Coinductive algorithms for Büchi automata*

Denis Kuperberg, Laureline Pinault, Damien Pous

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

{*denis.kuperberg,laureline.pinault,damien.pous*}*@ens-lyon.fr*

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) [29, 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, 28, 23].

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 [30, 3, 10]

*This is the author version of the paper with the same title published by IOS Press in Fundamenta Informaticae, vol. 180, no. 4, pp. 351-373, 2021, available at https://doi.org/10.3233/FI-2021-2046. 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).

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 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.

Let us insist on the fact that the goal of the present paper is not to demonstrate the practical superiority of this approach, but merely to present an algorithm, together with some ideas for tweaking it, that might prove useful in the future exploration of efficient algorithms.

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. to make their use of the Büchi transition monoid [24] explicit (Sect. 3). We define the new algorithm $\texttt{HKC}^\omega$ in Sect. 4. We discuss more advanced refinements of the algorithm in Sect. 5. We conclude with directions for future work in Sect. 6.

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: they recognise *weighted languages*. 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 2.1. (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 (i.e., it preserves the inclusion ordering), it admits the union of all bisimulations as a greatest fixpoint, by Knaster-Tarski's theorem [19, 27]. This greatest-fixpoint is actually language equivalence:

Theorem 2.2. For all $x, y \in S$, $[x] = [y]$ if and only if 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

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 $R := \emptyset$; $todo := \{\langle x, y\rangle\}$;
2 **while** $todo \neq \emptyset$ **do**
  // `invariant:` $\langle x, y\rangle \in R \cup todo \wedge R \subseteq g(f(R \cup todo))$
3 extract $\langle x', y'\rangle$ from $todo$;
4 **if** $o(x') \neq o(y')$ **then return** *false*;
5 **if** $\langle x', y'\rangle \in f(R \cup todo)$ **then** skip; `// back to the beginning of the loop`
6 **forall** $a \in A$ **do**
7 insert $\langle t_a(x'), t_a(y')\rangle$ in $todo$;
8 insert $\langle x', y'\rangle$ in R;
9 **return** *true*; `// because:` $\langle x, y\rangle \in R \subseteq g(f(R))$

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

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 [26].

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, 26], can be seen as an improvement of this naive algorithm where one searches for bisimulations up to equivalence rather than plain bisimulations:

Definition 2.3. 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 2.4. ([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 whole 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.

Formally, when doing so, the function g is defined as in Sect. 2.1, but with respect to the determinised DFA with state space $\mathcal{P}(S)$, so its type is $Rel(\mathcal{P}(S)) \to Rel(\mathcal{P}(S))$:

$$g(R) = \left\{\langle X, Y\rangle \mid o^\#(X) = o^\#(Y) \text{ and } \forall a \in A,\ t^\#_a(X)\ R\ t^\#_a(Y)\right\}$$

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 automata determinised through the powerset construction. This is done using *bisimulations up to congruence*.

Definition 2.5. Let $c\colon Rel(\mathcal{P}(S)) \to Rel(\mathcal{P}(S))$ be the function mapping a relation R to the least equivalence relation R' containing R and such that $X\ R'\ Y$ and $X'\ R'\ Y'$ entail $(X \cup X')\ R'\ (Y \cup Y')$ for all $X, X', Y, Y' \in \mathcal{P}(S)$. A *bisimulation up to congruence* is a relation R such that $R \subseteq g(c(R))$.

Proposition 2.6. ([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 line 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

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 $R := \emptyset$; $todo := \{\langle X, Y\rangle\}$;
2 **while** $todo \neq \emptyset$ **do**
  `// invariant:` $\langle X, Y\rangle \in R \cup todo \wedge R \subseteq g'(c(R \cup todo))$
3 extract $\langle X', Y'\rangle$ from $todo$;
4 **if** $\langle X', Y'\rangle \in c(R \cup todo)$ **then** skip; `// back to the beginning of the loop`
5 **forall** $a \in A$ **do**
6 insert $\langle t_a^{\#}(X'), t_a^{\#}(Y')\rangle$ in $todo$;
7 insert $\langle X', Y'\rangle$ in R;
8 **return** R;

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

automaton, leaving aside the verification that the output function agrees on all pairs (the test on line 4 in Fig. 1 has been removed). This corresponds to using a function g' instead of g, defined as

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

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

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

$UP(L)$ 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 3.1. ([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 3.2. ([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. The constructed NFA has 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 the notion of *(Büchi) transition monoid* [24], which make the presentation easier and eventually make it possible to propose our algorithm.

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	$\star$	$\star$

Let $\mathcal{A} = \langle S, T\rangle$ be an NBW and 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.

The *Kleene star* M^* of a matrix $M \in \mathscr{M}$ is defined by $M^* := \sum_{i \in \mathbb{N}} M^i$, where the sum is defined componentwise with respect to the + operation defined above on $\mathbb{3}$. As before, the coefficient $M^*(x, x')$ represents the type of the "best" available path of any length from x to x': it is $\star$ if there is a path containing a Büchi transition, 1 if there is a path but not one with a Büchi transition, and 0 if there is no path at all. Using a pumping argument, we can remark that it is enough to consider paths with at most $2|S|$ transitions, so $M^* = \sum_{0 \leq i \leq 2|S|} M^i$.

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 $x \xrightarrow{v^n} y \overset{v^m}{\Longrightarrow} y$. 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 $\mathcal{A}$, as in [24].

Lemma 3.3. 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 3.4. For all sets X of states from $\mathcal{A}$, we have $[X]_{\mathcal{A}^\$} = ([X]_{\mathcal{A}})^\$$.

Example 3.5. To illustrate this construction, consider the NBW depicted on the left in Fig. 3, where double lines represent Büchi transitions. 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 $[1]^\$_{\mathcal{A}} = a^* \$ a^+$.

The corresponding NFA $\mathcal{A}^\$$ is depicted on the right. Its states 0 and 1 form the first layer; they respectively recognise the two previous rational languages. The second layer is reached from those states when reading the letter \$. We only depicted the reachable part of the second layer: those states consisting of matrices of the form T_u for some word u. There are only three such matrices in this example since $T_a \cdot T_b = T_b \cdot T_a = T_b \cdot T_b = T_b$ and $T_a \cdot T_a = T_a$.

[1] To compute M^*, one can use the fact that $M^* = (I + M)^{2n}$ with $n = |S|$, and use iterated squaring.

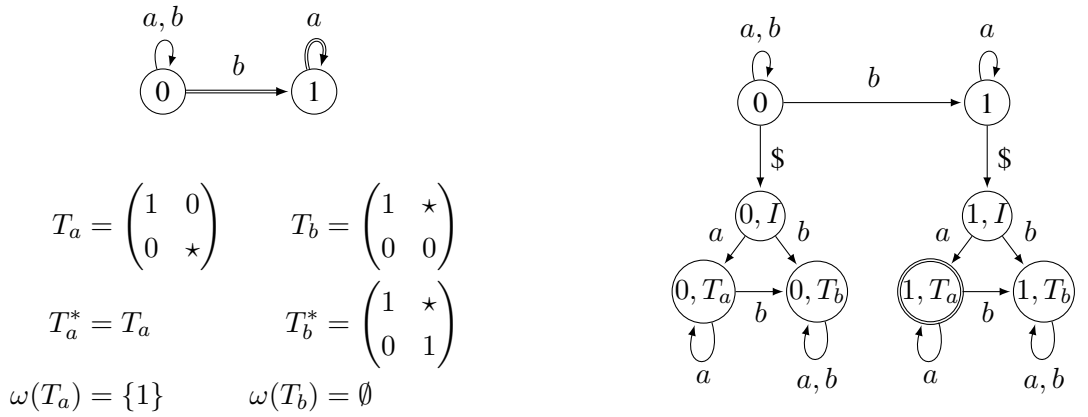


Figure 3. A NBW $\mathcal{A}$ (left) and the reachable part of its associated NFA $\mathcal{A}^{\$}$ (right).

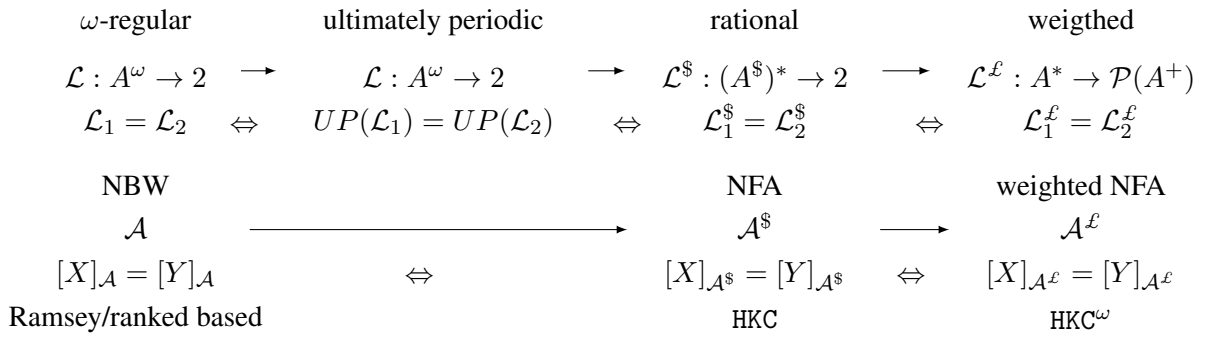


Figure 4. Summary of the operations and algorithms on languages and automata.

By definition, the second layer consists of several blocks (here, two) whose transitions are identical, and which differ only by the accepting status of their states. Given that the first block has no accepting state in this example, it might seem interesting to prune $\mathcal{A}^{\$}$ so that all states may reach an accepting state. We restrain ourselves from doing so because we want to exploit the fact that all blocks share the same structure.

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.

4. HKC for Büchi automata

By Prop. 3.1 and Thm. 3.4, 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 blocks of the second layer, without ever realising that the transition structure of those sub-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 automaton.

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

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

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

$$T^{\pounds}_a(x) = \{x' \mid T_a(x, x') \neq 0\} \qquad\qquad o^{\pounds}(x) = \left\{ v \in A^+ \mid v^\omega \in [x]_{\mathcal{A}} \right\}$$

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 4.1. 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. 3.3 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 4.2. For all sets $X, Y \subseteq S$,

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

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 discriminating set $\emptyset$ is useless to distinguish two sets $X, Y \subseteq S$.

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 $\mathcal{D} := \emptyset$; $\mathcal{M} := \emptyset$; $todo := \{I\}$;
2 **while** $todo \neq \emptyset$ **do**
3 extract M from $todo$;
4 **if** $M \in \mathcal{M}$ **then** skip;
5 **forall** $a \in A$ **do**
6 insert $M \cdot T_a$ in $todo$;
7 insert M in $\mathcal{M}$; insert $\omega(M)$ in $\mathcal{D}$;
8 **return** $\mathcal{D}$;

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

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 $R := \texttt{HKC}'(\mathcal{A}^{\mathcal{L}}, X, Y)$ || $\mathcal{D} := \texttt{Discr}(\mathcal{A})$;
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*;

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

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).

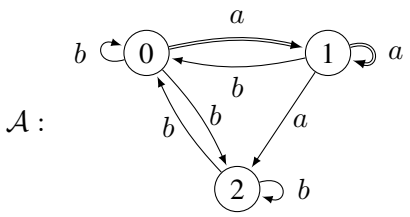
Example 4.3. We execute $\texttt{HKC}^{\omega}$ on the NBW on the left of Fig 7, starting with states $\{0\}$ and $\{1\}$.

The transition monoid has 13 elements, which are listed with their discriminating sets in Fig. 8. In fact the exploration of the monoid by the algorithm `Discr` stops because of the following equations:

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

It then returns three different discriminating sets: $\emptyset$, $\{0, 1\}$, and $\{0, 1, 2\}$, which arise for instance from the matrices T_b, T_a and T_{ba}.

On the other hand 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



$$R = \texttt{HKC}'(\mathcal{A}^{\mathcal{L}}, \{0\}, \{1\}) = \{\langle \{0\}, \{1\}\rangle, \langle \{1\}, \{1,2\}\rangle\}$$
$$\mathcal{D} = \texttt{Discr}(\mathcal{A}) = \{\emptyset, \{0,1\}, \{0,1,2\}\}$$

Figure 7. An example run of $\texttt{HKC}^\omega$

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

u	aab	aba	baa	bab	$abab$	$baab$
T_u	$\begin{pmatrix}\star&0&\star\\\star&0&\star\\0&0&0\end{pmatrix}$	$\begin{pmatrix}0&\star&0\\0&\star&0\\0&0&0\end{pmatrix}$	$\begin{pmatrix}0&\star&\star\\0&\star&\star\\0&\star&\star\end{pmatrix}$	$\begin{pmatrix}\star&0&0\\\star&0&0\\\star&0&0\end{pmatrix}$	$\begin{pmatrix}\star&0&0\\\star&0&0\\0&0&0\end{pmatrix}$	$\begin{pmatrix}\star&0&\star\\\star&0&\star\\\star&0&\star\end{pmatrix}$
T_u^*	$\begin{pmatrix}\star&0&\star\\\star&1&\star\\0&0&1\end{pmatrix}$	$\begin{pmatrix}1&\star&0\\0&\star&0\\0&0&1\end{pmatrix}$	$\begin{pmatrix}1&\star&\star\\0&\star&\star\\0&\star&\star\end{pmatrix}$	$\begin{pmatrix}\star&0&0\\\star&1&0\\\star&0&1\end{pmatrix}$	$\begin{pmatrix}\star&0&0\\\star&1&0\\0&0&1\end{pmatrix}$	$\begin{pmatrix}\star&0&\star\\\star&1&\star\\\star&0&\star\end{pmatrix}$
$\omega(T_u)$	$\{0,1\}$	$\{0,1\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1\}$	$\{0,1,2\}$

Figure 8. Reachable part of the transition monoid of Ex. 4.3. The discriminating sets are calculated using the formula (†) on page 8.

from $\langle \{0\}, \{1\}\rangle$ by reading the words b, aa, and ab, are skipped thanks to the up to congruence technique. For instance the pair $\langle \{0,2\}, \{0\}\rangle$ belongs to the congruence closure of R thanks to the following argument: 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 $\langle \{2\}, \{2\}\rangle$; we finally obtain $\langle \{0,2\}, \{0\}\rangle$ by transitivity and symmetry.

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. Further refinements

A weakness of the algorithm $\texttt{HKC}^\omega$ is that it must fully explore the transition monoid of the starting NBW, which may contain up to 3^{n^2} elements when starting with a NBW with n states. Since the goal of this exploration is to obtain discriminating sets, we would like to isolate parts of the transition monoid that can safely be skipped: for instance because they will lead to discriminating sets which have already been encountered, or which are subsumed by previously encountered ones. This leads us to optimisations which are similar in spirit to those brought by HKC for the analysis of the prefix automaton.

To make this idea precise, given a set of sets of states $\mathcal{E}$, define the following equivalence relation on sets of states:

$$X \sim_{\mathcal{E}} Y \quad \text{if} \quad \forall D \in \mathcal{E}, X \cap D = \emptyset \Leftrightarrow Y \cap D = \emptyset$$

By Prop. 4.2, we can replace the sub-algorithm `Discr` (Fig. 5) by any algorithm returning a subset $\mathcal{D}'$ of $\mathcal{D}$ such that $\sim_{\mathcal{D}'} = \sim_{\mathcal{D}}$.

This sub-algorithm basically computes the least solution to an equation (the least set of matrices containing the identity and closed under multiplication on the right by the transition matrices of the starting NBW), and computes a set of discriminating sets out of this solution. We can improve it by weakening the equation to be satisfied, in the very same way HKC improves over HK by allowing to

look for bisimulations up to congruence rather than bisimulations up to equivalence. We shall use the following abstract lemma about partial orders to prove the correctness of such improvements: This lemma is inspired by the theory of coinduction up-to [25], where the *compatibility* condition $f \circ r \leq r \circ f$ plays a central role. We explain how we will instantiate this lemma below.

Lemma 5.1. Let $\mathcal{X}, \mathcal{Y}$ be two partial orders. Let $r, f \colon \mathcal{X} \to \mathcal{X}$ and $s \colon \mathcal{X} \to \mathcal{Y}$ be three monotone functions such that $f \circ r \leq r \circ f$; $id \leq f$; $f \circ f \leq f$; and $s \circ f \leq s$. Fix $x_0 \in \mathcal{X}$, suppose that x is a least element of $\mathcal{X}$ such that $x_0 \leq x \leq r(x)$, and assume that x' is an element such that $x_0 \leq x' \leq r(f(x'))$ and $x' \leq x$. Then we have $s(x) = s(x')$.

Proof:
Since $f \circ r \leq r \circ f$ and $f \circ f \leq f$, we have $f(x') \leq f(r(f(x'))) \leq r(f(f(x'))) \leq r(f(x'))$. Since $id \leq f$, we also have $x_0 \leq x' \leq f(x')$, so that $x \leq f(x')$ by minimality of x. We deduce $s(x') \leq s(x) \leq s(f(x')) \leq s(x')$ by monotonicity of s and $s \circ f \leq s$. □

Given a set $\mathcal{M}$ of matrices, set $d(\mathcal{M}) = \{\omega(M) \mid M \in \mathcal{M}\}$. We can apply the above lemma by choosing $\mathcal{X} = \langle \mathcal{P}(\mathscr{M}), \subseteq \rangle$, $\mathcal{Y} = \langle Rel(\mathcal{P}(S)), \supseteq \rangle$, $x_0 = \{I\}$, and

$$r(\mathcal{M}) = \{M \mid \forall a \in A, M \cdot T_a \in \mathcal{M}\} \qquad s(\mathcal{M}) = \sim_{d(\mathcal{M})}$$

We have $\mathcal{M} \subseteq r(\mathcal{M})$ if and only if $\mathcal{M}$ is closed under multiplication on the right by the $(T_a)_{a \in A}$. Accordingly, the x from the statement of the lemma is the set of matrices $\mathcal{M} = \{T_u \mid u \in A^*\}$ obtained at the end of the execution of `Discr`. It follows that $d(x)$ is the returned set $\mathcal{D}$ of discriminating sets, and $s(x)$ is the equivalence relation $\sim_{\mathcal{D}}$.

We will show how to instantiate the function f from the lemma in the following sections. Intuitively, a function f satisfying the other requirements of the lemma can be used as an up-to technique, in order to skip elements from the transition monoid. Indeed, we can obtain an algorithm $\texttt{Discr}_f$ by replacing line 4 from `Discr` (Fig. 5) with

4' | **if** $M \in f(\mathcal{M} \cup todo)$ **then** skip;

This algorithm terminates with a subset $\mathcal{M}' \subseteq \mathcal{M}$ of matrices corresponding to the x' from the statement of the lemma, and returns a set $\mathcal{D}'$ of discriminating sets for which the lemma guarantees that we have $\sim_{\mathcal{D}'} = \sim_{\mathcal{D}}$, as required.

Such techniques can drastically improve performances: when an element is skipped thanks to the up-to technique, all elements which were reachable only through this element virtually disappear. We give two examples of such techniques in the sequel.

5.1. Working up to unions

A first property which we can exploit in order to cut-down the exploration of the transition monoid is the following: if two discriminating sets D, D' have been discovered, then their union

$D \cup D'$ is not useful as a discriminating set. Formally, for all $\mathcal{D} \subseteq \mathcal{P}(S)$, if $D, D' \in \mathcal{D}$ then $\sim_{\{D \cup D'\} \cup \mathcal{D}} = \sim_{\mathcal{D}}$.

One could think that this should allow us to skip matrices from the transition monoid when they can be written as sums of already visited matrices. This is however wrong, because the discriminating set of a sum is in general *not* the union of the underlying discriminating sets. For instance, we have:

$$\omega\begin{pmatrix} 0 & \star \\ 1 & 0 \end{pmatrix} = \{0,1\} \neq \emptyset \cup \emptyset = \omega\begin{pmatrix} 0 & \star \\ 0 & 0 \end{pmatrix} \cup \omega\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$$

In order to find an operation on matrices which corresponds to unions when taking discriminating sets, we need to slightly generalise the notion of matrix.

Say that a matrix is a *vector* if it contains at most one non-zero coefficient per line. Let $\mathscr{V}$ denote the set of vectors. The three matrices above are vectors. A *generalised matrix* is a set of vectors. We write $\mathscr{M}'$ for the set of generalised matrices. We order vectors and matrices pointwise using the order $0 < 1 < \star$. Given a matrix M, we write $\underline{M}$ for the generalised matrix $\{V \in \mathscr{V} \mid V \leq M\}$. While the map $M \mapsto \underline{M}$ is injective, it is not surjective: there are generalised matrices which cannot be represented using a single matrix. We equip $\mathscr{M}'$ with a sum $\oplus$, a mixed product $\bullet$, and an operation ω as follows ($\mathbf{M}, \mathbf{N}$ range over generalised matrices, N ranges over matrices, V ranges vectors):

$$\mathbf{M} \oplus \mathbf{N} = \mathbf{M} \cup \mathbf{N} \qquad \mathbf{M} \bullet N = \bigcup_{V \in \mathbf{M}} \underline{V \cdot N} \qquad \omega(\mathbf{M}) = \bigcup_{V \in \mathbf{M}} \omega(V)$$

By definition, the mixed product is distributive on the left, and the function ω is a homomorphism of semilattices:

Lemma 5.2. For all generalised matrices $\mathbf{M}, \mathbf{N} \in \mathscr{M}'$ and matrix $O \in \mathscr{M}$, we have

$$(i)\ (\mathbf{M} \oplus \mathbf{N}) \bullet O = (\mathbf{M} \bullet O) \oplus (\mathbf{N} \bullet O) \qquad (ii)\ \omega(\mathbf{M} \oplus \mathbf{N}) = \omega(\mathbf{M}) \cup \omega(\mathbf{N})$$

The counter-example above shows that in general, $\underline{M} \oplus \underline{N} \neq \underline{M + N}$. However, we do have:

Lemma 5.3. For all matrices $M, N \in \mathscr{M}$, we have:

$$(i)\ \underline{M} \bullet N = \underline{M \cdot N} \qquad (ii)\ \omega(\underline{M}) = \omega(M)$$

Proof:

(i) We have

$$\underline{M} \bullet N = \bigcup_{V \leq M} \underline{V \cdot N} \qquad \underline{M \cdot N} = \{U \in \mathscr{V} \mid U \leq M \cdot N\}$$

The direct inclusion comes from monotonicity of matrix multiplication: if $U \leq V \cdot N$ for some vector $V \leq M$, then $V \cdot N \leq M \cdot N$, whence $U \leq M \cdot N$. For the other inclusion, assume a vector $U \leq M \cdot N$. The non-empty elements of U can be described by a function

δ associating to each row i the column $\delta(i)$ of the non-empty coefficient of that row (or an arbitrary one if the row is all zeros). For all row i, we have $U_{i,\delta(i)} \leq \Sigma_k M_{i,k} N_{k,\delta(i)}$, and we can find an index $\delta'(i)$ such that $\Sigma_k M_{i,k} N_{k,\delta(i)} = M_{i,\delta'(i)} N_{\delta'(i),\delta(i)}$. δ' determines a vector $V \leq M$ such that $U \leq V \cdot N$, as required.

(ii) The fact that $x \in \omega(M)$ is witnessed by an accepting lasso in M, and such a lasso can be assumed to be simple (i.e., every state is visited at most once, except the last visited state which is visited twice). Such a simple lasso yields $x \in \omega(V)$ for a vector $V \leq M$ (just select in M those transitions that are required by the simple lasso).

□

Now lift the functions r, d, s we defined after Lem. 5.1 to sets of generalised matrices $\mathcal{P}(\mathscr{M}')$:

$$r(E) = \{\mathbf{M} \mid \forall a \in A,\ \mathbf{M} \bullet T_a \in E\} \qquad d(E) = \{\omega(\mathbf{M}) \mid \mathbf{M} \in E\} \qquad s(E) = \sim_{d(E)}$$

Finally define the up-to technique as the following function $u\colon \mathcal{P}(\mathscr{M}') \to \mathcal{P}(\mathscr{M}')$:

$$u(E) = \{\mathbf{M}_1 \oplus \cdots \oplus \mathbf{M}_n \mid n \in \mathbb{N},\ \forall i \leq n,\ \mathbf{M}_i \in E\}$$

Intuitively, this function allows us to cut-down the exploration on the transition monoid whenever we encounter a matrix which can be written as a union (in the sense of $\oplus$) of already encountered matrices.

Proposition 5.4. The functions r, u and s satisfy the requirements of Lem. 5.1 (taking u for f).

Proof:
For compatibility of u w.r.t. r ($u \circ r \leq r \circ u$), let $\mathbf{M}_1 \oplus \cdots \oplus \mathbf{M}_n$ with $\forall a, i \leq n,\ \mathbf{M}_i \bullet T_a \in E$ be an element of $u(r(E))$ for some E. We have to show that this sum belongs to $r(u(e))$, i.e., $\forall a, (\mathbf{M}_1 \oplus \cdots \oplus \mathbf{M}_n) \bullet T_a \in u(E)$. This follows directly from distributivity of $\bullet$ over $\oplus$ (Lem. 5.2(i)).

The function u is obviously extensive ($\mathrm{id} \leq u$) and idempotent ($u \circ u = u$).

The last requirement ($s \circ u \leq s$) follows from Lem. 5.2(ii) and the observation at the beginning of Sect. 5.1. □

Generalised matrices are not convenient to use in practice: many matrices expand into generalised matrices of exponential size. However, we use them only to establish the correctness of the optimisation: thanks to Lem. 5.3, the version of the algorithm `Discr` where we use the function u to cut down the search-space only manipulates generalised matrices of the form $\underline{M}$, which can thus be represented as plain matrices.

It remains to check that we can implement the refined check on line 4'. The following lemma shows that this is relatively expensive (at least theoretically, since state-of-the art SAT solvers tend to be efficient in practice).

Proposition 5.5. Given a set $\mathcal{M}$ of matrices and a matrix N, the problem of deciding if $\underline{N} \in u(\{\underline{M} \mid M \in \mathcal{M}\})$ is coNP-complete.

Proof:
We prove hardness in App. A. For membership in coNP, observe that $\underline{N} \in u(\{\underline{M} \mid M \in \mathcal{M}\})$ iff $\forall V \in \underline{N},\ \exists M \in \mathcal{M},\ M \leq N$ and $V \in \underline{M}$. The existential subformula can be checked in polynomial time. □

Example 5.6. When running this refined version of $\mathtt{HKC}^\omega$ on the NBW on the top right in Fig. 9, the up-to-union technique makes it possible to explore only 11 matrices of the monoid, although it contains 17 elements. Indeed, 4 matrices are skipped, being recognised as sums of previously encountered matrices, and 2 matrices are not even computed because they are reachable only through the 4 previous matrices.

The explored part of the transition monoid of the NBW is detailed in Fig. 9, together with the discriminating sets associated to its elements and the justification for the elements skipped thanks to the up-to-union technique. Note that $T_{ca} = T_{bc} + T_{ccc}$ but $\underline{T_{ca}} \neq \underline{T_{bc}} \oplus \underline{T_{ccc}}$.

5.2. Working up to equivalence

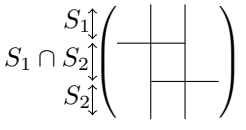
There is also room for improvement when we start with a disjoint union of NBWs: the starting NBWs most probably contain loops, and the transition monoid of the disjoint union will need to unfold those loops until they 'synchronise'. Take for instance the two NBWs $\mathcal{A}_1$ and $\mathcal{A}_2$ over a single letter a, defined by the two matrices on the left in Fig 10, whose disjoint $\mathcal{A}$ union can be represented by the diagonal block matrix on the right.

We have $T_{1_{(aa)a}} = T_{1_a}$: the transition monoid of $\mathcal{A}_1$ has size 3 (including I); we have $T_{2_{(aaa)a}} = T_{2_a}$: the transition monoid of $\mathcal{A}_2$ has size 4; and we have $T_{(aaaaaa)a} = T_a$: the transition monoid of $\mathcal{A}$ has size 7. Generalising 2 and 3 into n and m in the example, the transition monoid of the disjoint union contains $\mathtt{lcm}(n, m) + 1$ matrices. By designing an up-to-equivalence technique reminiscent of the one used in Hopcroft and Karp's algorithm, we will obtain an algorithm that explores at most the first $n + m + 1$ matrices. (On this specific example all matrices but I give rise to the same discriminating set, so that we could stop even earlier; but there is no generic argument behind this observation.)

We fix in the sequel a NBW $\mathcal{A} = \langle S, T\rangle$ and two subsets $S_1, S_2 \subseteq S$. Let $\mathscr{M}_d$ be the set of matrices M such that:

$$\forall i \in \{1, 2\},\ \forall x, y \in S,\ x \in S_i \ \wedge\ M(x, y) \neq 0 \Rightarrow y \in S_i$$

Such matrices look like the picture on the right. We require $T_a \in \mathscr{M}_d$ for all $a \in A$: states from S_i should only reach states from S_i. Since $\mathscr{M}_d$ is closed under products (it actually forms a sub-semiring of $\mathscr{M}$), we deduce $T_u \in \mathscr{M}_d$ for all $u \in A^*$.



If $S_1 = S_2 = S$ then the requirement is void, as well as the optimisation to be described below; if $S_1 \cap S_2 = \emptyset$ and $S_1 \cup S_2 = S$ then this corresponds to the case where $\mathcal{A}$ is a disjoint union of two NBWs. Intermediate cases are allowed. In practice if we want to test the equivalence between starting sets X and Y we will take as S_1 (resp. S_2) the set of accessible states from X (resp. Y).

For $i = 1, 2$, let $\mathscr{M}_i$ be the set of matrices indexed by S_i and let $\pi_i \colon \mathscr{M}_d \to \mathscr{M}_i$ be the obvious surjective semiring homomorphism. For all $M \in \mathscr{M}_d$, we have $\omega(M) \cap S_i = \omega(\pi_i(M))$. Define

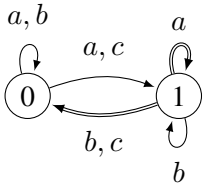
Kept matrices

u	T_u	$\omega(T_u)$
ϵ	$\begin{pmatrix}1&0\\0&1\end{pmatrix}$	$\varnothing$
a	$\begin{pmatrix}1&1\\0&\star\end{pmatrix}$	$\{0,1\}$
b	$\begin{pmatrix}1&0\\\star&1\end{pmatrix}$	$\varnothing$
c	$\begin{pmatrix}0&1\\\star&0\end{pmatrix}$	$\{0,1\}$
aa	$\begin{pmatrix}1&\star\\0&\star\end{pmatrix}$	$\{0,1\}$
ac	$\begin{pmatrix}\star&1\\\star&0\end{pmatrix}$	$\{0,1\}$
bc	$\begin{pmatrix}0&1\\\star&\star\end{pmatrix}$	$\{0,1\}$
ca	$\begin{pmatrix}0&\star\\\star&\star\end{pmatrix}$	$\{0,1\}$
cc	$\begin{pmatrix}\star&0\\0&\star\end{pmatrix}$	$\{0,1\}$
bcc	$\begin{pmatrix}\star&0\\\star&\star\end{pmatrix}$	$\{0,1\}$
ccc	$\begin{pmatrix}0&\star\\\star&0\end{pmatrix}$	$\{0,1\}$

Equations

$$
\begin{aligned}
T_{bb} &= T_b \\
T_{cb} &= T_{ac} \\
T_{acb} &= T_{ac} \\
T_{aaa} &= T_{aa} \\
T_{aab} &= T_{aca} \\
T_{aac} &= T_{ac} \\
T_{bca} &= T_{ca} \\
T_{bcb} &= T_{ab} \\
T_{caa} &= T_{ca} \\
T_{cab} &= T_{aab} \\
T_{cac} &= T_{bcc} \\
T_{cca} &= T_{acc} \\
T_{ccb} &= T_{bcc} \\
T_{bccc} &= T_{ca}
\end{aligned}
$$

Automaton



Skipped matrices (by union)

$$
\begin{aligned}
T_{ab} &= \begin{pmatrix}\star&1\\\star&\star\end{pmatrix} = T_{ac} \oplus T_{cc} \oplus T_{bc} \\
T_{ba} &= \begin{pmatrix}1&1\\\star&\star\end{pmatrix} = T_b \oplus T_{bc} \oplus T_a \\
T_{acc} &= \begin{pmatrix}\star&\star\\0&\star\end{pmatrix} = T_{aa} \oplus T_{cc} \\
T_{aca} &= \begin{pmatrix}\star&\star\\\star&\star\end{pmatrix} = T_{ab} \oplus T_{ca}
\end{aligned}
$$

Accessible matrices not generated

$$
\begin{aligned}
T_{baa} &= \begin{pmatrix}1&\star\\\star&\star\end{pmatrix} \quad ; \quad \omega(T_{baa}) = \{0,1\} \\
T_{accc} &= \begin{pmatrix}\star&\star\\\star&0\end{pmatrix} \quad ; \quad \omega(T_{accc}) = \{0,1\}
\end{aligned}
$$

Figure 9. Exploration of a transition monoid with up-to-union technique. To alleviate notations, we identified matrices M with their associated generalised matrices $\underline{M}$.

the following function $e' \colon \mathcal{P}(\mathscr{M}_d) \to \mathcal{P}(\mathscr{M}_d)$:

$$e'(\mathcal{M}) = \{N \mid \langle \pi_1(N), \pi_2(N)\rangle \in e(\{\langle \pi_1(M), \pi_2(M)\rangle \mid M \in \mathcal{M}\})\}$$

where $e(R)$ denotes the equivalence closure of a relation R, here for relations on $\mathscr{M}_1 \uplus \mathscr{M}_2$.

Like in the previous section, we will show by using Lem. 5.1, that when $\mathtt{HKC}^\omega$ is restricted to starting sets $\langle X, Y\rangle \in \mathcal{P}(S_1) \times \mathcal{P}(S_2)$, it remains correct when using e' as an up-to technique on line 4 from Fig. 5.

$$T_{1a} = \begin{pmatrix} 0 & \star \\ \star & 0 \end{pmatrix} \qquad T_{2a} = \begin{pmatrix} 0 & \star & 0 \\ 0 & 0 & \star \\ \star & 0 & 0 \end{pmatrix} \qquad T_a = \begin{pmatrix} 0 & \star & & & \\ \star & 0 & & & \\ & & 0 & \star & 0 \\ & & 0 & 0 & \star \\ & & \star & 0 & 0 \end{pmatrix}$$

Figure 10. Motivating example for up-to-equivalence technique.

We need to work in a larger structure than sets of matrices. Moreover, we need to turn the set of discriminating sets into a relation. Set $U = \{1\} \times S_1 \cup \{2\} \times S_2$. Given a relation $\mathcal{E} \in Rel(U)$, define the following relation between $\mathcal{P}(S_1)$ and $\mathcal{P}(S_2)$:

$$X_1 \approx_{\mathcal{E}} X_2 \quad \text{if} \quad \forall \langle \langle i, D \rangle, \langle j, D' \rangle \rangle \in \mathcal{E},\ X_i \cap D = \emptyset \Leftrightarrow X_j \cap D' = \emptyset$$

(We define $\approx_{\mathcal{E}}$ as a relation between $\mathcal{P}(S_1)$ and $\mathcal{P}(S_2)$ because when starting with sets $X_1 \subseteq S_1$ and $X_2 \subseteq S_2$, `HKC`' will return such a relation.)

Set $\mathscr{M}'' = (\{1\} \times \mathscr{M}_1 \ \cup \ \{2\} \times \mathscr{M}_2)^2$, write iM for the pair $\langle i, M \rangle \in \{i\} \times \mathscr{M}_i$ and define a mixed product operation $\cdot : \mathscr{M}'' \times \mathscr{M}_d \to \mathscr{M}''$ by setting:

$$\langle iM, jN \rangle \cdot O = \langle i(M \cdot \pi_i(O)), j(N \cdot \pi_j(O)) \rangle$$

Now lift the functions r, s we defined after Lem. 5.1 to work on $\mathcal{P}(\mathscr{M}'')$:

$$\begin{aligned} r(R) &= \{\mathbf{M} \in \mathscr{M}'' \mid \forall a \in A,\ \mathbf{M} \cdot T_a \in R\} \\ d(R) &= \{\langle i\omega(M), j\omega(N) \rangle \mid \langle iM, jN \rangle \in R\} \\ s(R) &= \approx_{d(R)} \end{aligned}$$

Recall that e is the function taking the equivalence closure of a relation.

Proposition 5.7. The functions r, e and s satisfy the requirements of Lem. 5.1 (taking e for f).

Proof:
For compatibility of e w.r.t. r ($e \circ r \leq r \circ e$), assume $\langle i_1 M_1, i_n M_n \rangle \in e(r(R))$. There are $(i_k, M_k)_{k \in [2..n[}$ such that for all $k < n$, either $\langle i_k M_k, i_{k+1} M_{k+1} \rangle \in r(R)$ or $\langle i_{k+1} M_{k+1}, i_k M_k \rangle \in r(R)$. We need to show that $\langle i_1 M_1, i_n M_n \rangle \in r(e(R))$. Let $a \in A$; for all $k < n$, either $\langle i_k M_k, i_{k+1} M_{k+1} \rangle \cdot T_a \in R$ or $\langle i_{k+1} M_{k+1}, i_k M_k \rangle \cdot T_a \in R$, which means by definition that $\langle i_k M_k \cdot \pi_k(T_a), i_{k+1} M_{k+1} \cdot \pi_{k+1}(T_a) \rangle \in R$ or $\langle i_{k+1} M_{k+1} \cdot \pi_{k+1}(T_a), i_k M_k \cdot \pi_k(T_a) \rangle \in R$. Therefore, for all $a \in A$, $\langle i_1 M_1 \cdot \pi_1(T_a), i_n M_n \cdot \pi_n(T_a) \rangle \in e(R)$, which means $\langle i_1 M_1, i_n M_n \rangle \in r(e(R))$, as required.

The function e is obviously extensive and idempotent, so that it only remains to show that $s \circ e \leq s$, i.e., for all R, $s(R) \subseteq s(e(R))$ (recall that we take reverse inclusions for the partial order $\mathcal{Y}$). Suppose $\langle X_1, X_2 \rangle \in s(R)$, i.e., $X_1 \approx_{d(R)} X_2$, we have to show $\langle X_1, X_2 \rangle \in s(e(R))$, i.e.,

$X_1 \approx_{d(e(R))} X_2$. Let $\langle i_1 D_1, i_n D_n\rangle \in d(e(R))$. There are M_1, M_n such that $D_1 = \omega(M_1)$, $D_n = \omega(M_n)$, and $(i_k, M_k)_{k\in[2..n[}$ such that for all $k < n$, either $\langle i_k M_k, i_{k+1} M_{k+1}\rangle \in R$ or $\langle i_{k+1} M_{k+1}, i_k M_k\rangle \in R$. Since $X_1 \approx_{d(R)} X_2$, we deduce that for all $k < n$, either $X_{i_k} \cap \omega(M_k) = \emptyset \Leftrightarrow X_{i_{k+1}} \cap \omega(M_{k+1}) = 0$, or $X_{i_{k+1}} \cap \omega(M_{k+1}) = \emptyset \Leftrightarrow X_{i_k} \cap \omega(M_k) = 0$, which is just the same. By transitivity of logical equivalence, we deduce that $X_{i_1} \cap \omega(M_1) = \emptyset \Leftrightarrow X_{i_n} \cap \omega(M_n) = 0$, as required. □

Overloading the notation from Sect. 5.1, given a matrix $M \in \mathscr{M}_d$, write $\underline{M} = \langle 1\pi_1(M), 2\pi_2(M)\rangle \in \mathscr{M}''$. Then we have:

$$(1)\ \forall\, M, N \in \mathscr{M}_d,\ \underline{M \cdot N} = \underline{M} \cdot N \quad (2)\ \forall\, X_1 \subseteq S_1, X_2 \subseteq S_2,\ X_1 \sim_{\omega(M)} X_2 \text{ iff } X_1 \approx_{d(\underline{M})} X_2$$

The first property guarantees that when taking $x_0 = \{\underline{I}\}$, the x from Lem. 5.1 is the set $\{\underline{T_u} \mid u \in A^*\}$. The second property ensures that $s(x)$ properly discriminates the pairs provided by HKC' (R). By Lem. 5.1, so does $s(x')$, which can easily be shown to correspond to the computation with the optimised algorithm $\texttt{Discr}_{e'}$, where we use the up-to-equivalence technique to skip redundant matrices.

As in Hopcroft and Karp's algorithm [17], one can implement the up-to-equivalence test efficiently using an appropriate union-find data structure.

Example 5.8. When running this refined version of $\texttt{HKC}^\omega$ on the NBW over a single letter defined by the matrix T_a in Fig. 10, the up-to-equivalence technique makes it possible to retain only 5 matrices of the monoid, although it contains 7 elements. Indeed, the matrix T_{a^5} is skipped because the pair of its components is in the equivalence closure of the set of pairs of components of already explored matrices. The explored part of the transition monoid and the skipped matrix are detailed in Fig. 11. Note that T_{a^6} and T_{a^7} do not even need to be generated.

Kept matrices

$$u = \quad \epsilon \qquad a \qquad aa \qquad aaa \qquad aaaa$$

$$T_u = \begin{pmatrix} 1 & 0 & & & \\ 0 & 1 & & & \\ & & 1 & 0 & 0 \\ & & 0 & 1 & 0 \\ & & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & \star & & & \\ \star & 0 & & & \\ & & 0 & \star & 0 \\ & & 0 & 0 & \star \\ & & \star & 0 & 0 \end{pmatrix} \begin{pmatrix} \star & 0 & & & \\ 0 & \star & & & \\ & & 0 & 0 & \star \\ & & \star & 0 & 0 \\ & & 0 & \star & 0 \end{pmatrix} \begin{pmatrix} 0 & \star & & & \\ \star & 0 & & & \\ & & \star & 0 & 0 \\ & & 0 & \star & 0 \\ & & 0 & 0 & \star \end{pmatrix} \begin{pmatrix} \star & 0 & & & \\ 0 & \star & & & \\ & & 0 & \star & 0 \\ & & 0 & 0 & \star \\ & & \star & 0 & 0 \end{pmatrix}$$

Skipped matrix

$$T_{aaaaa} = \begin{pmatrix} 0 & \star & & & \\ \star & 0 & & & \\ & & 0 & 0 & \star \\ & & \star & 0 & 0 \\ & & 0 & \star & 0 \end{pmatrix} \quad \text{because} \quad \begin{pmatrix} 0 & \star \\ \star & 0 \end{pmatrix} \sim \begin{pmatrix} 0 & \star & 0 \\ 0 & 0 & \star \\ \star & 0 & 0 \end{pmatrix} \sim \begin{pmatrix} \star & 0 \\ 0 & \star \end{pmatrix} \sim \begin{pmatrix} 0 & 0 & \star \\ \star & 0 & 0 \\ 0 & \star & 0 \end{pmatrix}$$

Figure 11. Exploration of a transition monoid with up-to-equivalence technique.

6. 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. The coinductive framework makes it possible to develop up-to techniques similar to the ones used in HKC in order to compute the discriminating sets more efficiently. We provide two such techniques, namely coinduction up to unions (Sect. 5.1) and coinduction up to equivalence (Sect. 5.2). It is not clear to us whether the two techniques can be used at the same time.

We also want to investigate how to exploit techniques using simulation relations, which were successfully used in [10, 1, 2, 23] and which tend to nicely fit in the coinductive framework we exploit here [4, Sect. 5].

Our algorithm stems from the construction of Calbrix et al. [6], which we revisited using notions from [24] in Sect. 3. $\mathtt{HKC}^\omega$ is rather close to *Ramsey-based* algorithms [11, 1] (as opposed to *rank-based* ones [22, 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.

A prototype implementation is available at `https://framagit.org/dpous/hkcw`; it makes it possible to test several combinations of up-to techniques.

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 *CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 132–147. Springer, 2010. doi:10.1007/978-3-642-14295-6_14.

[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 *CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 187–202. Springer, 2011. doi:10.1007/978-3-642-23217-6_13.

[3] P. A. Abdulla, Y.-F. Chen, L. Holík, R. Mayr, and T. Vojnar. When simulation meets antichains. In *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 158–174. Springer, 2010. doi:10.1007/978-3-642-12002-2_14.

[4] F. Bonchi and D. Pous. Checking NFA equivalence with bisimulations up to congruence. In *POPL*, pages 457–468. ACM, 2013. doi:10.1145/2429069.2429124.

[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 *MFPS*, volume 802 of *Lecture Notes in Computer Science*, pages 554–566. Springer, 1993. doi:10.1007/3-540-58027-1_27.

[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 *TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 451–465. Springer, 2007. doi:10.1007/978-3-540-71209-1_34.

[9] L. Doyen and J. Raskin. Antichains for the automata-based approach to model-checking. *Logical Methods in Computer Science*, 5(1), 2009. doi:10.2168/LMCS-5(1:5)2009.

[10] L. Doyen and J.-F. Raskin. Antichain Algorithms for Finite Automata. In *TACAS*, volume 6015 of *Lecture Notes in Computer Science*. Springer, 2010. doi:10.1007/978-3-642-12002-2_2.

[11] S. Fogarty and M. Y. Vardi. Büchi complementation and size-change termination. In *TACAS*, volume 5505 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2009. doi:10.1007/978-3-642-00768-2_2.

[12] S. Fogarty and M. Y. Vardi. Efficient Büchi universality checking. In *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 205–220. Springer, 2010. doi:10.1007/978-3-642-12002-2_17.

[13] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *CAV*, pages 53–65. Springer, 2001.

[14] S. Gurumurthy, O. Kupferman, F. Somenzi, and M. Y. Vardi. On complementing nondeterministic Büchi automata. In *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 *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. URL: http://techreports.library.cornell.edu:8081/Dienst/UI/1.0/Display/cul.cs/TR71-114.

[18] M. Hutagalung, M. Lange, and E. Lozes. Revealing vs. concealing: More simulation games for Büchi inclusion. In *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. doi:10.1006/inco.1994.1037.

[21] D. Kuperberg, L. Pinault, and D. Pous. Web appendix for this paper, 2019. URL: http://perso.ens-lyon.fr/damien.pous/covece/hkcw.

[22] O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3):408–429, 2001. doi:10.1145/377978.377993.

[23] R. Mayr and L. Clemente. Advanced automata minimization. In *POPL, 2013*, pages 63–74. ACM, 2013. doi:10.1145/2429069.2429079.

[24] 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.

[25] D. Pous. Coinduction all the way up. In *LICS*, pages 307–316. ACM, 2016. doi:10.1145/2933575.2934564.

[26] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975. doi:10.1145/321879.321884.

[27] A. Tarski. A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific Journal of Mathematics*, 5(2):285–309, June 1955.

[28] 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.

[29] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for concurrency*, pages 238–266. Springer, 1996.

[30] M. D. Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 17–30. Springer, 2006. doi:10.1007/11817963_5.

A. CoNP-completeness of reasoning up to union

Theorem A.1. (Prop. 5.5)
Given a set $\mathcal{M}$ of matrices and and matrix N, deciding whether $\underline{N} \in u(\{\underline{M} \mid M \in \mathcal{M}\})$ is coNP-hard.

Proof:
Let's first detail what it means for $\underline{N}$ to be in $u(\{\underline{M} \mid M \in \mathcal{M}\})$:

$$\begin{aligned}\underline{N} \in u(\{\underline{M} \mid M \in \mathcal{M}\}) &\Leftrightarrow \underline{N} \in \cup_{\{M \in \mathcal{M} | M \leq N\}} \underline{M} \\ &\Leftrightarrow \forall V \in \underline{N}, V \in \cup_{\{M \in \mathcal{M} | M \leq N\}} \underline{M} \\ &\Leftrightarrow \forall V \in \underline{N}, \exists M \in \mathcal{M}, M \leq N \text{ and } \exists V' \in \underline{M} \text{ s.t. } V \leq V' \\ &\Leftrightarrow \forall V \in \underline{N}, \exists M \in \mathcal{M}, M \leq N \text{ and } V \in \underline{M}\end{aligned}$$

To show that the problem is coNP-Hard we will show that its complementary problem is NP-Hard via a reduction from 3-SAT. Let $\mathcal{I} = C_1 \wedge \cdots \wedge C_k$ be an instance of 3-SAT. We note $x_1, \ldots, x_n$ the Boolean variables of $\mathcal{I}$. We construct an instance of our problem as:

$$N = \begin{pmatrix} 1 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 1 & 0 & \cdots & 0 \end{pmatrix} \quad \mathcal{M} = \left\{ M_i : \begin{pmatrix} y_1^i & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ y_n^i & 0 & \cdots & 0 \end{pmatrix} \right\}_{1 \leq i \leq k} \qquad y_j^i = \begin{cases} 1\,0 \text{ if } \overline{x_j} \in C_i \\ 0\,1 \text{ if } x_j \in C_i \\ 1\,1 \text{ if } \overline{x_j}, x_j \notin C_i \end{cases}$$

We can do some observations on this instance:

- For all $M_i \in \mathcal{M}$, $M_i \leq N$.
- There is a bijection between the set of truth value distribution of $x_1, \ldots, x_n$ and $\underline{N}$ via the function:

$$f : \begin{cases} 2^n & \to \underline{N} \\ \delta & \mapsto V_\delta \end{cases} \qquad (V_\delta)_j = \begin{cases} 1\,0\,0 \cdots 0 \text{ iff } \delta(x_j) = \top \\ 0\,1\,0 \cdots 0 \text{ iff } \delta(x_j) = \bot \end{cases}$$

- For any $M_i \in \mathcal{M}$, $V_\delta \in \underline{M_i}$ if and only if δ does not satisfy the clause C_i.

Then:

$$\begin{aligned}\mathcal{I} \text{ is not satisfiable } &\Leftrightarrow \forall \delta \in 2^n, \exists C_i \text{ s.t. } \delta \text{ does not satisfy } C_i \\ &\Leftrightarrow \forall V_\delta \in f(2^n), \exists M_i \in \mathcal{M} \text{ s.t. } V_\delta \in \underline{M_i} \\ &\Leftrightarrow \forall V \in \underline{N}, \exists M_i \in \mathcal{M} \text{ s.t. } V \in \underline{M_i} \\ &\Leftrightarrow \underline{N} \in u(\{\underline{M} \mid M \in \mathcal{M}\})\end{aligned}$$

We have shown than $\mathcal{I}$ is satisfiable if and only if $\underline{N} \notin u(\{\underline{M} \mid M \in \mathcal{M}\})$.

The initial problem is thus coNP-hard. □