Coinductive algorithms for Büchi automata

Denis Kuperberg   Laureline Pinault   Damien Pous

LIP, Ens de Lyon

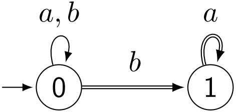
DLT, Warsaw
August 5th 2019

Introduction and motivations

<u>Büchi automata (NBWs):</u>

$\langle S, T \rangle$ with $T : A \to 3^{S^2}$
$\mathcal{L}(\mathcal{A}) : A^\omega \to 2$

<u>Example:</u>



$\mathcal{L} = (a+b)^* b a^\omega$

<u>Problem:</u>
$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$?
PSPACE-Complete

<u>Modelize:</u>

- Programs and sytems
- Specifications of systems (eg. LTL formulae)

<u>Application:</u>
Verification of systems

Past works

Existing algorithms

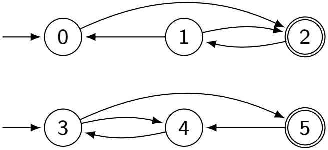
Problem	NFAs	NBWs
Inclusion	Antichain-based Algorithms [Wulf,Doyen,Henzinger,Raskin '06] [Abdulla,Chen,Holik,et al '11] [Doyen,Raskin '10] Tools : Powerset construction and subsumption techniques	Antichain-based Algorithms [Fogarty,Vardi '09 '10] [Doyen,Raskin '09 '10] [Abdulla,Chen,Clemente,et al '10 '11] Tools : Ranked-based or Ramsey-based complementation
Equality	Coinduction-based Algorithm `HKC` [Bonchi,Pous '13] Tools : Powerset construction and up-to techniques	**?**

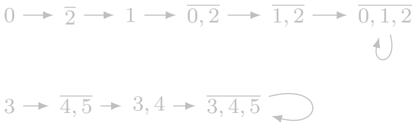
HKC [Bonchi,Pous '13]

Principle: Try to compute a bisimulation up to congruence.

Input: A NFA $\mathcal{A}$ and two sets of states X, Y.
Output: $true$ if X and Y recognize the same language; $false$ otherwise.

- Explore the powerset construction on the fly
- Examine pairs of sets of states
  - Check if the accepting condtions match, if not **return** $false$
  - If the pair is in the congruence closure of already seen pairs then skip it
  - Add successors to the list of pairs to check
- If no more pair to look at, **return** $true$



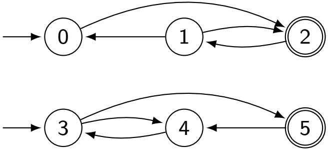


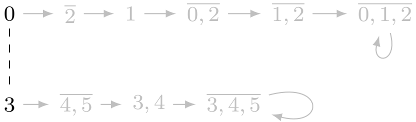
HKC [Bonchi,Pous '13]

Principle: Try to compute a bisimulation up to congruence.

Input: A NFA $\mathcal{A}$ and two sets of states X, Y.
Output: $true$ if X and Y recognize the same language; $false$ otherwise.

- Explore the powerset construction on the fly
- Examine pairs of sets of states
  - Check if the accepting condtions match, if not **return** $false$
  - If the pair is in the congruence closure of already seen pairs then skip it
  - Add successors to the list of pairs to check
- If no more pair to look at, **return** $true$

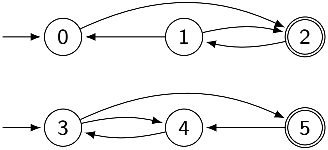




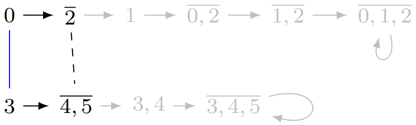
HKC [Bonchi,Pous '13]

Principle: Try to compute a bisimulation up to congruence.

Input: A NFA $\mathcal{A}$ and two sets of states X, Y.
Output: $true$ if X and Y recognize the same language; $false$ otherwise.



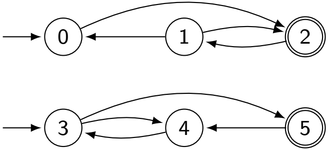
- Explore the powerset construction on the fly
- Examine pairs of sets of states
  - Check if the accepting condtions match, if not **return** $false$
  - If the pair is in the congruence closure of already seen pairs then skip it
  - Add successors to the list of pairs to check
- If no more pair to look at, **return** $true$



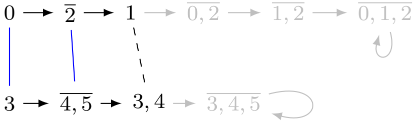
HKC [Bonchi,Pous '13]

Principle: Try to compute a bisimulation up to congruence.

Input: A NFA $\mathcal{A}$ and two sets of states X, Y.
Output: $true$ if X and Y recognize the same language; $false$ otherwise.



- Explore the powerset construction on the fly
- Examine pairs of sets of states
  - Check if the accepting condtions match, if not **return** $false$
  - If the pair is in the congruence closure of already seen pairs then skip it
  - Add successors to the list of pairs to check
- If no more pair to look at, **return** $true$

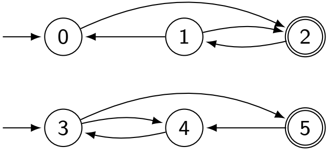


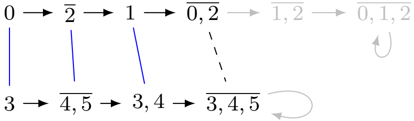
HKC [Bonchi,Pous '13]

Principle: Try to compute a bisimulation up to congruence.

Input: A NFA $\mathcal{A}$ and two sets of states X, Y.
Output: $true$ if X and Y recognize the same language; $false$ otherwise.

- Explore the powerset construction on the fly
- Examine pairs of sets of states
  - Check if the accepting condtions match, if not **return** $false$
  - If the pair is in the congruence closure of already seen pairs then skip it
  - Add successors to the list of pairs to check
- If no more pair to look at, **return** $true$



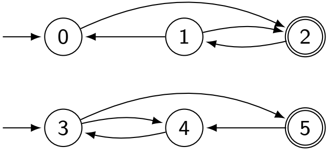


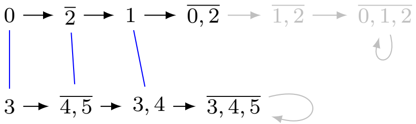
HKC [Bonchi,Pous '13]

Principle: Try to compute a bisimulation up to congruence.

Input: A NFA $\mathcal{A}$ and two sets of states X, Y.
Output: $true$ if X and Y recognize the same language; $false$ otherwise.

- Explore the powerset construction on the fly
- Examine pairs of sets of states
  - Check if the accepting condtions match, if not **return** $false$
  - If the pair is in the congruence closure of already seen pairs then skip it
  - Add successors to the list of pairs to check
- If no more pair to look at, **return** $true$





Difficulties for adapting HKC to NBWs

- Non local acceptation conditions

- No proper determinization operation

Ultimately periodic words [Calbrix, Nivat, Podelski '93]

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

Fact 1

If $\mathcal{L}_1$ and $\mathcal{L}_2$ are ω-regular then $\mathcal{L}_1 = \mathcal{L}_2$ if and only if $UP(\mathcal{L}_1) = UP(\mathcal{L}_2)$

- **Proof** Closure properties of rational ω-languages + non-empty languages contain at least one ultimately periodic word.

Fact 2

If $\mathcal{L}$ is ω-regular then $\mathcal{L}^\$$ is regular.

- **Proof** Construction of Calbrix, Nivat and Podelski.

This paper

Big picture

ω-regular		ultimately periodic		Fact 2 regular
$\mathcal{L} \subseteq A^\omega$ $\mathcal{L}_1 = \mathcal{L}_2$	$\longrightarrow$ $\Leftrightarrow$ Fact 1	$UP(\mathcal{L}) \subseteq A^\omega$ $UP(\mathcal{L}_1) = UP(\mathcal{L}_2)$	$\longrightarrow$ $\Leftrightarrow$	$\mathcal{L}^\$ \subseteq (A \cup \{\$\})^*$ $\mathcal{L}_1^\$ = \mathcal{L}_2^\$$

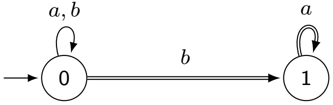
NBW		NFA
$\mathcal{A}$ $\lvert\mathcal{A}\rvert = n$ $[x]_\mathcal{A} = [y]_\mathcal{A}$	(revisited) construction of Calbrix, Nivat and Podelski $\longrightarrow$ $\Leftrightarrow$	$\mathcal{A}^\$$ $\lvert\mathcal{A}^\$\rvert = n + n3^{n^2}$ $[x]_{\mathcal{A}^\$} = [y]_{\mathcal{A}^\$}$

$\Rightarrow$ `HKC` on $\mathcal{A}^\$$ + Exploiting the structure of the construction

Construction of $\mathcal{A}^{\$}$

Consider a NBW $\mathcal{A}$ with n states.

- Construct the (Büchi) transition monoïd of $\mathcal{A}$
- Compute the loop structure
- Construct the prefix layer
- Add a $\$$ transition from each state
- Copy n times the transition monoid
- Define the final states



$\mathcal{L}(\mathcal{A}) = (a+b)^* b a^\omega$

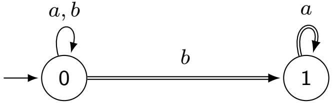
Construction of $\mathcal{A}^\$$

Consider a NBW $\mathcal{A}$ with n states.

- Construct the (Büchi) transition monoïd of $\mathcal{A}$
- Compute the loop structure
- Construct the prefix layer
- Add a $\$$ transition from each state
- Copy n times the transition monoid
- Define the final states

$$T_v(x,y) = \begin{cases} 0 & \text{if } x \overset{v}{\not\to} y \\ 1 & \text{if } x \xrightarrow{v} y \\ \star & \text{if } x \overset{v}{\Rightarrow} y \end{cases}$$

$$\begin{tikzcd}
 & T_b = \begin{pmatrix} 1 & 0 \\ 0 & \star \end{pmatrix} \arrow[loop right, "a"] \arrow[d, "b"] \\
I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \arrow[ur, "a"] \arrow[r, "b"'] & T_b = \begin{pmatrix} 1 & \star \\ 0 & 0 \end{pmatrix} \arrow[loop right, "a,b"]
\end{tikzcd}$$



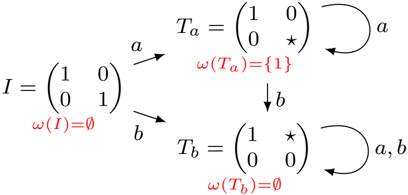
$\mathcal{L}(\mathcal{A}) = (a+b)^* b a^\omega$

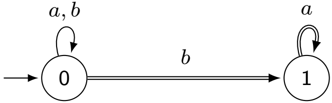
Construction of $\mathcal{A}^{\$}$

Consider a NBW $\mathcal{A}$ with n states.

- Construct the (Büchi) transition monoïd of $\mathcal{A}$
- Compute the loop structure
- Construct the prefix layer
- Add a $\$$ transition from each state
- Copy n times the transition monoid
- Define the final states

$$x \in \omega(T_v) \text{ iff } v^\omega \in [x]_{\mathcal{A}}$$
$$\text{iff } x \xrightarrow{v^k} y \overset{v^{k'}}{\Longrightarrow} y$$



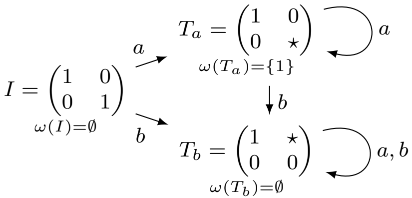


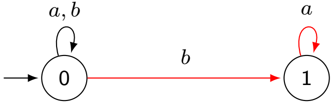
$$\mathcal{L}(\mathcal{A}) = (a+b)^* b a^\omega$$

Construction of $\mathcal{A}^{\$}$

Consider a NBW $\mathcal{A}$ with n states.

- Construct the (Büchi) transition monoïd of $\mathcal{A}$
- Compute the loop structure
- Construct the prefix layer
- Add a $\$$ transition from each state
- Copy n times the transition monoid
- Define the final states



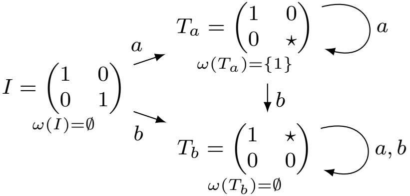


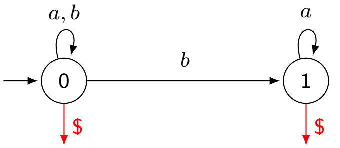
$\mathcal{A}$ without accepting conditions

Construction of $\mathcal{A}^{\$}$

Consider a NBW $\mathcal{A}$ with n states.

- Construct the (Büchi) transition monoïd of $\mathcal{A}$
- Compute the loop structure
- Construct the prefix layer
- Add a \$ transition from each state
- Copy n times the transition monoid
- Define the final states

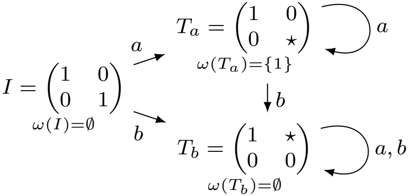


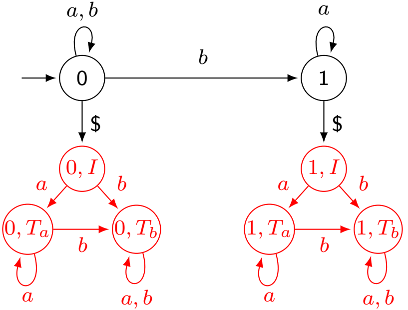


Construction of $\mathcal{A}^{\$}$

Consider a NBW $\mathcal{A}$ with n states.

- Construct the (Büchi) transition monoïd of $\mathcal{A}$
- Compute the loop structure
- Construct the prefix layer
- Add a $\$$ transition from each state
- Copy n times the transition monoid
- Define the final states



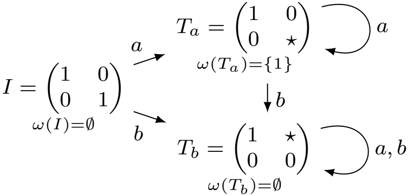


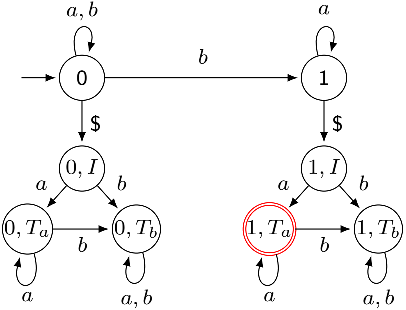
Construction of $\mathcal{A}^{\$}$

Consider a NBW $\mathcal{A}$ with n states.

- Construct the (Büchi) transition monoïd of $\mathcal{A}$
- Compute the loop structure
- Construct the prefix layer
- Add a $\$$ transition from each state
- Copy n times the transition monoid
- Define the final states

(x, T_v) is accepting iff $x \in \omega(T_v)$

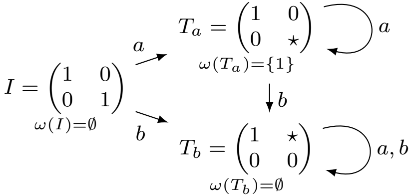


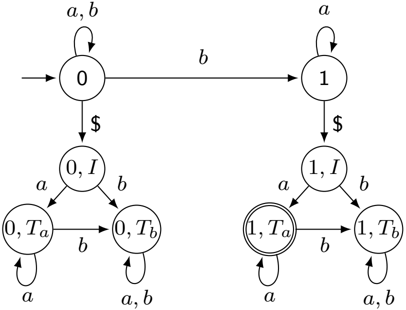


Construction of $\mathcal{A}^{\$}$

Consider a NBW $\mathcal{A}$ with n states.

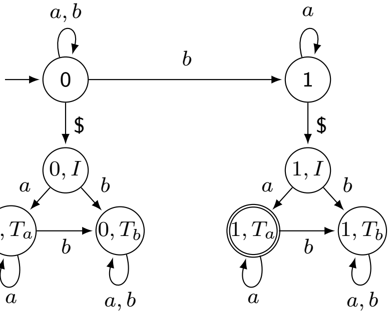
- Construct the (Büchi) transition monoïd of $\mathcal{A}$
- Compute the loop structure
- Construct the prefix layer
- Add a $\$$ transition from each state
- Copy n times the transition monoid
- Define the final states





$\mathcal{L}(\mathcal{A}^{\$}) = (a+b)^* b a^* \$ a^+$

Exploiting the structure of the construction



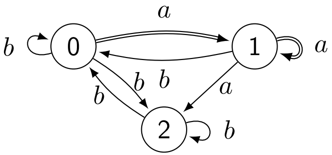
$$\mathcal{L}^{\$} = \{u\$v \mid uv^\omega \in \mathcal{L}\}$$

- Second layer deterministic $\Rightarrow$ no need for congruence on it.
- Second layer consists in n times the same structure, only final states change $\Rightarrow$ want to share the computation.
- First layer can be seen as a weighted automaton $\mathcal{A}^{\pounds}$ recognizing $\mathcal{L}^{\pounds} : u \mapsto \{v \mid uv^\omega \in \mathcal{L}\}$ $\Rightarrow$ apply a variant of `HKC` on it.

HKC' on $\mathcal{A}^{\pounds}$

Changes: Postpone the verifications and return list of non-skipped pairs.

Input: A NBW $\mathcal{A}$ and two sets of states X, Y.
Output: A pre-bisimulation up to congruence.



$[0]_{\mathcal{A}} = [1]_{\mathcal{A}} =$ infinitely many a's

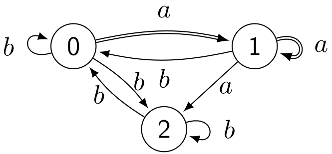
Pairs: $\langle\{0\},\{1\}\rangle$; $\langle\{1\},\{1,2\}\rangle$

Skipped pairs: $\langle\{0,2\},\{0\}\rangle$; $\langle\{1,2\},\{1,2\}\rangle$; $\langle\{0\},\{0,2\}\rangle$

HKC' on $\mathcal{A}^{\pounds}$

Changes: Postpone the verifications and return list of non-skipped pairs.

Input: A NBW $\mathcal{A}$ and two sets of states X, Y.
Output: A pre-bisimulation up to congruence.



$[0]_{\mathcal{A}} = [1]_{\mathcal{A}} =$ infinitely many a's

Pairs: $\langle\{0\},\{1\}\rangle$; $\langle\{1\},\{1,2\}\rangle$

In the congruence closure:

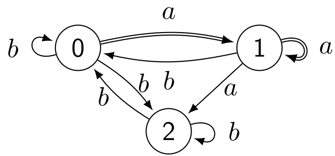
- $\langle\{0\},\{1,2\}\rangle$ by transitvity
- $\langle\{2\},\{2\}\rangle$ by reflexivity
- $\langle\{0,2\},\{1,2\}\rangle$ by union
- $\langle\{1,2\},\{1\}\rangle$ and $\langle\{1\},\{0\}\rangle$ by symmetry
- $\langle\{0,2\},\{1\}\rangle$ by transitvity
- $\langle\{0,2\},\{0\}\rangle$ by transitvity

Skipped pairs: $\langle\{0,2\},\{0\}\rangle$; $\langle\{1,2\},\{1,2\}\rangle$; $\langle\{0\},\{0,2\}\rangle$

Comparing the outputs: discriminating sets

A priori need to compare language of words $\Rightarrow$ infinitely many of them.
But All the information needed in the set of $\omega(T_v)$ $\Rightarrow$ finite.

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



- Go through the transition monoïd
- Keep in memory the different $\omega(T_v)$'s encoutered

Transition monoid has 13 elements

$\mathcal{D} = \emptyset \; ; \; \{0,1\} \; ; \; \{0,1,2\}$

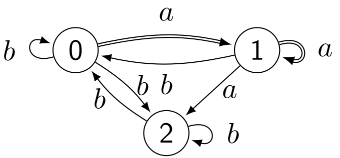
$$\underset{\omega(T_b) = \emptyset}{\begin{pmatrix} 1 & \cdot & 1 \\ 1 & \cdot & \cdot \\ 1 & \cdot & 1 \end{pmatrix}} \quad \underset{\omega(T_a) = \{0,1\}}{\begin{pmatrix} \cdot & \star & \cdot \\ \cdot & \star & 1 \\ \cdot & \cdot & \cdot \end{pmatrix}} \quad \underset{\omega(T_{ba}) = \{0,1,2\}}{\begin{pmatrix} \cdot & \star & \cdot \\ \cdot & \star & \cdot \\ \cdot & \star & \cdot \end{pmatrix}}$$

Global Algorithm

Compute the Pairs || Compute the discriminating sets $\mathcal{D}$

For all pair $\langle X, Y \rangle$ and all discriminating set D, check that:

$$X \cap D = \emptyset \Leftrightarrow Y \cap D = \emptyset$$



$\rightarrow$ returns $true$

Pairs $= \langle \{0\}, \{1\} \rangle \ ; \ \langle \{1\}, \{1, 2\} \rangle$

$\mathcal{D} = \emptyset \ ; \ \{0, 1\} \ ; \ \{0, 1, 2\}$

Conclusion

A coinductive-based algorithm to solve the language equality problem for Büchi automata.

Prototype implementation avaible at
`http://perso.ens-lyon.fr/damien.pous/covece/hkcw/`.

Advantages:

- Two independant and parallel parts
- Advanced up-to techniques to study the prefix layer
- Sharing of information to study the periodic layer

Drawback:

- Need to explore the whole transition monoïd to find discriminating sets

Future work

- Design up-to techniques to take advantage of the structure of the periodic layer
- Adapt known optmization for already existing algorithms on NBW