Coinductive algorithms for Büchi automata

Denis Kuperberg, Laureline Pinault, Damien Pous

Coinductive algorithms for Büchi automata

Denis Kuperberg, Laureline Pinault, and Damien Pous

Plume team, LIP, CNRS - ENS Lyon, 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) [28,13].

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 [5,14,1,17] and implemented in various tools [15,27,22].

Two families of algorithms where recently discovered for non-deterministic automata on finite words, which drastically improved over the pre-existing ones: antichain-based algorithms [29,3,10] 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 the antichain techniques and an older technique for deterministic automata, due to Hopcroft and Karp [16].

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 determinisation operation, 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, where determinisation is available.

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 period 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 nevertheless show that coinductive techniques like the ones at work in [4] can also be used there.

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 [23] explicit (Sect. 3). We define a first version of the algorithm $\mathtt{HKC}^\omega$ in Sect. 4, which we refine in Sect. 5 using techniques for reasoning about the transition monoid. We conclude with directions for future work (Sect. 6).

Notation

We denote sets by capital letters $X, Y, S, T \dots$ and functions by lower case letters $f, g, \dots$ 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 $\mathit{Rel}(S) = \mathcal{P}(S^2)$. Given a relation $R \in \mathit{Rel}(X)$, we write $x\ R\ y$ for $\langle x, y\rangle \in R$. We fix an alphabet A of letters ranged over using 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, represented as functions from natural numbers to A. We use 2 for the set $\{0, 1\}$ (Booleans) and 3 for the set $\{0, 1, \star\}$.

A semilattice as 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 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 $t\colon S \to S^A$ is the transition function which returns, for each state x and for each letter $a \in A$, 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 $[\cdot]_{\mathcal{A}}$ mapping states to weighted languages with weights in O (O^{A^*}), 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 $b\colon Rel(S) \to Rel(S)$ be the function on relations defined as*

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

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

The above function b being monotone, it admits the union of all bisimulations as a greatest fixpoint, by Knaster-Tarski's theorem [18,26]. This greatest-fixpoint is actually language equivalence:

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

This theorem immediately gives a naive and quadratic algorithm for checking language equivalence (Fig. 1): given two states $x, y \in S$, 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.

The standard algorithm by Hopcroft and Karp [16], which is almost linear [25], 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 $R \subseteq b(e(R))$.*

```
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: $\langle x, y\rangle \in R \subseteq b(f(R \cup todo))$
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;
6      forall $a \in A$ do
7          insert $\langle t_a(x'), t_a(y')\rangle$ in $todo$;
8      end
9      insert $\langle x', y'\rangle$ in $R$;
10 end
11 return $true$; // because: $\langle x, y\rangle \in R \subseteq b(f(R))$
```

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 ($\texttt{HK}(\mathcal{A}, x, y)$).

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: we start with a discrete partition with n equivalence classes and each insertion merges two of them.

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 $t\colon S \to \mathcal{P}(S)^A$ is the transition function which returns, for each state x and for each letter $a \in A$, 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* $c\colon Rel(\mathcal{P}(S)) \to Rel(\mathcal{P}(S))$ *be the function mapping a relation* R *to the least equivalence relation* T *containing* R *and such that* $X\ T\ Y$ *and* $X'\ T\ Y'$ *entail* $(X+X')\ T\ (Y+Y')$ *for all* X, X', Y, Y'*. A* bisimulation up to congruence *is a relation* R *such that* $R \subseteq b(c(R))$*.*

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 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 automaton, leaving aside the verification that the output function agrees on all encountered pairs. 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.

The advantage of `HKC` over `HK` is that it often makes it possible to skip reachable subsets from the determinised automaton, 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

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 $T\colon A \to 3^{S^2}$ is the transition function.

```
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 $\forall \langle X', Y'\rangle \in R,\ o^{\#}(X') = o^{\#}(Y')$

$R := \emptyset$; $todo := \{\langle X, Y\rangle\}$;
while $todo \neq \emptyset$ do
    // invariant: $\langle X, Y\rangle \in R \subseteq b'(c(R \cup todo))$
    extract $\langle X', Y'\rangle$ from $todo$;
    if $\langle X', Y'\rangle \in c(R \cup todo)$ then skip;
    forall $a \in A$ do
        insert $\langle t^{\#}_a(X'), t^{\#}_a(Y')\rangle$ in $todo$;
    end
    insert $\langle X', Y'\rangle$ in $R$;
end
return $R$;
```

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

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, those transitions that should be fired infinitely often in order to accept an infinite word.

Given a NBW $\mathcal{A} = \langle S, t\rangle$ and $u : A^\omega$ an infinite word, we say that a sequence of states $\chi : S^\omega$ accepts u if the sequence $(T_{u_i}(\chi_i, \chi_{i+1}))_{i \in \mathbb{N}}$ 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 and a finite non-empty word v, 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. 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 ([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 rational, then $L^\$$ is rational.*

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 [23] which make the presentation easier and eventually make it possible to propose our algorithm. We use the *(Büchi) transition monoid* of the NBW $\mathcal{A} = \langle S, t\rangle$ [23] to define the second layer.

Consider the set 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 $\mathscr{M} = 3^{S^2}$ for the set of square matrices over 3 indexed by S; it forms a Kleene algebra [7,19] and in particular a semiring. 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 with 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

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

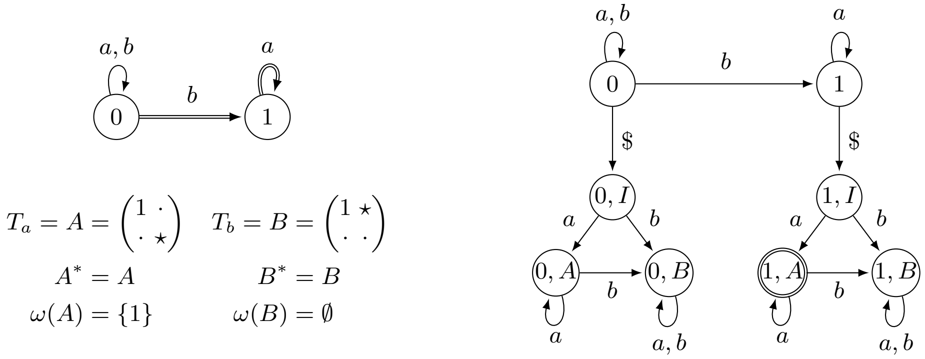
At this point, one can notice that with the previously defined operations, matrices and subsets form the *Wilke algebra* associated to the NBW as in [23].

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

$$\begin{cases} t^\$_a(x) = \{x' \mid T_a(x,x') \neq 0\} \\ t^\$_a(\langle x, M\rangle) = \{\langle x, M \cdot T_a\rangle\} \end{cases} \qquad \begin{cases} t^\$_\$(x) = \{\langle x, I\rangle\} \\ t^\$_\$(\langle x, M\rangle) = \emptyset \end{cases} \qquad \begin{cases} o^\$(x) = 0 \\ o^\$(\langle x, M\rangle) = x \in \omega(M) \end{cases}$$

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



$$T_a = A = \begin{pmatrix} 1 & \cdot \\ \cdot & \star \end{pmatrix} \qquad T_b = B = \begin{pmatrix} 1 & \star \\ \cdot & \cdot \end{pmatrix}$$

$$A^* = A \qquad B^* = B$$

$$\omega(A) = \{1\} \qquad \omega(B) = \emptyset$$

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

(Where I denotes the identity matrix from $\mathscr{M}$.) 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].

Theorem 2. *For all set of states X 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 $[0]^\$_{\mathcal{A}} = (a+b)^*ba^*\$a^+$ 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 relevant 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 could be willing to prune the automaton $\mathcal{A}^\$$ (i.e., remove the four states which are not co-reachable), but we want in the sequel to exploit the structure shared by the copies of the transition monoid: those copies 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].

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: many computations would be duplicated. We show in this section that we can do better.

Given a an ω-language L, the language $L^\$$ can be seen as a weighted language $L^\pounds \colon \mathcal{P}(A^+)^{A^*}$ 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

$$t^\pounds_a(x) = \{x' \mid T_a(x, x') \neq 0\} \qquad\qquad o^\pounds(x) = \{v \mid v^\omega \in [x]_{\mathcal{A}}\}$$

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. We show in a second step, in Sect. 5 how to exploit optimisations *à la* `HK`/`HKC` for this part of the computations.

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

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

Proof. Immediate consequence of Lem. 1 and the definitions of $o^\pounds$ and $o^{\pounds\#}$. (Note that we do not need to impose that v is non-empty in the statement since $\omega(T_\epsilon) = \omega(I) = \emptyset$.)

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

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

Let $\mathcal{D} = \{\omega(T_v) \mid v \in A^*\}$. We call the sets in $\mathcal{D}$ *discriminating sets*; as subsets of S, there are at most $2^{|S|}$ discriminating sets. Those can be enumerated since there are finitely many matrices of the form T_v (at most $3^{|S|^2}$). This is what is done in the algorithm from Fig. 4.

We finally obtain the algorithm in Fig. 5 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 in Fig. 6, starting with states $\{0\}$ and $\{1\}$. The transition monoid has 13 elements, which we list in App. A. Those matrices give rise to three discriminating sets: $\emptyset$, $\{0, 1\}$, and $\{0, 1, 2\}$. Those arise, for instance, from the matrices

$$T_b = \begin{pmatrix} 1 & \cdot & 1 \\ 1 & \cdot & \cdot \\ 1 & \cdot & 1 \end{pmatrix} \qquad\qquad T_a = \begin{pmatrix} \cdot & \star & \cdot \\ \cdot & \star & 1 \\ \cdot & \cdot & \cdot \end{pmatrix} \qquad\qquad T_{ba} = \begin{pmatrix} \cdot & \star & \cdot \\ \cdot & \star & \cdot \\ \cdot & \star & \cdot \end{pmatrix}$$

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

$\mathcal{D} := \emptyset$; $\mathcal{M} := \emptyset$; $todo := \{I\}$;
while $todo \neq \emptyset$ do
    extract $M$ from $todo$;
    if $M \in \mathcal{M}$ then skip;
    forall $a \in A$ do
        insert $M \cdot T_a$ in $todo$;
    end
    insert $M$ in $\mathcal{M}$; insert $\omega(M)$ in $\mathcal{D}$;
end
return $\mathcal{D}$;
```

Fig. 4. Discr($\mathcal{A}$): exploring the Büchi 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 $R :=$ HKC'$(\mathcal{A}^{\pounds}, X, Y)$     ||     $\mathcal{D} :=$ 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 end
5 return $true$;
```

Fig. 5. HKC$^\omega(\mathcal{A}, X, Y)$: checking language equivalence in a NBW using bisimulations up to congruence.

where we denoted 0 by $\cdot$ for clarity purposes. 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. The two pairs of R cannot be told apart using the three discriminating sets so that HKC$^\omega$ returns *true*. States 0 and 1 are indeed equivalent: both of them accept the words with infinitely many a's.

If instead we run HKC$^\omega$ starting from sets $\{0\}$ and $\{2\}$ then it returns *false* since the discriminating set $\{0,1\}$ distinguishes $\{0\}$ and $\{2\}$. Indeed, the state 2 recognises the words with infinitely many a's and starting with a b.

Note that 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 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

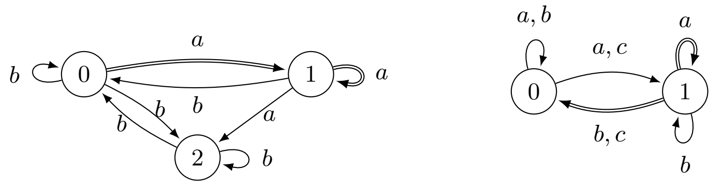


Fig. 6. The NBWs for Ex. 2 and Ex.3.

inserted in $\mathcal{D}$ and R. Alternatively, those checks can be delegated to a SAT solver. Indeed, given a discriminating set D, define the following formula with $2|S|$ variables:

$$\varphi_D = \bigvee_{x \in D} x_1 \;\Leftrightarrow\; \bigvee_{x \in D} x_2$$

For all sets X_1, X_2, we have $X_1 \cap D = \emptyset \Leftrightarrow Y_2 \cap D = \emptyset$ iff φ_D evaluates to *true* under the assignment $x_i \mapsto x \in X_i$. Given the set of discriminating sets $\mathcal{D}$, it thus suffices to build the formula $\varphi_{\mathcal{D}} = \bigwedge_{D \in \mathcal{D}} \varphi_D$ 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:

$$\psi_{\langle X,Y \rangle} = \bigvee_{x \in X} x \;\Leftrightarrow\; \bigvee_{y \in Y} y$$

For all set D, we have $X \cap D = \emptyset \Leftrightarrow Y \cap D = \emptyset$ iff $\psi_{\langle X,Y \rangle}$ evaluates to *true* under the assignment $z \mapsto z \in D$. 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. 5, we can replace the sub-algorithm `Discr` (Fig. 4) 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:

Lemma 3. *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 order-preserving 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 $x_0 \in \mathcal{X}$ and assume x, x' are the least elements of $\mathcal{X}$ such that $x_0 \leq x \leq r(x)$ and $x_0 \leq x' \leq r(f(x'))$, respectively. Then we have $s(x) = s(x')$.*

We prove this lemma in App. B. It is inspired by the abstract theory of coinduction up-to [24], where the *compatibility* condition $f \circ r \leq r \circ f$ plays a central role.

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

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

With such parameters, the x from statement of the lemma is the set of matrices $\mathcal{M}$ obtained at the end of the execution of `Discr`. Accordingly, $d(x)$ is the returned set $\mathcal{D}$ of discriminating sets, and $s(x)$ is the equivalence relation $\sim_{\mathcal{D}}$.

Assume a function f satisfying the other requirements of the lemma for those parameters. This function can be used as an up-to technique, in order to skip elements from the transition monoid: we can obtain an algorithm $\texttt{Discr}_f$ by replacing line 4 from `Discr` (Fig. 4) with

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

This algorithm terminates with a set $\mathcal{M}'$ 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.

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 $\mathcal{E} \subseteq \mathcal{P}(S)$, if $D, D' \in \mathcal{E}$ then $\sim_{\{D \cup D'\} \cup \mathcal{E}} = \sim_{\mathcal{E}}$.

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 completely wrong, because the discriminating set of a sum is in general *not* the union of the underlying discriminating sets. For instance, we have

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

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 an antichain of vectors, i.e, a non-empty set of vectors which are pairwise incomparable, where vectors are ordered pointwise using the order $0 < 1 < \star$. We write $\mathscr{M}'$ for the set of generalised matrices. Given a non-empty set $\mathcal{M}$ of matrices, write $\lfloor \mathcal{M} \rfloor$ for the antichain where only maximal elements from $\mathcal{M}$ are kept. $\mathscr{M}'$ forms an idempotent semiring by setting

$$\mathbf{M} \oplus \mathbf{N} = \lfloor \mathbf{M} \cup \mathbf{N} \rfloor \quad 0 = \{0\} \quad \mathbf{M} \cdot \mathbf{N} = \lfloor \{V \cdot W \mid V \in \mathbf{M},\ W \in \mathbf{N}\} \rfloor \quad 1 = \{I\}$$

Given a matrix M, write $\underline{M}$ for the generalised matrix $\lfloor \{V \in \mathscr{V} \mid V \leq M\} \rfloor$. 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. Moreover, the counter-example above shows that in general, $\underline{M + N} \neq \underline{M} \oplus \underline{N}$.

For $\mathbf{M} \in \mathscr{M}'$, set $\omega(\mathbf{M}) = \bigcup_{V \in \mathbf{M}} \omega(V)$; this function is a homomorphism of semilattices:

Lemma 4. *For all* $\mathbf{M}, \mathbf{N} \in \mathscr{M}'$ *and* $M, N \in \mathscr{M}$*, we have*

$$(i)\ \omega(\mathbf{M} \oplus \mathbf{N}) = \omega(\mathbf{M}) \cup \omega(\mathbf{N}) \quad (ii)\ \underline{M \cdot N} = \underline{M} \cdot \underline{N} \quad (iii)\ \omega(\underline{M}) = \omega(M)$$

Now lift the functions r, d, s we defined after Lem. 3 to sets of generalised matrices: set $r(W) = \{\mathbf{M} \mid \forall a \in A,\ \mathbf{M} \cdot \underline{T_a} \in W\}$, $d(W) = \{\omega(\mathbf{M}) \mid \mathbf{M} \in W\}$, and $s(W) = \sim_{d(W)}$. Finally define the up-to technique as the following function $u \colon \mathcal{P}(\mathscr{M}') \to \mathcal{P}(\mathscr{M}')$:

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

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

Proposition 6. *The functions* r, u *and* s *satisfy the requirements of Lem. 3.*

Proof. Compatibility of u w.r.t. r ($u \circ r \leq r \circ u$) follows from distributivity of $\cdot$ over $\oplus$. The function u is obviously extensive and idempotent. $s \circ u \leq s$ follows from Lem. 4(i) and the observation at the beginning of Sect. 5.1. □

The semiring of generalised matrices is not convenient to use in practice: many matrices expand into generalised matrices of exponential size (e.g. dense matrices). However, we use this semiring only to establish the correctness of the optimisation: thanks to Lem 4(ii), 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':

Proposition 7. *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*-complete.*

Proof. We prove hardness in App. D. 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 $V \in M$. From this fact we can easily derive a boolean formula which allows to delegate the refined check to a SAT solver. □

Example 3. When running this refined version of $\mathtt{HKC}^\omega$ on the NBW on the right in Fig. 6, 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. We give the details of this computation in App. C.

5.2 Working up to equivalence

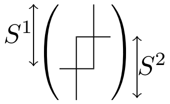
There is also room for improvement when we start with a disjoint union of NBWs: the starting NBWs most probably contain loops, and the transition monoid of the disjoint union will need to unfold those loops until they 'synchronise'. Take for instance the two NBWs $\mathcal{A}^1$ and $\mathcal{A}^2$ over a single letter a, defined by the two matrices on the left, whose disjoint $\mathcal{A}$ union can be represented by the diagonal block matrix on the right:

$$T^1_a \begin{pmatrix} \cdot & \star \\ \star & \cdot \end{pmatrix} \qquad T^2_a = \begin{pmatrix} \cdot & \star & \cdot \\ \cdot & \cdot & \star \\ \star & \cdot & \cdot \end{pmatrix} \qquad T_a = \begin{pmatrix} \cdot & \star & & & \\ \star & \cdot & & & \\ & & \cdot & \star & \cdot \\ & & \cdot & \cdot & \star \\ & & \star & \cdot & \cdot \end{pmatrix}$$

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

$$\forall x, y \in S,\ M(x,y) = 0 \ \vee\ \langle x, y\rangle \in (S^1 \times S^1) \cup (S^2 \times S^2)$$



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.

For $i = 1, 2$, let $\mathscr{M}^i$ be the set of matrices indexed by S^i and let $\pi_i : \mathscr{M}_d \to \mathscr{M}^i$ 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 the following function $e' : \mathcal{P}(\mathscr{M}_d) \to \mathcal{P}(\mathscr{M}_d)$:

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

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, by working in a larger structure than sets of matrices and by using Lem 3, one can show 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. 4. We give more details on this proof in App. E. As in Hopcroft and Karp's algorithm [16], one can implement the up-to equivalence test efficiently using an appropriate union-find data structure.

6 Conclusion and future work

We presented an algorithm for checking language equivalence of non-deterministic Büchi automata. This algorithm exploits advanced coinductive techniques, at two places: first, to analyse the finite prefixes of the considered languages, through bisimulations up to congruence, as in the algorithm HKC for NFA; and second, to analyse the periodic words of the considered languages, through coinduction up to unions (Sect. 5.1) or coinduction up to equivalence (Sect. 5.2). For this second part, 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.

We also want to investigate how to exploit techniques using simulation relations, which were successfully used in [10,1,2,22] and which tend to nicely fit in the coinductive framework we exploit here [4, Sect. 5].

The algorithm we proposed stems from the construction of Calbrix et al. [6], which we revisited using notions from [20] in Sect. 3. $\mathtt{HKC}^\omega$ is rather close to *Ramsey-based* algorithms [11,1] (as opposed to *rank-based* ones [21,8,9,10]). 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 [20].

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 *LNCS*, 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 *LNCS*, 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 *LNCS*, 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 *LNCS*, 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 *LNCS*, pages 451–465. Springer, 2007.
9. L. Doyen and J. Raskin. Antichains for the automata-based approach to model-checking. *LMCS*, 5(1), 2009.
10. L. Doyen and J.-F. Raskin. Antichain Algorithms for Finite Automata. In *Proc. TACAS*, volume 6015 of *LNCS*. Springer, 2010.
11. S. Fogarty and M. Y. Vardi. Büchi complementation and size-change termination. In *Proc. TACAS*, volume 5505 of *LNCS*, pages 16–30. Springer, 2009.
12. S. Fogarty and M. Y. Vardi. Efficient büchi universality checking. In *Proc. TACAS*, volume 6015 of *LNCS*, 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 and R. M. Karp. A linear algorithm for testing equivalence of finite automata. Technical Report 114, Cornell Univ., December 1971.
17. M. Hutagalung, M. Lange, and E. Lozes. Revealing vs. concealing: More simulation games for büchi inclusion. In *Proc. LATA*, pages 347–358. Springer, 2013.
18. B. Knaster. Un théorème sur les fonctions d'ensembles. *Annales de la Société Polonaise de Mathématiques*, 6:133–134, 1928.
19. D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Inf. and Comp.*, 110(2):366–390, 1994.
20. D. Kuperberg, L. Pinault, and D. Pous. Web appendix for this paper, 2018.
21. O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3):408–429, 2001.

22. R. Mayr and L. Clemente. Advanced automata minimization. In *Proc. POPL, 2013*, pages 63–74. ACM, 2013.
23. 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.
24. D. Pous. Coinduction all the way up. In *Proc. LICS*, pages 307–316. ACM, 2016.
25. R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, 1975.
26. A. Tarski. A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific Journal of Mathematics*, 5(2):285–309, June 1955.
27. 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.
28. M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for concurrency*, pages 238–266. Springer, 1996.
29. 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 *LNCS*, pages 17–30. Springer, 2006.

A Further details for Example 2

The transition monoid of the NBW on the left in Fig. 6 is given below, together with the discriminating sets associated to its elements:

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

Equations

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

B Proof of Lemma 3

Proof. Since $id \leq f$, we have $x \leq r(x) \leq r(f(x))$ and thus $x' \leq x$ by minimality of x'. Since $f \circ r \leq r \circ f$ and $f \circ f \leq f$, we also have $f(x') \leq f(r(f(x'))) \leq r(f(f(x'))) \leq r(f(x'))$, 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 $s \circ f \leq s$. □

C Further details for Example 3

The explored part of the transition monoid of the NBW on the right in Fig. 6 is given below, together with the discriminating sets associated to its elements and the justification for the elements skipped thanks to the up-to-union technique:

Kept matrices

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

Equations

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

Unions

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

Accessible matrices not generated

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

To alleviate notations, we identified matrices M with their associated generalised matrix $\underline{M}$ in the third column. (Note that $T_{ca} = T_{bc} + T_{ccc}$ but $\underline{T_{ca}} \neq \underline{T_{bc}} \oplus \underline{T_{ccc}}$).

D CoNP-completeness of reasoning up to union

Theorem 3 (Prop. 7). *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 when we represent $\underline{N}$ and $\underline{M}$ as plain matrices.*

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

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

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:

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

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 problem is indeed coNP-Hard

E Reasoning up to equivalence: proofs

In order to reason up to equivalence, 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 $\mathcal{E} \in \mathit{Rel}(U)$, 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 $X \subseteq S^1$ and $Y \subseteq S^2$, `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 $\cdot \colon \mathscr{M}'' \times \mathscr{M}_d \to \mathscr{M}''$ by setting:

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

Now lift the functions r, s we defined after Lem. 3 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 8. *The functions r, e and s satisfy the requirements of Lem. 3.*

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 $\langle X, Y\rangle \in s(R)$, i.e., $X \approx_{d(R)} Y$, we have to show $\langle X, Y\rangle \in s(e(R))$, i.e., $X \approx_{d(e(R))} Y$. 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 $X \approx_{d(R)} Y$, 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 $\underline{M} = \langle 1\pi_1(M), 2\pi_2(M)\rangle \in \mathscr{M}''$. For all $M, N \in \mathscr{M}_d$, we have $\underline{M \cdot N} = \underline{M} \cdot N$ and for all $X \subseteq S^1$ $Y \subseteq S^2$,

$$X \sim_{\omega(M)} Y \quad \text{iff} \quad X \approx_{d(\underline{M})} Y$$

The first property guarantees that when taking $x_0 = \{\underline{I}\}$, the x from Lem. 3 is the set $\{\underline{M} \mid M \in \mathcal{M}\}$ where $\mathcal{M}$ is the set of matrices computed by `Discr`. The second property ensures that $s(x)$ properly discriminates the pairs provided by `HKC`' (R). By Lem. 3, 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.