset)\rfloor)_p$ **do**
    if $\neg\mathrm{Inc}^\mathcal{B}(x, (y_1, y_2))$ **then return false**;
return true;

▶ **Theorem 5.2.** *The algorithm* BAIncS *decides* $L^\omega(\mathcal{A}) \subseteq L^\omega(\mathcal{B})$.

5.1 Illustrative Example of BAIncS

We show the execution of a run of BAIncS on the BAs $\mathcal{C}$ and $\mathcal{D}$ depicted in Fig. 1. As a result, the algorithm will correctly decide that $L^\omega(\mathcal{C})$ is not included in $L^\omega(\mathcal{D})$ (e.g., $ab^\omega \in L^\omega(\mathcal{C})$ but $ab^\omega \notin L^\omega(\mathcal{D})$). Observe that since $\mathcal{C}$ consists of a single state, vectors are not needed.

First, the algorithm evaluates the sequence $\{\lfloor P^n_{\mathcal{C},\mathcal{D}}(\emptyset)\rfloor\}_{n\in\mathbb{N}} \in (\wp(\wp(Q_\mathcal{D})))^\mathbb{N}$, where $P_{\mathcal{C},\mathcal{D}}(X) = \{\{q_0\}\} \cup \{x \star a \mid x \in X\} \cup \{x \star b \mid x \in X\}$.

(1) $\lfloor P^1_{\mathcal{C},\mathcal{D}}(\emptyset)\rfloor = \{\{q_0\}\}$,

(2) $\lfloor P^2_{\mathcal{C},\mathcal{D}}(\emptyset)\rfloor = \lfloor\{\{q_0\}\} \cup \{\{q_0\} \star a, \{q_0\} \star b\}\rfloor = \{\{q_0\}, \{q\}\}$,

(3) $\lfloor P^3_{\mathcal{C},\mathcal{D}}(\emptyset)\rfloor = \lfloor\{\{q_0\}\} \cup \{\{q_0\} \star a, \{q_0\} \star b, \{q\} \star a, \{q\} \star b\}\rfloor = \{\{q_0\}, \{q\}\}$.

Hence, $\lfloor P^3_{\mathcal{C},\mathcal{D}}(\emptyset)\rfloor = \lfloor P^2_{\mathcal{C},\mathcal{D}}(\emptyset)\rfloor$ and the computations for the prefix iterates stop at the third iteration.

Next, the algorithm evaluates the sequence $\{\lfloor R^n_{\mathcal{C},\mathcal{D}}(\emptyset)\rfloor\}_{n\in\mathbb{N}} \in (\wp(\wp(Q^2_\mathcal{D}) \times \wp(Q^2_\mathcal{D})))^\mathbb{N}$. Let $y \triangleq \{(q_0, q), (q, q)\}$, $z_1 \triangleq \{(q_0, q_0), (q, q_0)\}$ and $z_2 \triangleq \{(q, q_0)\}$. We have that $y, z_1, z_2 \in \wp(Q^2_\mathcal{D})$, $y = c^\mathcal{D}[a] = f^\mathcal{D}[a]$, $z_1 = c^\mathcal{D}[b]$, $z_2 = f^\mathcal{D}[b]$ and $\{(c^\mathcal{D}(c), f^\mathcal{D}(c)) \mid i \xrightarrow{c} i \wedge c \in \{a, b\}\} = \{(y, y), (z_1, z_2)\}$. For each pair $p = (p_1, p_2) \in \wp(Q^2_\mathcal{D}) \times \wp(Q^2_\mathcal{D})$ and each $c \in \Sigma^*$, we define $p * c \triangleq p_1 \circ f^\mathcal{D}(c) \cup p_2 \circ c^\mathcal{D}(c) \in \wp(Q^2_\mathcal{D})$. We then have:

(1) $R^1_{\mathcal{C},\mathcal{D}}(X) = \{(y, y), (z_1, z_2)\} \cup \mathrm{Post}^{\preccurlyeq^\mathcal{D}}_\mathcal{C}(X)$

$$= \{(y, y), (z_1, z_2)\} \cup \{(p_1 \circ c^\mathcal{D}[c], p * c) \mid i \xrightarrow{c} i \wedge c \in \{a, b\} \wedge (p_1, p_2) \in X\},$$

so that $\lfloor R^1_{\mathcal{C},\mathcal{D}}(\emptyset)\rfloor = \lfloor\{(y, y), (z_1, z_2)\}\rfloor = \{(y, y), (z_1, z_2)\}$.

(2) $\lfloor R^2_{\mathcal{C},\mathcal{D}}(\emptyset)\rfloor =$

$$\lfloor\{(y,y),(z_1,z_2)\}\cup\{(y\circ c^{\mathcal{D}}[a],(y,y)*a),(z_1\circ c^{\mathcal{D}}[a],(z_1,z_2)*a),$$
$$(y\circ c^{\mathcal{D}}[b],(y,y)*b),(z_1\circ c^{\mathcal{D}}[b],(z_1,z_2)*b)\}\rfloor =$$
$$\lfloor\{(y,y),(z_1,z_2)\}\cup\{(y,y),(z_1,z_2),(z_1,z_1)\}\rfloor.$$

Since $(z_1,z_2)\subseteq^2(z_1,z_1)$, $\lfloor R^2_{\mathcal{C},\mathcal{D}}(\emptyset)\rfloor = \lfloor\{(y,y),(z_1,z_2),(z_1,z_1)\}\rfloor = \{(y,y),(z_1,z_2)\}$.

Thus, $\lfloor R^2_{\mathcal{C},\mathcal{D}}(\emptyset)\rfloor = \lfloor R^1_{\mathcal{C},\mathcal{D}}(\emptyset)\rfloor$ and the computations for the period iterates stop at the second iteration.

It turns out that $\neg\mathrm{Inc}^{\mathcal{D}}(\{q\},(z_1,z_2))$: this, intuitively, corresponds to the counterexample ab^ω that belongs to $L^\omega(\mathcal{C})$ but not $L^\omega(\mathcal{D})$. Hence, the inclusion $L^\omega(\mathcal{C})\subseteq L^\omega(\mathcal{D})$ does not hold.

6 Implementation and Experimental Evaluation

Benchmarks. We collected new benchmarks from various trusted sources that significantly expand the set of problem instances available to the research community on language inclusion. In this section, a benchmark means an ordered pair of BAs.

The first set of benchmarks consists of verification tasks defined together with the early versions of the RABIT tool [37]. The BAs are models of mutual exclusion algorithms [2], where in each benchmark one BA is the result of translating a set of guarded commands defining the protocol while the other BA translates a modified set of guarded commands, typically obtained by randomly weakening or strengthening one guard. The resulting BAs are on the binary alphabet $\{0,1\}$ and their sizes range from 20 to 7 963 states. Even though more details about transition labels and acceptance conditions are given [1, 2], it is unclear which basic properties this reduction satisfies, for instance, whether inclusion is preserved when the modified version of the protocol is the result of adding to the original version some “nop” statements. Moreover, we are not aware of any use of this reduction other than generating the RABIT examples.

Our second collection of benchmarks stems from an automated theorem prover for combinatorics on words called Pecan [34]. Here, BAs encode sets of solutions of predicates, hence logical implication between predicates reduces to a language inclusion problem between BAs. The benchmarks correspond to theorems of type $\forall x,\ P(x)\to Q(x)$ about Sturmian words [19]. We collected 58 benchmarks from Pecan for which inclusion holds, where these BAs have alphabets of varying size (from 3 to 256) and their sizes range from 1 to 21 395 states. The third collection of benchmarks stems from software verification. Ultimate Automizer (UA) [17, 18] is a well-known software model checker that verifies program correctness using automata-based reasoning, and that reduces termination problems to inclusion problems between BAs. Overall, we collected 600 benchmarks from UA for which inclusion holds. The BAs have alphabets of varying size (from 6 to 13 173) and sizes ranging from 3 to 6 972 states.

The addition of the Pecan and UA benchmarks significantly expands the set of available benchmarks while, at the same time, increases the diversity of their provenance. This set of benchmarks, which is available on GitHub [11], is biased towards instances where inclusion holds (as opposed to instances where inclusion does not hold). The rationale for this choice is that non-inclusion should be somehow viewed as a separate problem. This claim is supported by the existence of orthogonal approaches explicitly devoted to the non-inclusion problem [30] and specifically tailored approaches and optimizations within tools, like in RABIT. Nevertheless, let us remark that our approach decides the generic inclusion problem and has been evaluated on both positive and negative instances.

Table 1 Runtime in milliseconds on the BAs of Fig. 2. M/O means memory out.

Value of n	1	10	100	1 000	10 000	20 000	30 000	40 000	50 000
BAIT	34	48	100	531	92 102	342 526	821 234	1 284 618	2 074 829
RABIT	75	71	114	919	55 247	M/O	M/O	M/O	M/O

Table 2 Runtimes for RABIT benchmarks in millisec. GOAL$^-$ is Piterman inclusion algorithm without simulations (invocation flag `containment -m piterman`). M/O means memory out.

	Included									Not-included				
Tools	bk	bkv2	Fis	Fisv2	Fisv3	Fisv4	mcs	Pet	Φ	bkv3	Fisv5	Φv2	Φv3	Φv4
RABIT	2 220	4 552	2 260	213	3 985	1 271	49 193	71	136	2 697	6 002	334	265	287
ROLL	4 340	7 590	4 170	1 910	6 690	3 320	14 900	690	1 000	750	600	290	370	310
GOAL	88 320	71 090	128 680	3 500	41 620	18 120	456 480	1 380	2 260	2 450	12 580	1 480	1 510	2 260
GOAL$^-$	M/O	M/O	857 840	6 470	306 370	75 960	3 194 940	2 240	4 360	48 010	85 880	4 110	2 430	2 960
BAIT	1 180 770	M/O	M/O	1 153	654 385	M/O	M/O	321	278 794	6 347	M/O	1 749	997	65 629

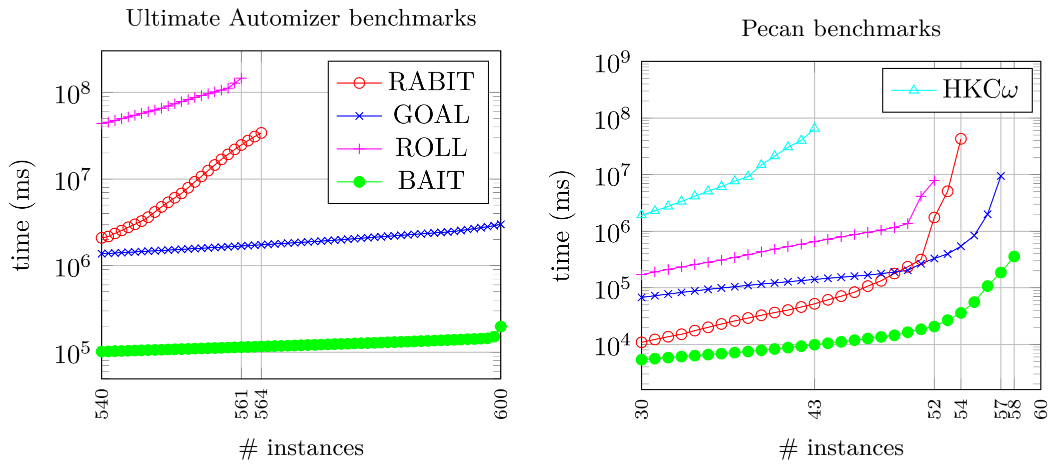
Tools. We implemented our state-based inclusion algorithm BAIncS in a tool called BAIT, developed in Java and which is available on GitHub [10]. We compared BAIT with the following language inclusion checking tools: RABIT 2.5.0, ROLL 1.0, GOAL (20200822), and HKCω (fall 2018). RABIT [7] implements a Ramsey-based algorithm and an advanced preprocessor using simulation relations. ROLL [27, 28] also uses the preprocessor of RABIT but then it relies on automata learning and word sampling techniques to decide inclusion. GOAL [41] implements a "complement-then-intersect-and-check-emptiness" approach using advanced complementation algorithms for BAs. HKCω [24] decides inclusion using up-to techniques. Further details on these tools are given in App. C.

Results. We ran our experiments on a server with 20 GB of RAM, 2 Xeon E5640 2.6 GHz CPUs and Debian stretch 64 bit. In what follows, "left"/"right" BAs refer, resp., to the automata on the left/right of a language inclusion instance.

We start with the following research question: *What is the impact in having separate qos for prefixes and periods?* To answer it, we first examine the performance of BAIT on the contrived family of examples of Fig. 2. In this set of instances, almost no computation is carried out in the fixpoints for the periods ($R_{\mathcal{A},\mathcal{B},p}$ of BAIncS), since they converge in one iteration. Tab. 1 displays the corresponding runtime comparison with RABIT, which processes prefixes and periods the same way. It turns out that for sufficiently large values of n, RABIT runs out of memory while BAIT safely terminates (in max 35 minutes).

Beyond the contrived family of BAs of Fig. 2, we claim that reasoning with separate qos for prefixes and periods gives an advantage to BAIT. Actually, we found that BAIT is the state-of-the-art on all but the RABIT benchmarks. On the RABIT benchmarks, Tab. 2 shows that BAIT runs out of memory on 4/9 of the included benchmarks and on 1/5 of the not-included benchmarks. On these benchmarks, simulation relations are key enablers for RABIT, ROLL and GOAL. Since the pair of BAs in each benchmark stems from two close revisions of the same mutual exclusion protocol, it turns out that the simulation relations being used retain enough information to dramatically lower the effort of showing inclusion (in many cases, these simulation relations alone are sufficient to show language inclusion).

To interpret these outcomes for BAIT, we looked at the graph structure of the "left" BAs of these RABIT benchmarks and we found that they roughly consist of one large strongly connected component (SCC): this is expected since these BAs model agents running a mutual exclusion protocol in an infinite loop. The computations of BAIT on these benchmarks are



(a) Plot not shown between 1 and 539 for clarity. HKCω not depicted: more than 60 memory out (8 GB virtual memory limit).

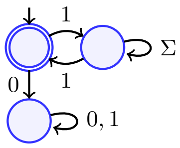
(b) Plot not shown between 1 and 29 for clarity.

Figure 3 Each benchmark has a timeout value of 12h. Survival plot with a logarithmic y axis and linear x axis. No plot for abscissa value x and tool r means that, for $60-x$ Pecan benchmarks (or $600-x$ for the case of Ultimate), r did not return an answer.

dominated by the fixpoints of $R_{\mathcal{A},\mathcal{B},p}$ for the periods, which compute over *sets of pairs of states*, as opposed to the fixpoints of $P_{\mathcal{A},\mathcal{B}}$ for the prefixes which compute over *sets of states*. Also, the computations for $R_{\mathcal{A},\mathcal{B},p}$ dominate the memory use. However, this scenario of BAs consisting of one large SCC does not occur for the other benchmarks: the "left" automata for Pecan and UA benchmarks tend to have few SCCs including final states and each of them are rather small. Here, BAIT is at an advantage because most computations are carried out on the fixpoints of $P_{\mathcal{A},\mathcal{B}}$ for the prefixes.

Because of the large number of available Pecan (60) and UA (600) benchmarks, we use survival plots for displaying our experimental results. Let us recall how to obtain them for a family of benchmarks $\{p_i\}_{i=1}^n$: (1) run the tool on each benchmark p_i and store its runtime t_i (or timeout event); (2) sort the t_i's in increasing order (discarding the timeouts); (3) plot the points $(t_1, 1), (t_1 + t_2, 2), \ldots$, and in general $(\sum_{i=1}^k t_i, k)$; (4) repeat for each tool under evaluation. The runtimes for BAIT include a phase of preprocessing that reduced the set of final states of the "left" BA while preserving the accepted language, in accordance with Remark 4.8. This preprocessing used the function `acc -min` of the tool GOAL, a polynomial time algorithm that relies on computing SCCs. The survival plots in compact form are depicted in Fig. 3 and with more detail in App. D.

These results show that BAIT is the state-of-the-art approach for the Pecan and UA benchmarks. They also show that GOAL performs quite well on the Pecan and UA benchmarks compared to RABIT and ROLL whose approaches are less efficacious. This is expected because both Pecan and UA rely on complementation for their decision procedure, so that they produce their "right" BAs through some heuristics to make them easy to complement (as confirmed to us by the developers of Pecan and UA). Indeed, we claim that GOAL's performance quickly degrades when the "right" BAs are hard to complement. Our claim is supported by Fig. 4 where GOAL and BAIT are compared on a contrived family of benchmarks based on Michel's family of hard to complement BAs (see [33] and [39, Theorem 5.3] for further details).



	$\Sigma = \{0,1,2,3\}$		$\Sigma = \{0,1,2,3,4\}$		$\Sigma = \{0,1,2,3,4,5\}$	
Value of n	BAIT	GOAL$^-$	BAIT	GOAL$^-$	BAIT	GOAL$^-$
3	45	13 550	$\not\subseteq$	$\not\subseteq$	$\not\subseteq$	$\not\subseteq$
4	45	14 040	77	9 187 680	$\not\subseteq$	$\not\subseteq$
5	46	13 120	89	9 148 710	225	T/O
10	54	13 840	100	T/O	291	T/O
100	123	17 020	282	T/O	1 301	T/O

Figure 4 Runtime in milliseconds using Michel's family for the "right" BAs (parameterized by n) and the depicted BA for the "left". GOAL$^-$ refers to `containment -m piterman` (as in Table 2). $\not\subseteq$ means not included, T/O is time out (12h).

7 Conclusion and Future Work

We designed a family of algorithms for the inclusion problem between ω-regular and ω-context-free languages into ω-regular languages, represented by automata. Our algorithms are conceptually simple: least fixpoint computations for the languages of finite prefixes and periods of ultimately periodic infinite words. The functions to iterate for these fixpoints are readily derived from the "left" automaton and the fixpoints converge in finitely many iterations thanks to a well-quasiorder abstraction on words. Finally, language inclusion is decided by a straightforward membership check. The height of the lattices of our least fixpoint computations allows us to derive some information about the worst case complexity of our algorithms. For each least fixpoint computation performed at line 3 of BAIncS, the worst case is adding exactly one element in a subset of $\wp(Q^2_{\mathcal{B}}) \times \wp(Q^2_{\mathcal{B}})$ to some entry of the $|Q_{\mathcal{A}}|$-dimensional vector at each iteration step, so that $|Q_{\mathcal{A}}| \times 2^{2|Q_B|^2}$ is an upper bound on N_2 in BAIncS. We leave as future work a detailed worst case complexity analysis of our algorithms. In practice, a simple Java implementation of our inclusion algorithm was competitive against state-of-the-art tools, thus showing the benefits of having separate well-quasiorders for prefixes and periods. We expect that this latter approach can be further refined using, for instance, family of right-congruences [31], paving the way to even more efficient inclusion algorithms.

References

1 Parosh Aziz Abdulla, Yu-Fang Chen, Lorenzo Clemente, Lukáš Holík, Chih-Duo Hong, Richard Mayr, and Tomáš Vojnar. Simulation Subsumption in Ramsey-Based Büchi Automata Universality and Inclusion Testing. In *Proc. Int. Conf. on Computer Aided Verification (CAV)*. Springer, 2010. `doi:10.1007/978-3-642-14295-6_14`.

2 Parosh Aziz Abdulla, Yu-Fang Chen, Lorenzo Clemente, Lukáš Holík, Chih-Duo Hong, Richard Mayr, and Tomáš Vojnar. Advanced Ramsey-Based Büchi Automata Inclusion Testing. In *Proc. Int. Conf. on Concurrency Theory (CONCUR)*. Springer LNCS, 2011. `doi:10.1007/978-3-642-23217-6_13`.

3 Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In *Proc. of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. ACM, 2013. `doi:10.1145/2429069.2429124`.

4 Filippo Bonchi and Damien Pous. Hacking nondeterminism with induction and coinduction. *Commun. ACM*, 58(2), 2015. `doi:10.1145/2713167`.

5 Hugues Calbrix, Maurice Nivat, and Andreas Podelski. Ultimately periodic words of rational ω-languages. In *Proc. Int. Symp. on Mathematical Foundations of Programming Semantics (MFPS)*, LNCS. Springer, 1994. `doi:10.1007/3-540-58027-1\_27`.

6 Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem. *Handbook of Model Checking*. Springer, 1st edition, 2018. `doi:10.1007/978-3-319-10575-8`.

7 Lorenzo Clemente and Richard Mayr. Efficient reduction of nondeterministic automata with application to language inclusion testing. *Logical Methods in Computer Science*, 15(1), 2019. doi:10.23638/LMCS-15(1:12)2019.

8 Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM Symp. on Principles of Programming Languages (POPL)*. ACM, 1977. doi:10.1145/512950.512973.

9 Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Proc. of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*. ACM, 1979. doi:10.1145/567752.567778.

10 K. Doveri, P. Ganty, F. Parolini, and F. Ranzato. BAIT: Büchi Automata Inclusion Tester. https://github.com/parof/bait, 2021.

11 K. Doveri, P. Ganty, F. Parolini, and F. Ranzato. Büchi Automata benchmarks for language inclusion. https://github.com/parof/buchi-automata-benchmark, 2021.

12 Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proc. 14th Int. Symp. on Automated Technology for Verification and Analysis (ATVA)*, LNCS. Springer, 2016. doi:10.1007/978-3-319-46520-3_8.

13 Javier Esparza. Automata theory – An algorithmic approach. Lecture Notes, 2017. URL: https://www7.in.tum.de/~esparza/autoskript.pdf.

14 Seth Fogarty and Moshe Y. Vardi. Efficient Büchi Universality Checking. In *Proc. Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS. Springer, 2010. doi:10.1007/978-3-642-12002-2_17.

15 Pierre Ganty, Francesco Ranzato, and Pedro Valero. Language inclusion algorithms as complete abstract interpretations. In *Proc. 26th Int. Static Analysis Symposium (SAS)*, LNCS. Springer, 2019. doi:10.1007/978-3-030-32304-2_8.

16 Pierre Ganty, Francesco Ranzato, and Pedro Valero. Complete abstractions for checking language inclusion. *ACM Trans. on Computational Logic*, To appear, 2021. arXiv:1904.01388.

17 Matthias Heizmann, Yu-Fang Chen, Daniel Dietsch, Marius Greitschus, Jochen Hoenicke, Yong Li, Alexander Nutz, Betim Musa, Christian Schilling, Tanja Schindler, and Andreas Podelski. Ultimate Automizer and the search for perfect interpolants – (competition contribution). In *Proc. 24th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS. Springer, 2018. doi:10.1007/978-3-319-89963-3_30.

18 Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Software model checking for people who love automata. In *Proc. Int. Conf. on Computer Aided Verification (CAV)*. Springer LNCS, 2013. doi:10.1007/978-3-642-39799-8_2.

19 Philipp Hieronymi, Dun Ma, Reed Oei, Luke Schaeffer, Zhengyao Lin, Christian Schulz, and Jeffrey Shallit. Decidability for Sturmian words, 2021. arXiv:2102.08207.

20 Martin Hofmann and Wei Chen. Abstract interpretation from Büchi automata. In *Proc. of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. ACM Press, 2014. doi:10.1145/2603088.2603127.

21 Takumi Kasai and Shigeki Iwata. Some problems in formal language theory known as decidable are proved EXPTIME complete, 1992. URL: https://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/0796-02.pdf.

22 Bakhadyr Khoussainov and Nerode, Anil. *Automata Theory and Its Applications*. Springer, 2001. doi:10.1007/978-1-4612-0171-7.

23 Denis Kuperberg, Laureline Pinault, and Damien Pous. HKCω: Coinductive algorithms for Büchi automata. http://perso.ens-lyon.fr/damien.pous/covece/hkcw/, 2018.

24 Denis Kuperberg, Laureline Pinault, and Damien Pous. Coinductive Algorithms for Büchi Automata. In *Proc. Int. Conf. on Developments in Language Theory (DLT)*, LNCS. Springer, 2019. doi:10.1007/978-3-030-24886-4_15.

25 Orna Kupferman. Automata Theory and Model Checking. In *Handbook of Model Checking*. Springer, 2018. doi:10.1007/978-3-319-10575-8_4.

26 Orna Kupferman and Moshe Y. Vardi. Verification of fair transition systems. In *Proc. Int. Conf. on Computer Aided Verification (CAV)*. Springer, 1996. doi:10.1007/3-540-61474-5_84.

27 Yong Li, Yu-Fang Chen, Lijun Zhang, and Depeng Liu. A novel learning algorithm for Büchi automata based on family of DFAs and classification trees. *Information and Computation*, 2020. doi:10.1016/j.ic.2020.104678.

28 Yong Li, Xuechao Sun, Andrea Turrini, Yu-Fang Chen, and Junnan Xu. ROLL 1.0: ω-regular language learning library. In *Proc. 25th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS. Springer, 2019. doi:10.1007/978-3-030-17462-0_23.

29 Yong Li and Andrea Turini. Roll library: Regular Omega Language Learning library. https://github.com/ISCAS-PMC/roll-library, 2020.

30 Yong Li, Andrea Turrini, Xuechao Sun, and Lijun Zhang. Proving non-inclusion of Büchi automata based on Monte Carlo sampling. In *Proc. 14th Int. Symp. on Automated Technology for Verification and Analysis (ATVA)*. Springer, 2020. doi:10.1007/978-3-030-59152-6_26.

31 Oded Maler and Ludwig Staiger. On syntactic congruences for ω-languages. Technical report, Verimag, France, 2008. URL: http://www-verimag.imag.fr/~maler/Papers/congr.pdf.

32 Roland Meyer, Sebastian Muskalla, and Elisabeth Neumann. Liveness verification and synthesis: New algorithms for recursive programs, 2017. arXiv:1701.02947.

33 M. Michel. Complementation is more difficult with automata on infinite words. Technical report, CNET, Paris, 1988.

34 Reed Oei, Dun Ma, Christian Schulz, and Philipp Hieronymi. Pecan: An automated theorem prover for automatic sequences using Büchi automata, 2021. arXiv:2102.01727.

35 Dominique Perrin and Jean Eric Pin. *Infinite Words: Automata, Semigroups, Logic and Games*. Number 141 in Pure and Applied Mathematics Series. Elsevier, Amsterdam ; Boston, 1st edition, 2004.

36 Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3(3), 2007. doi:10.2168/lmcs-3(3:5)2007.

37 RABIT/Reduce: Tools for language inclusion testing and reduction of nondeterministic Büchi automata and NFA. http://www.languageinclusion.org/doku.php?id=tools. Accessed: 2021-01-29.

38 Roman R. Redziejowski. An improved construction of deterministic omega-automaton using derivatives. *Fundamenta Informaticae*, 119(3–4), 2012. doi:10.3233/FI-2012-744.

39 Wolfgang Thomas. Languages, Automata, and Logic. In *Handbook of Formal Languages: Volume 3 Beyond Words*. Springer, 1997. doi:10.1007/978-3-642-59126-6_7.

40 Ming-Hsien Tsai, Seth Fogarty, Moshe Vardi, and Yih-Kuen Tsay. State of Büchi complementation. *Logical Methods in Computer Science*, 10(4), 2014. doi:10.2168/lmcs-10(4:13)2014.

41 Ming-Hsien Tsai, Yih-Kuen Tsay, and Yu-Shiang Hwang. GOAL for Games, Omega-Automata, and Logics. In *Proc. Int. Conf. on Computer Aided Verification (CAV)*. Springer, 2013. doi:10.1007/978-3-642-39799-8_62.

42 William M. Waite and Gerhard Goos. *Compiler Construction*. Springer-Verlag, New York, USA, 1984.

A Generalised Word-Based Algorithm

We give a generalised word-based algorithm gBAIncW, briefly explained in Section 4.1, which computes sequences of pruned Kleene iterates $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_{p,n}\}_{n\in\mathbb{N}}$, for each $p \in F$. We obtain the correcteness of BAIncW as a consequence of the correcteness of gBAIncW.

▶ **Theorem A.1.** *Given all the required input data,* gBAIncW *decides* $L^\omega(\mathcal{A}) \subseteq M$*.*

gBAIncW Word-based algorithm for checking $L^\omega(\mathcal{A}) \subseteq M$.

Data: Büchi automaton $\mathcal{A} = (Q, \delta, i_{\mathcal{A}}, F)$
Data: Procedure deciding $uv^\omega \in^? M$ given $u, v \in \Sigma^*$
Data: Decidable right-monotonic wqos $\leq, \preccurlyeq$ s.t. $\rho_{\leq\times\preccurlyeq}(I_M) = I_M$
Data: For each $p \in F$ and $n \in \mathbb{N}$, sequences $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_{p,n}\}_{n\in\mathbb{N}}$ in $\wp(\Sigma^*)^{|Q|}$ s.t. $\rho_{\leq}(P^n_{\mathcal{A}}(\emptyset)) = \rho_{\leq}(X_n)$ and $\rho_{\preccurlyeq}(R^n_{\mathcal{A},p}(\emptyset)) = \rho_{\preccurlyeq}(Y_{p,n})$.

```
Compute X_{N_1} with least N_1 s.t. ∀q ∈ Q, ρ_≤((X_{N_1+1})_q) ⊆ ρ_≤((X_{N_1})_q)
foreach p ∈ F do
    Compute Y_{p,N_2} with least N_2 s.t. ∀q ∈ Q, ρ_≼((Y_{p,N_2+1})_q) ⊆ ρ_≤((Y_{p,N_2})_q)
    foreach u ∈ (X_{N_1})_p, v ∈ (Y_{p,N_2})_p do
        if uv^ω ∉ M then return false;
return true;
```

B State-Based Algorithm for ω-context-free $\subseteq$ ω-regular

We derive a state-based inclusion algorithm that, given a BPDA $\mathcal{P} = (Q_{\mathcal{P}}, \Gamma, \delta_{\mathcal{P}}, i_{\mathcal{P}}, F_{\mathcal{P}})$ and a BA $\mathcal{B} = (Q_{\mathcal{B}}, \delta_{\mathcal{B}}, i_{\mathcal{B}}, F_{\mathcal{B}})$, decides whether $L^\omega(\mathcal{P}) \subseteq L^\omega(\mathcal{B})$ holds or not by operating on the states of $\mathcal{P}$ and $\mathcal{B}$ only. Similarly to the ω-regular case, words and operations/tests on words are abstracted, resp., into states and operations/tests on states, using the state-based qos derived from $\mathcal{B}$, as explained in Section 3. Recall that in the context-free case we need qos that are both right- and left- monotonic. Hence, we consider the pair of qos $\trianglelefteq^{\mathcal{B}}, \preccurlyeq^{\mathcal{B}}$ (see Section 3).

Given a CFG $\mathcal{G} = (V, P)$ in CNF, we define the functions $R_{1,\mathcal{G}}$ over $\wp(Q^2_{\mathcal{B}})^V$ and $R_{2,\mathcal{G}}$ over $(\wp(\wp(Q^2_{\mathcal{B}}) \times \wp(Q^2_{\mathcal{B}})))^V$ as follows:

$$\begin{aligned}
R^1_{\mathcal{G}}(S) &\triangleq \langle \{x \circ y \mid \exists X_j \to X_k X_l \in P, x \in S_k \wedge y \in S_l\}\rangle_{j\in[0,n]} \ , \\
R^2_{\mathcal{G}}(S) &\triangleq \langle \{(x_1 \circ y_1, (x_1 \circ y_2) \cup (x_2 \circ y_1)) \mid \exists X_j \to X_k X_l \in P, \\
&\qquad\qquad (x_1, x_2) \in S_k, (y_1, y_2) \in S_l\}\rangle_{j\in[0,n]} \ .
\end{aligned}$$

Let us define the vectors $b^1_{\mathcal{G}} \in \wp(Q^2_{\mathcal{B}})^V$ and $b^2_{\mathcal{G}} \in (\wp(\wp(Q^2_{\mathcal{B}}) \times \wp(Q^2_{\mathcal{B}})))^V$ as follows:

$$\begin{aligned}
b^1_{\mathcal{G}} &\triangleq \langle \{c^{\mathcal{B}}[\beta] \mid X_j \to \beta, \beta \in \Sigma \cup \{\epsilon\}\}\rangle_{j\in[0,n]} \ , \\
b^2_{\mathcal{G}} &\triangleq \langle \{(c^{\mathcal{B}}[\beta], f^{\mathcal{B}}[\beta]) \mid X_j \to \beta, \beta \in \Sigma \cup \{\epsilon\}\}\rangle_{j\in[0,n]} \ .
\end{aligned}$$

Let $\mathcal{P}^1_{q\gamma}$ and $\mathcal{P}^2_{q\gamma}$, for each $q \in Q_{\mathcal{P}}$ and $\gamma \in \Gamma$, be the two PDAs defined from $\mathcal{P}$ and such that the ultimately periodic words generated by the pairs in $\bigcup_{(q,\gamma)\in Q\times\Gamma} L^*(\mathcal{P}^1_{q,\gamma}) \times L^*(\mathcal{P}^2_{q,\gamma})$ coincide with the ultimately periodic words in $L^\omega(\mathcal{P})$. Let $\mathcal{G}^1_{q\gamma} \triangleq \mathrm{PDA2CFG}(\mathcal{P}^1_{q\gamma})$ and $\mathcal{G}^2_{q\gamma} \triangleq \mathrm{PDA2CFG}(\mathcal{P}^2_{q\gamma})$, where PDA2CFG is a procedure to convert a PDA into a CFG in CNF. For each $q \in Q_{\mathcal{P}}$ and each $\gamma \in \Gamma$, we define the functions $P_{q\gamma\mathcal{B}} \triangleq \lambda X. b^1_{\mathcal{G}^1_{q\gamma}} \cup R^1_{\mathcal{G}^1_{q\gamma}}(X)$ and $R_{q\gamma\mathcal{B}} \triangleq \lambda X. b^2_{\mathcal{G}} \cup R^2_{\mathcal{G}^2_{q\gamma}}(X)$. Let us define the following state-abstraction of the membership test:

$$\mathrm{Inc}^{\mathcal{B}cf}(x, y_1, y_2) \triangleq \exists p, q \in Q_{\mathcal{B}}, (i_{\mathcal{B}}, p) \in x \wedge (p, q) \in y_1^* \wedge (q, q) \in y_1^* \circ y_2 \circ y_1^* \ .$$

▶ **Lemma B.1.** $uv^\omega \in L^\omega(\mathcal{B}) \iff \mathrm{Inc}^{\mathcal{B}cf}(c^{\mathcal{B}}[u], c^{\mathcal{B}}[v], f^{\mathcal{B}}[v])$.

Algorithm BPDAIncS: State-based algorithm for $L^\omega(\mathcal{P}) \subseteq L^\omega(\mathcal{B})$.

Data: BPDA $\mathcal{P} = (Q, \Gamma, \delta, q_0, Z_0, F)$ and BA $\mathcal{B} = (Q_\mathcal{B}, \delta_\mathcal{B}, i_\mathcal{B}, F_\mathcal{B})$

foreach $q \in Q,\ \gamma \in \Gamma$ **do**
    $\mathcal{G}_1 := \mathrm{PDA2CFG}(\mathcal{P}^1_{[q\gamma]})$; $\mathcal{G}_2 := \mathrm{PDA2CFG}(\mathcal{P}^2_{[q\gamma]})$;
    Compute $\lfloor P^{N_1}_{q\gamma\mathcal{B}} \rfloor$ with least N_1 s.t. $\forall j \in V_{\mathcal{G}^1_{q\gamma}}$, $\rho_\subseteq((P^{N_1+1}_{q\gamma\mathcal{B}}(\emptyset))_j) \subseteq \rho_\subseteq((P^{N_1}_{q\gamma\mathcal{B}}(\emptyset))_j)$;
    Compute $\lfloor R^{N_2}_{q\gamma\mathcal{B}} \rfloor$ with least N_2 s.t. $\forall j \in V_{\mathcal{G}^2_{q\gamma}}$, $\rho_{\subseteq^2}((R^{N_2+1}_{q\gamma\mathcal{B}}(\emptyset))_j) \subseteq \rho_{\subseteq^2}((R^{N_2}_{q\gamma\mathcal{B}}(\emptyset))_j)$;
    foreach $x \in (\lfloor P^{N_1}_{q\gamma\mathcal{B}} \rfloor)_0,\ (y_1, y_2) \in (\lfloor R^{N_2}_{q\gamma\mathcal{B}} \rfloor)_0$ **do**
        if $\neg\mathrm{Inc}^{\mathcal{B}cf}(x, y_1, y_2)$ **then return false**;
return true;

▶ **Theorem B.2.** *Given a BPDA $\mathcal{P}$ and BA $\mathcal{B}$,* BPDAIncS *decides $L^\omega(\mathcal{P}) \subseteq L^\omega(\mathcal{B})$.*

C Language Inclusion Checking Tools

RABIT [7] consists of about 20K lines of Java code and its source code is publicly available [37]. To check a language inclusion RABIT combines several techniques controlled via command line options. In our experiments we ran RABIT with options `-fast -jf` which RABIT states as providing the "best performance". Roughly speaking, RABIT performs the following operations: (1) Removing dead states and minimizing the automata with simulation-based techniques, thus yielding a smaller instance; (2) Witnessing inclusion by simulation already during the minimization phase; (3) Using the Ramsey-based method to witness inclusion or non-inclusion.

ROLL [27, 28] contains an inclusion checker that does a preprocessing similar to that of RABIT and then relies on automata learning and word sampling techniques to decide inclusion. ROLL consists of about 19K lines of Java code which is publicly available [29].

GOAL [41] contains several language inclusion checkers available with multiple options. We used the Piterman check (`containment -m piterman -sim -pre` on the command line) that constructs on-the-fly the intersection of the "left" BA and the complement of the "right" BA which is itself built on-the-fly by the Piterman construction [36]. The options `-sim -pre` compute and use simulation relations to further improve performance. The Piterman check was deemed the "best effort" (cf. [7, Section 9.1] and [40]) among the inclusion checkers provided in GOAL. GOAL is written in Java and the source code of the release we used is not publicly available.

HKCω [24] includes an inclusion checker using the so-called up-to techniques. HKCω consists of 3K lines of OCaml code which is publicly available [23]. Up-to techniques form the state-of-the-art approach to decide equivalence for languages of finite words given by finite state automata [3, 4]. The extension of up-to techniques to ω-words has been implemented in HKCω, although only partially. Indeed, as stated in the code documentation, even if up-to techniques have been defined for both prefixes and periods of ultimately periodic words, HKCω only implements them for prefixes. HKCω also includes some preprocessing of the BAs using simulation relations.

As far as we know all these implementations are sequential except for RABIT which, using the `-jf` option, performs some computations in a separate thread.

BAIT is our implementation of the BAIncS algorithm defined in Section 5. BAIT consists of less than 1 750 lines of Java code. BAIT relies exclusively on a few standard packages from the Java SE Platform, notably standard collections such as HashSet or HashMap. One of the design goals of BAIT was to have simple and unencumbered code. Unlike RABIT, HKCω, ROLL and GOAL, BAIT does not compute or exploit simulation relations. Also, BAIT is implemented as a purely sequential algorithm although some computations are easily parallelizable such as the fixpoints for the prefixes and for the periods.

SPOT We did not consider the Spot tool [12] in our evaluation because we believe GOAL is a better fit in our setting as we argue below. First, Spot works with a symbolic alphabet where symbols are encoded using Boolean propositions, and sets of symbols are represented and processed using OBDDs. We used GOAL in the classical alphabet mode where symbols are explicitly represented as in ROLL, RABIT and BAIT. Second, the inclusion algorithm of Spot complements the "right" BA using Redziejowski's method with some additional optimizations including simulation-based optimizations [12]. GOAL implements Piterman's complementation method [36], which inspired that of Redziejowski [38]. The Piterman's method of GOAL also offers simulation-based optimizations and, furthermore, GOAL specialized Piterman's method to the inclusion problem by constructing on-the-fly the intersection of the "left" automaton and the complement of the "right" automaton constructed on-the-fly by the Piterman's method [40]. Finally, Spot is written in C++ while GOAL is written in Java as ROLL, RABIT and BAIT, thus making their runtime comparison more meaningful.

Experimental Setup. We ran our experiments on a server with 20 GB of RAM, 2 Xeon E5640 2.6 GHz CPUs and Debian stretch 64 bit. We used openJDK 11.0.9.1 2020-11-04 when compiling Java code and ran the JVM with default options. For RABIT and BAIT the execution time is computed using timers internal to their implementations. For ROLL and GOAL the execution time is given by the "real" value of the `time(1)` command.

D Detailed Graphs of the Experimental Comparison

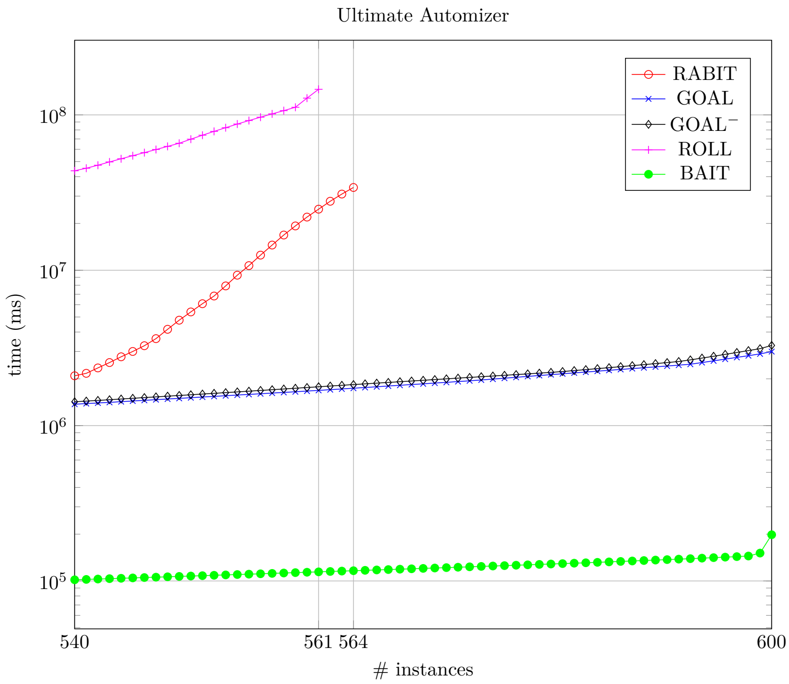


Figure 5 Survival plot with a logarithmic y axis and linear x axis. Plot not depicted between 1 and 539 for clarity. Each benchmark has a timeout value of 12h. No plot for abscissa value x and tool r means that, for $600-x$ benchmarks, r did not return an answer (i.e. it either ran out of memory or time). HKCω not depicted: more than 60 memory out (8 GB virtual memory limit).

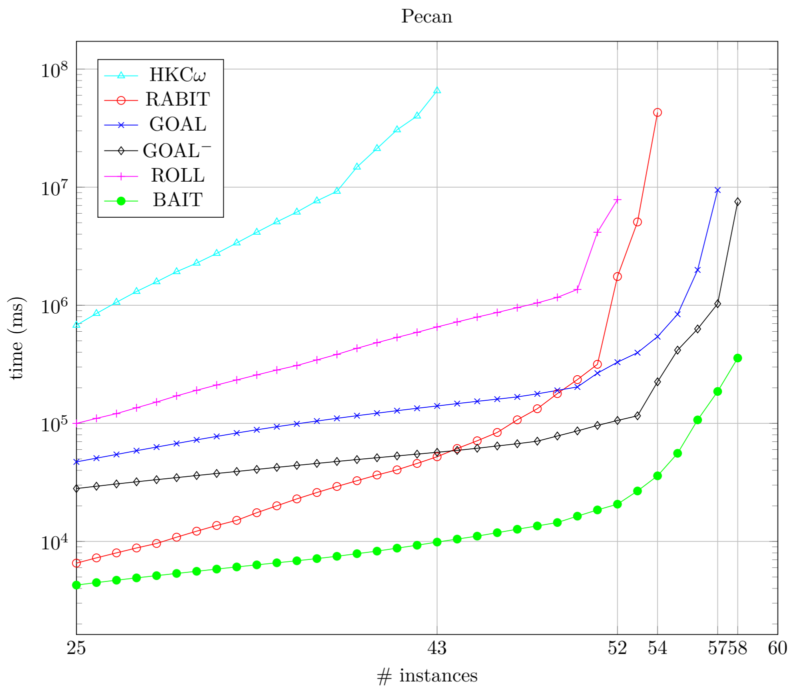


Figure 6 Survival plot with a logarithmic y axis and linear x axis. Plot not depicted between 1 and 24 for clarity. Each benchmark has a timeout value of 12h. No plot for abscissa value x and tool r means that, for $60-x$ Pecan benchmarks, r did not return an answer (i.e., it either ran out of memory or time).