Coinductive algorithms for Büchi automata*

Denis Kuperberg[1], Laureline Pinault[1], and Damien Pous[1]

Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, France

Abstract. We propose a new algorithm for checking language equivalence of non-deterministic Büchi automata. We start from a construction proposed by Calbrix, Nivat and Podelski, which makes it possible to reduce the problem to that of checking equivalence of automata on finite words. Although this construction generates large and highly non-deterministic automata, we show how to exploit their specific structure and apply state-of-the art techniques based on coinduction to reduce the state-space that has to be explored. Doing so, we obtain algorithms which do not require full determinisation or complementation.

Keywords: Büchi automata · Language equivalence · Coinduction.

1 Introduction

Büchi automata are machines which make it possible to recognise sets of infinite words. They form a natural counterpart to finite automata, which operate on finite words. They play a crucial role in logic for their links with monadic second order logic (MSO) [5], and in program verification. For instance, they are widely used in model-checking tools, in order to check whether a given program satisfies a linear temporal logic formula (LTL) [31,13].

A key algorithmic property of Büchi automata is that checking whether two automata recognise the same language is decidable, and in fact PSPACE-complete, like in the finite case with non-deterministic finite automata. This is how one obtains model-checking algorithms. Several algorithms have been proposed in the literature [5,14,1,18] and implemented in various tools [15,30,24].

Two families of algorithms were discovered for non-deterministic automata on finite words, which drastically improved over the pre-existing ones in practice: antichain-based algorithms [32,3,10] and algorithms based on bisimulations up to congruence [4]. In both cases, those algorithms explore the starting automata by resolving non-determinism on the fly through the powerset construction, and they exploit subsumption techniques to avoid the need to explore all reachable states (which can be exponentially many). The algorithms based on bisimulations up to congruence improve over those based on antichains by using simultaneously

* This work has been funded by the European Research Council (ERC) under the European Union's Horizon 2020 programme (CoVeCe, grant agreement No 678157), and was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

the antichain techniques and an older technique for deterministic automata, due to Hopcroft and Karp [17]. Note that both families of algorithms require exponential space (and time) in worst-case complexity, for a problem which is only PSPACE. In practice however, they perform better than existing PSPACE algorithms, because the latter require exponential time even for best cases.

The antichain-based algorithms could be adapted to Büchi automata by exploiting constructions to compute the complement of a Büchi automaton, either Ramsey-based [11,12] or rank-based [9,10]. Unfortunately, those complementation operations do not make it possible to adapt the algorithms based on bisimulations up to congruence: those require a proper powerset construction for determinisation, which is not available for Büchi automata. Here we propose to circumvent this difficulty using a construction by Calbrix, Nivat, and Podelski [6], which makes it possible to reduce the problem of checking Büchi automata equivalence to that of checking equivalence of automata on finite words.

The first observation, which is used implicitly in the so-called Ramsey-based algorithms from the literature [11,12,1], is that it suffices to consider ultimately periodic words: if the languages of two Büchi automata differ, then they must differ on an ultimately periodic word. The second observation is that the set of ultimately periodic words accepted by a Büchi automaton can be faithfully represented as a rational language of finite words, for which Calbrix et al. give an explicit non-deterministic finite automaton. This automaton contains two layers: one for the prefixes of the ultimately periodic words, and one for their periods. We show that algorithms like `HKC` [4] can readily be used to reason about the prefix layer, without systematically determinising it. The period layer requires more work in order to avoid paying a doubly exponential price. We show how to analyse it to compute *discriminating sets* that summarise the periodic behaviour of the automaton, and suffice to check language equivalence.

We first recall the algorithms from [4] for checking equivalence of automata on finite words (Sect. 2). Then we revisit the construction of Calbrix et al., making their use of the Büchi transition monoid [25] explicit (Sect. 3). We define the new algorithm $\texttt{HKC}^\omega$ in Sect. 4. We conclude with directions for future work in Sect. 5.

Notation. We denote sets by capital letters $X, Y, S, T \ldots$ and functions by lower case letters $f, g, \ldots$ Given sets X and Y, $X \times Y$ is their Cartesian product, $X \uplus Y$ is the disjoint union, X^Y is the set of functions $f\colon Y \to X$. The collection of subsets of S is denoted by $\mathcal{P}(S)$. The collection of relations on S is denoted by $Rel(S) = \mathcal{P}(S^2)$. Given a relation $R \in Rel(X)$, we write $x\ R\ y$ for $\langle x, y\rangle \in R$. We fix an arbitrary alphabet A ranged over using lowercase letters a, b. We write A^* for the set of all finite words over A; ϵ the empty word; $w_1 w_2$ the concatenation of words $w_1, w_2 \in A^*$; and $|w|$ for the length of a word w and w_i for its i^{th} letter (when $i < |w|$). We write A^+ for the set of non-empty words and A^ω for the set of infinite words over A. We use 2 for the set $\{0, 1\}$ (Booleans).

A semilattice is a tuple $\langle O, +, 0\rangle$ where O is a set of elements, $+\colon O^2 \to O$ is an associative, commutative and idempotent binary operation, and $0 \in O$ is a neutral element for $+$. For instance, $\langle 2, max, 0\rangle$ is a semilattice. More generally $\langle \mathcal{P}(X), \cup, \emptyset\rangle$ is a semi-lattice for every set X.

2 Coinductive algorithms for finite automata

We will need to work with *Moore machines*, which generalise finite automata by allowing output values in an arbitrary set rather than Booleans. We keep the standard automata terminology for the sake of readability.

A deterministic finite automaton (DFA) over the alphabet A and with outputs in O is a triple $\langle S, o, t\rangle$, where S is a finite set of states, $o\colon S \to O$ is the output function, and $t\colon A \times S \to S$ is the transition function which returns, for each letter $a \in A$ and for each state x, the next state $t_a(x)$. Note that we do not specify an initial state in the definition of DFA: rather than comparing two DFAs, we shall compare two states in a single DFA (obtained as disjoint union if necessary).

Every DFA $\mathcal{A} = \langle S, o, t\rangle$ induces a function $[\cdot]_{\mathcal{A}} : S \to O^{A^*}$, mapping each state to a weighted language with weights in O. This function is defined by $[x]_{\mathcal{A}}(\epsilon) = o(x)$ for the empty word, and $[x]_{\mathcal{A}}(aw) = [t_a(x)]_{\mathcal{A}}(w)$ otherwise. We shall omit the subscript $\mathcal{A}$ when it is clear from the context. For a state x of a DFA, $[x]$ is called the language accepted by x. The languages accepted by some state in a DFA with Boolean outputs are the *rational languages*.

2.1 Deterministic automata: Hopcroft and Karp's algorithm

We fix a DFA $\langle S, o, t\rangle$. Coinductive algorithms for checking language equivalence proceed by trying to find a *bisimulation* relating the given starting states.

Definition 1 (Bisimulation). *Let $g\colon Rel(S) \to Rel(S)$ be the function on relations defined as*

$$g(R) = \{\langle x, y\rangle \mid o(x) = o(y) \text{ and } \forall a \in A,\ t_a(x)\ R\ t_a(y)\}$$

A bisimulation *is a relation R such that $R \subseteq g(R)$.*

The above function g being monotone, it admits the union of all bisimulations as a greatest fixpoint, by Knaster-Tarski's theorem [19,29]. This greatest-fixpoint is actually language equivalence:

Theorem 1. *For all $x, y \in S$, $[x] = [y]$ iff there is a bisimulation R with $x\ R\ y$.*

This theorem yields two families of algorithms: on the one hand, backward algorithms like partition-refinement [16] make it possible to compute the largest bisimulation, and thus to minimise DFA; on the other hand, forward algorithms make it possible to compute the smallest bisimulation containing a given pair of states (if any), and thus to check language equivalence locally, between two states [17]. The latter problem is the one we are interested in in this paper. (Unlike with languages of finite words, there is no canonical notion of minimal automaton for Büchi automata.) For deterministic automata on finite words this problem is slightly easier complexity-wise: when the starting automaton has size n, minimisation can be solved in time $o(n\ln(n))$ while language equivalence of two given states can be tested in almost linear time [28].

input : A DFA $\mathcal{A} = \langle S, o, t\rangle$ and two states $x, y \in S$
output : true if $[x]_{\mathcal{A}} = [y]_{\mathcal{A}}$; false otherwise

```
1 R := ∅; todo := {⟨x, y⟩};
2 while todo ≠ ∅ do
      // invariant: ⟨x, y⟩ ∈ R ⊆ g(f(R ∪ todo))
3     extract ⟨x′, y′⟩ from todo;
4     if o(x′) ≠ o(y′) then return false;
5     if ⟨x′, y′⟩ ∈ f(R ∪ todo) then skip;
6     forall a ∈ A do
7         insert ⟨t_a(x′), t_a(y′)⟩ in todo;
8     insert ⟨x′, y′⟩ in R;
9 return true; // because: ⟨x, y⟩ ∈ R ⊆ g(f(R))
```

Fig. 1. Coinductive algorithm for language equivalence in a DFA; the function f on line 5 ranges over the identity for the naive algorithm ($\texttt{Naive}(\mathcal{A}, x, y)$) or e for Hopcroft & Karp's algorithm ($\texttt{HK}(\mathcal{A}, x, y)$).

A preliminary algorithm for checking language equivalence of two states $x, y \in S$ is obtained as follows: try to complete the relation $\{\langle x, y\rangle\}$ into a bisimulation, by adding the successors along all letters and checking that o agrees on all inserted pairs. This algorithm is described in Fig. 1; it is quadratic in worst case since a pair of states is added to the relation R at each iteration. The standard and almost linear algorithm by Hopcroft and Karp [17,28], can be seen as an improvement of this naive algorithm where one searches for bisimulations up to equivalence rather than plain bisimulations:

Definition 2. *Let $e\colon Rel(S) \to Rel(S)$ be the function mapping a relation R to the least equivalence relation containing R. A* bisimulation up to equivalence *is a relation R such that $R \subseteq g(e(R))$.*

This coarser notion makes it possible to take advantage of the fact that language equivalence is indeed an equivalence relation, so that one can skip pairs of states whose equivalence follows by transitivity from the previously visited pairs. The soundness of this technique is established by the following Proposition:

Proposition 1 ([4, Thm. 1]). *If R is a bisimulation up to equivalence, then $e(R)$ is a bisimulation.*

Complexity-wise, when looking for bisimulations up to equivalence in a DFA with n states, at most n pairs can be inserted in R in the algorithm in Fig. 1: at the beginning, $e(R)$ corresponds to a discrete partition with n equivalence classes; at each iteration, two classes of $e(R)$ are merged.

Note that Hopcroft and Karp's algorithm proceeds forward and computes the smallest bisimulation up to equivalence containing the starting pair of states, if any. As mentioned above, this contrasts with partition-refinement algorithms [16], which proceed backward: they start with a coarse partition (accepting v.s. non-accepting states), which they refine by reading transitions backward.

2.2 Non-deterministic automata: HKC

A *non-deterministic finite automaton* (NFA) over the alphabet A and with outputs in O is a triple $\langle S, o, t\rangle$, where S is a finite set of states, $o\colon S \to O$ is the output function, and $t\colon A \times S \to \mathcal{P}(S)$ is the transition function which returns, for each letter $a \in A$ and for each state x, a set $t_a(x)$ of potential successors. Like for DFA, we do not specify a set of initial states in the definition of NFA.

We fix an NFA $\langle S, o, t\rangle$ in this section and we assume that the set O of outputs is a semilattice. Under this assumption, an NFA $\mathcal{A} = \langle S, o, t\rangle$ can be transformed into a DFA $\mathcal{A}^{\#} = \langle \mathcal{P}(S), o^{\#}, t^{\#}\rangle$ using the well-known powerset construction:

$$o^{\#}(X) = \sum_{x \in X} o(x) \qquad\qquad t_a^{\#}(X) = \bigcup_{x \in X} t_a(x)$$

This construction makes it possible to extend the function $[\cdot]$ into a function from sets of states of a given NFA to weighted languages. It also gives immediately algorithms to decide language equivalence in NFA: just use algorithms for DFA on the resulting automaton. Note that when doing so, it is not always necessary to compute the determinised automaton beforehand. For instance, with coinductive algorithms like in Fig. 1, the determinised automaton can be explored on the fly. This is useful since this DFA can have exponentially many states, even when restricting to reachable subsets.

The key idea behind the HKC algorithm [4] is that one can actually do better than Hopcroft and Karp's algorithm by exploiting the semilattice structure of the state-space of NFA determinised through the powerset construction. This is done using *bisimulations up to congruence*.

Definition 3. *Let* $c\colon \mathit{Rel}(\mathcal{P}(S)) \to \mathit{Rel}(\mathcal{P}(S))$ *be the function mapping a relation* R *to the least equivalence relation* $\mathcal{H}$ *containing* R *and such that* $X\ \mathcal{H}\ Y$ *and* $X'\ \mathcal{H}\ Y'$ *entail* $(X \cup X')\ \mathcal{H}\ (Y \cup Y')$ *for all* $X, X', Y, Y' \in \mathcal{P}(S)$*. A* bisimulation up to congruence *is a relation* R *such that* $R \subseteq g(c(R))$*.*

The function g here is defined as in Sect. 2.1, but with respect to the determinized DFA with state space $\mathcal{P}(S)$, so its type is $\mathit{Rel}(\mathcal{P}(S)) \to \mathit{Rel}(\mathcal{P}(S))$.

Proposition 2 ([4, Thm. 2]). *If* R *is a bisimulation up to congruence, then* $c(R)$ *is a bisimulation.*

Checking whether a pair of sets belongs to the congruence closure of a finite relation can be done algorithmically (see [4, Sect.3.4]). The algorithm HKC [4] is obtained by running the algorithm from Fig. 1 on $\mathcal{A}^{\#}$, replacing the function f on l.5 with the congruence closure function c. We provide a variant of this algorithm in Fig. 2, where we prepare the ground for the algorithms we will propose for Büchi automata. There, we only explore the transitions of the determinised automaton, leaving aside the verification that the output function agrees on all pairs. This corresponds to using a function g' instead of g, defined as

$$g'(R) = \left\{\langle x, y\rangle \mid \forall a \in A,\ t_a^{\#}(x)\ R\ t_a^{\#}(y)\right\}$$

input : A NFA $\mathcal{A} = \langle S, o, t\rangle$ and two sets of states $X, Y \subseteq S$
output : a relation R such that $[X] = [Y]$ iff $\forall \langle X', Y'\rangle \in R,\ o^{\#}(X') = o^{\#}(Y')$

```
1 R := ∅; todo := {⟨X, Y⟩};
2 while todo ≠ ∅ do
  |  // invariant: ⟨X, Y⟩ ∈ R ⊆ g'(c(R ∪ todo))
3 |  extract ⟨X', Y'⟩ from todo;
4 |  if ⟨X', Y'⟩ ∈ c(R ∪ todo) then skip;
5 |  forall a ∈ A do
6 |  |  insert ⟨t_a^#(X'), t_a^#(Y')⟩ in todo;
7 |  insert ⟨X', Y'⟩ in R;
8 return R;
```

Fig. 2. HKC'$(\mathcal{A}, X, Y)$: computing a pre-bisimulation up to congruence in a NFA.

Indeed, while this verification step is usually done on the fly in order to fail faster when a counter-example is found (as in Fig. 1, line 4), it will be useful later to perform this step separately.

As mentioned in the Introduction, the advantage of HKC over HK is that in practice it often makes it possible to skip reachable subsets from the determinised automaton, even when the algorithm answers positively, thus achieving substantial gains in terms of performance: there are families of examples where it answers positively in polynomial time even though the underlying minimal DFA has exponential size. Actually it can also improve exponentially over the more recent antichain-based algorithms [4, Sect. 4]. These latter gains can be explained by the fact that we focus on language equivalence rather than language inclusion: while the two problems are interreducible (e.g., $[X] \subseteq [Y]$ iff $[X \cup Y] = [Y]$), working with equivalence relations makes it possible to strengthen the coinductive argument used implicitly by both algorithms.

3 From Büchi automata to finite words automata

Let $\mathbb{3}$ be the set $\{0, 1, \star\}$. A *(non-deterministic) Büchi automaton* (NBW) over the alphabet A is a tuple $\langle S, T\rangle$ where S is a finite set of states, and $T\colon A \to \mathbb{3}^{S^2}$ is the transition function. Like for DFA and NFA, we do not include a set of initial states in the definition. We work with Büchi automata with Büchi transitions rather than Büchi states, hence the type of T (the two models are equivalent and the one we chose is slightly more succinct). We write T_a for $T(a)$, $x \xrightarrow{a} x'$ when $T_a(x, x') \neq 0$, and $x \overset{a}{\Rightarrow} x'$ when $T_a(x, x') = \star$; the latter denote Büchi transitions, that should be fired infinitely often in order to accept an infinite word.

Given a NBW $\mathcal{A} = \langle S, T\rangle$ and $w \in A^{\omega}$ an infinite word, we say that a sequence of states $\chi \in S^{\omega}$ accepts w if the sequence $(T_{w_i}(\chi_i, \chi_{i+1}))_{i\in\mathbb{N}}$ contains infinitely many $\star$ and no 0. The ω-language $[X]_{\mathcal{A}}$ of a set of states $X \subseteq S$ is the set of infinite words accepted by a sequence χ such that $\chi_0 \in X$. The ω-languages accepted by some set of states in a NBW are the *rational ω-languages* [5].

Given a finite word $u \in A^*$ and a finite non-empty word $v \in A^+$, write uv^ω for the infinite word $w \in A^\omega$ defined by $w_i = u_i$ if $i < |u|$ and $w_i = v_{(i-|u|) mod |v|}$ otherwise. *Ultimately periodic words* are (infinite) words of the form uv^ω for some $u, v \in A^* \times A^+$. Given an ω-language $L \subseteq A^\omega$, we set

$$UP(L) = \{uv^\omega \mid uv^\omega \in L\} \qquad L^\$ = \{u\$v \mid uv^\omega \in L\}$$

$UP(L)$ is a ω-language over A while $L^\$$ is a language of finite words over the alphabet $A^\$ = A \uplus \{\$\}$. The first key observation is that the ultimately periodic words of a rational ω-language fully characterise it:

Proposition 3 ([6, Fact 1]). *For all rational ω-languages L, L', we have that $UP(L) = UP(L')$ entails $L = L'$.*

Proof. Consequence of the closure of rational ω-languages under Boolean operations [5], and the fact that every non-empty rational ω-language contains at least one ultimately periodic word. □

As a consequence, to compare the ω-languages of two sets of states in a NBW, it suffices to compare the ω-languages of ultimately periodic words they accept. Calbrix et al. show that these ω-languages can be faithfully represented as rational languages (of finite words):

Proposition 4 ([6, Prop. 4]). *If $L \subseteq A^\omega$ is ω-regular, then $L^\$$ is regular.*

To prove it, Calbrix et al. construct a NFA for $L^\$$ from a NBW $\mathcal{A}$ for L, with two layers. The first layer recognises the prefixes (the u in uv^ω). This is a copy of the NBW for L (without accepting states, and where the Büchi status of the transitions is ignored). This layer guesses non-deterministically when to read the \$ symbol and then jumps into the second layer, whose role is to recognise the period (the v in uv^ω). We depart from [6] here, by using notions from [25] which make the presentation easier and eventually make it possible to propose our algorithm. We use the *(Büchi) transition monoid* of the NBW $\mathcal{A} = \langle S, T \rangle$ [25] to define the second layer.

Consider the set $\mathbb{3}$ as an idempotent semiring, using the following operations:

$+$	0	1	$\star$
0	0	1	$\star$
1	1	1	$\star$
$\star$	$\star$	$\star$	$\star$

$\cdot$	0	1	$\star$
0	0	0	0
1	0	1	$\star$
$\star$	0	$\star$	$\star$

Write $\mathscr{M} = \mathbb{3}^{S^2}$ for the set of square matrices over $\mathbb{3}$ indexed by S; it forms a Kleene algebra [7,20] and in particular a semiring. Let I denote the identity matrix of $\mathscr{M}$. The transition function of $\mathcal{A}$ has type $A \to \mathscr{M}$; we extend it to finite words by setting $T_\epsilon = I$ and $T_{u_1 \dots u_n} = T_{u_1} \cdot \dots \cdot T_{u_n}$. We have that $T_u(x, x')$ is $\star$ if there is a path along u from x to x' visiting an accepting transition, 0 if there is no path from x to x' along u, and 1 otherwise. We extend the notations $x \xrightarrow{u} x'$ and $x \overset{u}{\Rightarrow} x'$ to words accordingly.

A periodic word v^ω is accepted from a state x in $\mathcal{A}$ if and only if there is a *lasso* for v starting from x: a state y and two natural numbers n, m such that $x \xrightarrow{v^n} y \xRightarrow{v^m} y$. This information can be computed from the matrix T_v: given a matrix M, compute[1] its Kleene star M^* and set

$$\omega(M) = \{x \in S \mid \exists y \in S,\ M^*(x,y) \neq 0 \wedge M^*(y,y) = \star\} \quad . \tag{$\dagger$}$$

At this point, one can notice that with the previously defined operations, matrices and subsets form the *Wilke algebra* associated to the NBW as in [25].

Lemma 1. *For all words v, v^ω is accepted from a state x iff $x \in \omega(T_v)$.*

We can now formally define the desired NFA: set $\mathcal{A}^\$ = \langle S^\$, o^\$, T^\$\rangle$, where $S^\$ = S \uplus S{\times}\mathscr{M}$ is the disjoint union of S and $|S|$ copies of $\mathscr{M}$, and

$$\begin{cases} T_a^\$(x) = \{x' \mid T_a(x,x') \neq 0\} \\ T_a^\$(\langle x, M\rangle) = \{\langle x, M \cdot T_a\rangle\} \end{cases} \qquad \begin{cases} T_\$^\$(x) = \{\langle x, I\rangle\} \\ T_\$^\$(\langle x, M\rangle) = \emptyset \end{cases} \qquad \begin{cases} o^\$(x) = 0 \\ o^\$(\langle x, M\rangle) = x \in \omega(M) \end{cases}$$

The set $\mathscr{M}$ can be replaced here by its accessible part $\mathscr{M}' = \{T_u \mid u \in A^*\}$. The main difference with the construction from [6] is that we use deterministic automata in the second layer, which enable a streamlined presentation in terms of matrices—which are not mentioned explicitly in [6]. The construction of $\mathcal{A}^\$$ preserves the semantics of all sets of states, up to $L \mapsto L^\$$:

Theorem 2. *For all sets X of states from $\mathcal{A}$, we have $[X]_{\mathcal{A}^\$} = ([X]_{\mathcal{A}})^\$$.*

Example 1. To illustrate this construction, consider the NBW depicted on the left in Fig. 3. The state 0 accepts the words with a finite but non-zero number of b's; the state 1 only accepts the word a^ω. Accordingly, we have $[0]^\$_{\mathcal{A}} = (a+b)^* b a^* \$ a^+$ and $[1]^\$_{\mathcal{A}} = a^* \$ a^+$. These are indeed the languages respectively recognised by the states 0 and 1 from the NFA $\mathcal{A}^\$$ on the right.

We only depicted the relevant part of the second layer: the only reachable matrices are those of the form T_u for some word u. There are only three of them in this example since $T_a \cdot T_b = T_b \cdot T_a = T_b \cdot T_b = T_b$ and $T_a \cdot T_a = T_a$. We might want to prune $\mathcal{A}^\$$ so that all states may reach an accepting state, but we want in the sequel to exploit the structure shared by the copies of the transition monoid: they only differ by the accepting status of their states, by definition.

Note that since the second layer of $\mathcal{A}^\$$ is already deterministic, one can determinise $\mathcal{A}^\$$ into a DFA with at most $2^n + 2^n 3^{n^2}$ states, where n is the number of states of $\mathcal{A}$. This is slightly better than the $2^n + 2^{2n^2+n}$ bound obtained in [6].

We summarise the operations defined so far on languages and automata in Fig. 4; we define the operations in the right-most column in the following section.

[1] To compute M^*, one can use the fact that $M^* = (I+M)^n$ with $n = |S|$, and use iterated squaring.

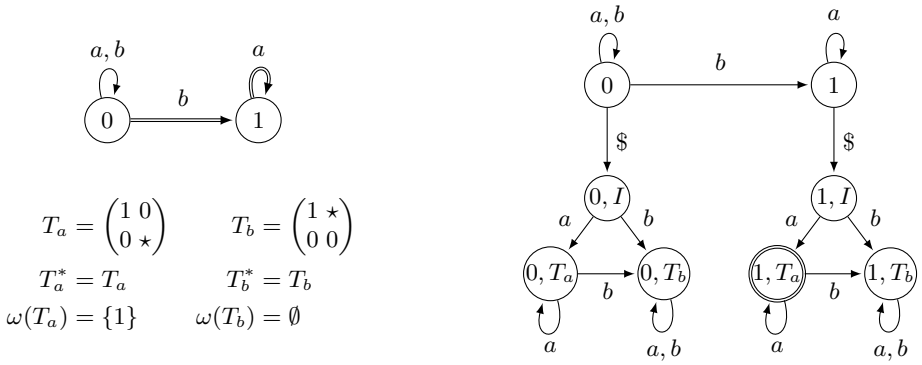


Fig. 3. A NBW $\mathcal{A}$ (left) and the reachable part of its associated NFA $\mathcal{A}^{\$}$ (right).

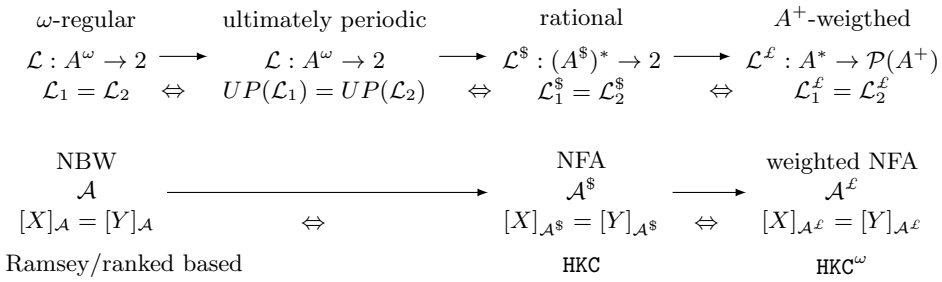


Fig. 4. Summary of the operations and algorithms on languages and automata.

4 HKC for Büchi automata

By Prop. 3 and Thm. 2, given two sets of states X, Y of a NBW $\mathcal{A}$, we have $[X]_{\mathcal{A}} = [Y]_{\mathcal{A}}$ iff $[X]_{\mathcal{A}^{\$}} = [Y]_{\mathcal{A}^{\$}}$. One can thus use any algorithm for language equivalence on NFA to solve language equivalence on NBW. Given the structure (and size) of $\mathcal{A}^{\$}$, this would however be inefficient: each time the letter \$ is read, the algorithm would explore one of the automata for the second layer, without ever realising that the transition structure of those automata is always the same, only the accepting status of their states differ. We show in this section that we can do better, by using a weighted automata.

Given a an ω-language L, the language $L^{\$}$ can be seen as a weighted language $L^{£} \colon A^* \to \mathcal{P}(A^+)$ with weights in the semilattice $\langle \mathcal{P}(A^+), \cup, \emptyset \rangle$:

$$L^{£} : u \mapsto \left\{ v \in A^+ \mid uv^\omega \in L \right\}$$

Given a NBW $\mathcal{A} = \langle S, T \rangle$, one can immediately construct a NFA $\mathcal{A}^{£} = \langle S^{£}, T^{£}, o^{£} \rangle$ such that for every set of states X, $[X]_{\mathcal{A}}^{£} = [X]_{\mathcal{A}^{£}}$. This is just the first layer from the previous construction: set $S^{£} = S$ and

$$T_a^{£}(x) = \{x' \mid T_a(x, x') \neq 0\} \qquad o^{£}(x) = \left\{ v \in A^+ \mid v^\omega \in [x]_{\mathcal{A}} \right\}$$

input : A NBW $\mathcal{A} = \langle S, T \rangle$
output : The set of discriminating sets $\mathcal{D} = \{\omega(T_v) \mid v \in A^*\}$

```
1 D := ∅; M := ∅; todo := {I};
2 while todo ≠ ∅ do
3 |   extract M from todo;
4 |   if M ∈ M then skip;
5 |   forall a ∈ A do
6 |   |   insert M · T_a in todo;
7 |   insert M in M; insert ω(M) in D;
8 return D;
```

Fig. 5. `Discr`($\mathcal{A}$): exploring the Büchi transition monoid of a NBW $\mathcal{A}$ to compute discriminating sets.

Let $\mathcal{A}^{\pounds\#}$ be the powerset automaton of $\mathcal{A}^{\pounds}$. To use algorithms such as HKC on $\mathcal{A}^{\pounds}$, it suffices to be able to compare the outputs of any two states of $\mathcal{A}^{\pounds\#}$, i.e., compare the languages $o^{\pounds\#}(X)$ and $o^{\pounds\#}(Y)$ for any two sets $X, Y \subseteq S$. Since those languages are rational (using the second layer of the previous construction), it might be tempting to use algorithms such as `HK` or `HKC` to perform this task. We proceed differently in order to exploit the shared structure of those languages.

Lemma 2. *For all states $x \in S$ and sets $X \subseteq S$, we have*

$$o^{\pounds}(x) = \left\{v \in A^+ \mid x \in \omega(T_v)\right\}$$
$$o^{\pounds\#}(X) = \left\{v \in A^+ \mid X \cap \omega(T_v) \neq \emptyset\right\}$$

Proof. Immediate consequence of Lem. 1 and the definitions of $o^{\pounds}$ and $o^{\pounds\#}$. Note that allowing empty v would not change the statement since $\omega(T_\epsilon) = \omega(I) = \emptyset$.

Proposition 5. *For all sets $X, Y \subseteq S$,*

$$o^{\pounds\#}(X) = o^{\pounds\#}(Y) \quad \textit{iff} \quad \textit{for all } v \in A^+,\ X \cap \omega(T_v) = \emptyset \Leftrightarrow Y \cap \omega(T_v) = \emptyset.$$

This result shows that an explicit computation of $o^{\pounds\#}$ is not necessary, as the knowledge of $\{\omega(T_v), v \in A^+\}$ is enough to assess whether X and Y have same output. Let $\mathcal{D} = \{\omega(T_v) \mid v \in A^+\}$. We call the sets in $\mathcal{D}$ *discriminating sets*. Again, allowing empty v here would make no difference: the discrimating set $\emptyset$ is useless to distinguish two sets $X, Y \subseteq S$. As subsets of S, there are at most $2^{|S|}$ discriminating sets. Those can be enumerated since the T_v range over finitely many matrices (at most $3^{|S|^2}$). This is what is done in the algorithm from Fig. 5.

We finally obtain the algorithm in Fig. 6 for language equivalence in a NBW: we compute the discriminating sets ($\mathcal{D}$) and a relation (R) which is almost a bisimulation up to congruence: the outputs of its pairs must be checked against the discriminating sets, which we achieve with a simple loop (lines 2-4).

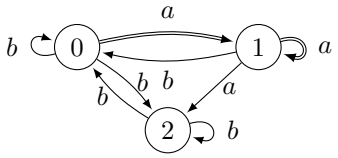
Example 2. We execute `HKC`$^\omega$ on the NBW on the left below, starting with states $\{0\}$ and $\{1\}$. The transition monoid has 13 elements, which we list in App. A.

input : A NBW $\mathcal{A} = \langle S, T\rangle$ and two sets $X, Y \subseteq S$
output : true if $[X]_{\mathcal{A}} = [Y]_{\mathcal{A}}$; false otherwise

1 $R := \texttt{HKC}'(\mathcal{A}^{\pounds}, X, Y)$ || $\mathcal{D} := \texttt{Discr}(\mathcal{A})$;
2 **forall** $\langle X', Y'\rangle \in R,\ D \in \mathcal{D}$ **do**
3 | **if** $X' \cap D = \emptyset \not\Leftrightarrow Y' \cap D = \emptyset$ **then return** *false*;
4 **return** *true*;

Fig. 6. $\texttt{HKC}^{\omega}(\mathcal{A}, X, Y)$: checking language equivalence in a NBW using bisimulations up to congruence.

They give rise to three discriminating sets: $\emptyset$, $\{0,1\}$, and $\{0,1,2\}$, which arise for instance from the three matrices on the right, using formula (†) on page 8:



$$T_b = \begin{pmatrix} 1\,0\,1 \\ 1\,0\,0 \\ 1\,0\,1 \end{pmatrix} \quad T_a = \begin{pmatrix} 0 \star 0 \\ 0 \star 1 \\ 0\,0\,0 \end{pmatrix} \quad T_{ba} = \begin{pmatrix} 0 \star 0 \\ 0 \star 0 \\ 0 \star 0 \end{pmatrix}$$

HKC' returns the relation $R = \{\langle\{0\},\{1\}\rangle, \langle\{1\},\{1,2\}\rangle\}$, which contains only two pairs. The pairs $\langle\{0,2\},\{0\}\rangle$, $\langle\{1,2\},\{1,2\}\rangle$, and $\langle\{0\},\{0,2\}\rangle$, which are reachable from $\langle\{0\},\{1\}\rangle$ by reading the words b, aa, and ab, are skipped thanks to the up to congruence technique. For instance to obtain the pair $\langle\{0,2\},\{0\}\rangle$, starting from $\langle\{0\},\{1\}\rangle$ and $\langle\{1\},\{1,2\}\rangle$ we can obtain $\langle\{0\},\{1,2\}\rangle$ by transitivity, from which we deduce $\langle\{0,2\},\{1,2\}\rangle$ by union with $\langle\{2\},\{2\}\rangle$. By transitivity and symmetry we can finally obtain $\langle\{0,2\},\{0\}\rangle$.

The two pairs of R cannot be told apart using the three discriminating sets and $\texttt{HKC}^{\omega}$ returns *true*. States 0 and 1 are indeed equivalent: they accept the words with infinitely many a's. If instead we start $\texttt{HKC}^{\omega}$ from sets $\{0\}$ and $\{2\}$, it returns *false*: the discriminating set $\{0,1\}$ distinguishes $\{0\}$ and $\{2\}$. Indeed, the state 2 recognises the words starting with b and with infinitely many a's.

Note that $\texttt{HKC}^{\omega}$ can be instrumented to return a counterexample in case of failure: it suffices to record the finite word u that lead to each pair in R as well the finite word v that lead to each discriminating set in $\mathcal{D}$: if the check on line 3 fails, the corresponding word uv^{ω} is a counter-example to language equivalence.

Also note that $\texttt{HKC}^{\omega}$ is intrinsically parallel: the computations of $\mathcal{D}$ and R can be done in parallel, and the checks in lines 2-4 can be performed using a producer-consumer pattern where they are triggered whenever new values are inserted in $\mathcal{D}$ or R. Alternatively, those checks can be delegated to a SAT solver. Indeed, given a discriminating set D, define the following formula with $2|D|$ variables $\{x_d \mid d \in D\} \cup \{y_d \mid d \in D\}$:

$$\varphi_D = \bigvee_{d\in D} x_d \ \Leftrightarrow \ \bigvee_{d\in D} y_d$$

For all sets $X, Y \subseteq S$, we have $X \cap D = \emptyset \Leftrightarrow Y \cap D = \emptyset$ iff φ_D evaluates to *true* under the assignment $x_d \mapsto d \in X$ and $y_d \mapsto d \in Y$. Given the set of

discriminating sets $\mathcal{D}$, it thus suffices to build the formula $\varphi_{\mathcal{D}} = \bigwedge_{D \in \mathcal{D}} \varphi_D$ with $2|S|$ variables, and to evaluate it on all pairs from the relation R returned by HKC'. The main advantage of proceeding this way is that the SAT solver might be able to represent $\varphi_{\mathcal{D}}$ in a compact and efficient way. If we moreover use an incremental SAT solver, this formula can be built incrementally, thus avoiding the need to store explicitly the set $\mathcal{D}$.

One can also use a (incremental) SAT solver in a symmetrical way: Given a pair of sets $\langle X, Y \rangle \in S^2$, define the following formula with $|S|$ variables $\{x_s \mid s \in S\}$:

$$\psi_{\langle X,Y \rangle} = \bigvee_{s \in X} x_s \;\Leftrightarrow\; \bigvee_{s \in Y} x_s$$

For all sets D, we have $X \cap D = \emptyset \Leftrightarrow Y \cap D = \emptyset$ iff $\psi_{\langle X,Y \rangle}$ evaluates to *true* under the assignment $x_s \mapsto s \in D$. Like previously, one can thus construct incrementally the formula $\psi_R = \bigwedge_{p \in R} \psi_p$ before evaluating it on all discriminating sets.

5 Conclusion and future work

We presented an algorithm for checking language equivalence of non-deterministic Büchi automata. This algorithm exploits advanced coinductive techniques to analyse the finite prefixes of the considered languages, through bisimulations up to congruence, as in the algorithm HKC for NFA. The periodic part of the considered languages is also analysed coinductively, in order to compute the discriminating sets. Those sets make it possible to classify the periodic words accepted by the various states of the starting automaton, thus providing all the necessary information together with the analysis of the finite prefixes.

A prototype implementation is available; it makes it possible to test several combinations of up-to techniques [22].

Our algorithm stems from the construction of Calbrix et al. [6], which we revisited using notions from [25] in Sect. 3. HKC$^\omega$ is rather close to *Ramsey-based* algorithms [11,1] (as opposed to *rank-based* ones [23,8,9,10]). In particular, our matrices are often called *super-graphs* in Ramsey-based algorithms. A key difference is that we focus on language equivalence, thus enabling stronger coinductive proof principles.

The next step is to design up-to techniques in order to reduce the exploration of the periodic layer, to compute the discriminating sets more efficiently. We provide two such techniques in the extended version of this abstract [21], namely coinduction up to unions and coinduction up to equivalence. Using the two techniques at the same time is likely to be possible, i.e., using coinduction up to congruence; this however requires further investigations, especially in order to find reasonably efficient ways to perform the corresponding tests.

Along the same vein, we also want to investigate how to exploit techniques using simulation relations, which were successfully used in [10,1,2,24] and which tend to nicely fit in the coinductive framework we propose here [4, Sect. 5].

Acknowledgements. We would like to thank Dmitriy Traytel for pointing us to the work of Calbrix et al. [6].

References

1. P. A. Abdulla, Y. Chen, L. Clemente, L. Holík, C. Hong, R. Mayr, and T. Vojnar. Simulation subsumption in ramsey-based Büchi automata universality and inclusion testing. In *Proc. CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 132–147. Springer, 2010.
2. P. A. Abdulla, Y. Chen, L. Clemente, L. Holík, C. Hong, R. Mayr, and T. Vojnar. Advanced ramsey-based Büchi automata inclusion testing. In *Proc. CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 187–202. Springer, 2011.
3. P. A. Abdulla, Y.-F. Chen, L. Holík, R. Mayr, and T. Vojnar. When simulation meets antichains. In *Proc. TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 158–174. Springer, 2010.
4. F. Bonchi and D. Pous. Checking NFA equivalence with bisimulations up to congruence. In *Proc. POPL*, pages 457–468. ACM, 2013.
5. J. R. Büchi. On a decision method in restricted second order arithmetic. In S. Mac Lane and D. Siefkes, editors, *The Collected Works of J. Richard Büchi*, pages 425–435. Springer New York, New York, NY, 1990.
6. H. Calbrix, M. Nivat, and A. Podelski. Ultimately periodic words of rational w-languages. In *Proc. MFPS*, volume 802 of *Lecture Notes in Computer Science*, pages 554–566. Springer, 1993.
7. J. H. Conway. *Regular algebra and finite machines*. Chapman and Hall, 1971.
8. L. Doyen and J. Raskin. Improved algorithms for the automata-based approach to model-checking. In *Proc. TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 451–465. Springer, 2007.
9. L. Doyen and J. Raskin. Antichains for the automata-based approach to model-checking. *Logical Methods in Computer Science*, 5(1), 2009.
10. L. Doyen and J.-F. Raskin. Antichain Algorithms for Finite Automata. In *Proc. TACAS*, volume 6015 of *Lecture Notes in Computer Science*. Springer, 2010.
11. S. Fogarty and M. Y. Vardi. Büchi complementation and size-change termination. In *Proc. TACAS*, volume 5505 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2009.
12. S. Fogarty and M. Y. Vardi. Efficient Büchi universality checking. In *Proc. TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 205–220. Springer, 2010.
13. P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *Proc. CAV*, pages 53–65. Springer, 2001.
14. S. Gurumurthy, O. Kupferman, F. Somenzi, and M. Y. Vardi. On complementing nondeterministic Büchi automata. In *Proc. Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 96–110. Springer, 2003.
15. G. J. Holzmann. The model checker spin. *IEEE Transactions on software engineering*, 23(5):279–295, 1997.
16. J. E. Hopcroft. An n log n algorithm for minimizing in a finite automaton. In *Proc. International Symposium of Theory of Machines and Computations*, pages 189–196. Academic Press, NY, USA, 1971.
17. J. E. Hopcroft and R. M. Karp. A linear algorithm for testing equivalence of finite automata. Technical Report 114, Cornell Univ., December 1971.
18. M. Hutagalung, M. Lange, and E. Lozes. Revealing vs. concealing: More simulation games for Büchi inclusion. In *Proc. LATA*, pages 347–358. Springer, 2013.
19. B. Knaster. Un théorème sur les fonctions d'ensembles. *Annales de la Société Polonaise de Mathématiques*, 6:133–134, 1928.

20. D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994.
21. D. Kuperberg, L. Pinault, and D. Pous. Extended version of this abstract, 2018.
22. D. Kuperberg, L. Pinault, and D. Pous. Web appendix for this paper, 2019.
23. O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3):408–429, 2001.
24. R. Mayr and L. Clemente. Advanced automata minimization. In *Proc. POPL, 2013*, pages 63–74. ACM, 2013.
25. D. Perrin and J.-É. Pin. Semigroups and automata on infinite words. *NATO ASI Series C Mathematical and Physical Sciences-Advanced Study Institute*, 466:49–72, 1995.
26. S. Safra. On the complexity of omega-automata. In *Proc. FoCS*, pages 319–327. IEEE, 1988.
27. S. Schewe. Beyond hyper-minimisation—minimising dbas and dpas is np-complete. In *Proc. FSTTCS*, pages 400–411, 2010.
28. R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.
29. A. Tarski. A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific Journal of Mathematics*, 5(2):285–309, June 1955.
30. M.-H. Tsai, Y.-K. Tsay, and Y.-S. Hwang. Goal for games, omega-automata, and logics. In N. Sharygina and H. Veith, editors, *CAV*, pages 883–889, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
31. M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for concurrency*, pages 238–266. Springer, 1996.
32. M. D. Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *Proc. CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 17–30. Springer, 2006.

A Further details for Example 2

The transition monoid of the NBW in example 2 is given below, together with the discriminating sets associated to its elements:

u	T_u	T_u^*	$\omega(T_u)$
ϵ	$\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}$	$\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}$	$\varnothing$
a	$\begin{pmatrix}0&\star&0\\0&\star&1\\0&0&0\end{pmatrix}$	$\begin{pmatrix}1&\star&\star\\0&\star&\star\\0&0&1\end{pmatrix}$	$\{0,1\}$
b	$\begin{pmatrix}1&0&1\\1&0&0\\1&0&1\end{pmatrix}$	$\begin{pmatrix}1&0&1\\1&1&1\\1&0&1\end{pmatrix}$	$\varnothing$
aa	$\begin{pmatrix}0&\star&\star\\0&\star&\star\\0&0&0\end{pmatrix}$	$\begin{pmatrix}1&\star&\star\\0&\star&\star\\0&0&1\end{pmatrix}$	$\{0,1\}$
ab	$\begin{pmatrix}\star&0&0\\\star&0&1\\0&0&0\end{pmatrix}$	$\begin{pmatrix}\star&0&0\\\star&1&1\\0&0&1\end{pmatrix}$	$\{0,1\}$
ba	$\begin{pmatrix}0&\star&0\\0&\star&0\\0&\star&0\end{pmatrix}$	$\begin{pmatrix}1&\star&0\\0&\star&0\\0&\star&1\end{pmatrix}$	$\{0,1,2\}$
bb	$\begin{pmatrix}1&0&1\\1&0&1\\1&0&1\end{pmatrix}$	$\begin{pmatrix}1&0&1\\1&1&1\\1&0&1\end{pmatrix}$	$\varnothing$
aab	$\begin{pmatrix}\star&0&\star\\\star&0&\star\\0&0&0\end{pmatrix}$	$\begin{pmatrix}\star&0&\star\\\star&1&\star\\0&0&1\end{pmatrix}$	$\{0,1\}$
aba	$\begin{pmatrix}0&\star&0\\0&\star&0\\0&0&0\end{pmatrix}$	$\begin{pmatrix}1&\star&0\\0&\star&0\\0&0&1\end{pmatrix}$	$\{0,1\}$
baa	$\begin{pmatrix}0&\star&\star\\0&\star&\star\\0&\star&\star\end{pmatrix}$	$\begin{pmatrix}1&\star&\star\\0&\star&\star\\0&\star&\star\end{pmatrix}$	$\{0,1,2\}$
bab	$\begin{pmatrix}\star&0&0\\\star&0&0\\\star&0&0\end{pmatrix}$	$\begin{pmatrix}\star&0&0\\\star&1&0\\\star&0&1\end{pmatrix}$	$\{0,1,2\}$
$abab$	$\begin{pmatrix}\star&0&0\\\star&0&0\\0&0&0\end{pmatrix}$	$\begin{pmatrix}\star&0&0\\\star&1&0\\0&0&1\end{pmatrix}$	$\{0,1\}$
$baab$	$\begin{pmatrix}\star&0&\star\\\star&0&\star\\\star&0&\star\end{pmatrix}$	$\begin{pmatrix}\star&0&\star\\\star&1&\star\\\star&0&\star\end{pmatrix}$	$\{0,1,2\}$

Equations

$T_{aaa} = T_{aa}$
$T_{abb} = T_{aab}$
$T_{bba} = T_{ba}$
$T_{bbb} = T_{bb}$
$T_{aaba} = T_{aba}$
$T_{aabb} = T_{aab}$
$T_{abaa} = T_{aa}$
$T_{baaa} = T_{baa}$
$T_{baba} = T_{ba}$
$T_{babb} = T_{baab}$