On testing the equality of sofic systems

Eugen Czeizler and Jarkko Kari

University of Turku, Turku Centre for Computer Science

Definitions

Given a finite automaton $\mathcal{A} = (Q, \Sigma, \delta)$

- $w \in \Sigma^*$ is a finite word,
- $u \in {}^{\omega}\Sigma$, $v \in \Sigma^{\omega}$ and $w \in {}^{\omega}\Sigma^{\omega}$ are left, right and bi-infinite words,

Definitions

Given a finite automaton $\mathcal{A} = (Q, \Sigma, \delta)$

- $w \in \Sigma^*$ is a finite word,
- $u \in {}^{\omega}\Sigma$, $v \in \Sigma^{\omega}$ and $w \in {}^{\omega}\Sigma^{\omega}$ are left, right and bi-infinite words,
- $(q_i)_{1 \le i \le k+1}$ is a finite path in $\mathcal{A}$ labeled by $w = (a_i)_{1 \le i \le k}$ iff $q_{i+1} \in \delta(q_i, a_i)$,
- $(q_i)_{i \le 0}$, $(q_i)_{i \ge 0}$ and $(q_i)_{i \in Z}$ are left, right and bi-infinite paths labeled by $(a_i)_{i<0}$, $(a_i)_{i \ge 0}$ and $(a_i)_{i \in Z}$ iff $q_{i+1} \in \delta(q_i, a_i)$.

Local Automata

A finite automaton $\mathcal{A} = (Q, \Sigma, \delta)$ is called *local* if there exist values *m* and *k*, $0 \leq m \leq k$, such that any two equally labeled paths of length *k* go through the same state at time *m*.

Local Automata

A finite automaton $\mathcal{A} = (Q, \Sigma, \delta)$ is called *local* if there exist values *m* and *k*, $0 \leq m \leq k$, such that any two equally labeled paths of length *k* go through the same state at time *m*.

The smallest value for *k* is called the synchronization delay of $\mathcal{A}$.

$\omega\omega$Automata

We say that a finite automaton $\mathcal{A} = (Q, \Sigma, \delta)$ is

$\omega\omega$Automata

We say that a finite automaton $\mathcal{A} = (Q, \Sigma, \delta)$ is

- $^{\omega\omega}injective$ iff for any bi-infinite word $w \in {}^{\omega}\Sigma^{\omega}$ there exists at most one path labeled by *w*;

$^{\omega\omega}$Automata

We say that a finite automaton $\mathcal{A} = (Q, \Sigma, \delta)$ is

- $^{\omega\omega}$*injective* iff for any bi-infinite word $w \in {}^{\omega}\Sigma^{\omega}$ there exists at most one path labeled by *w*;
- $^{\omega\omega}$*surjective* iff for any bi-infinite word $w \in {}^{\omega}\Sigma^{\omega}$ there exists at least one path labeled by *w*;

$^{\omega\omega}$Automata

We say that a finite automaton $\mathcal{A} = (Q, \Sigma, \delta)$ is

- $^{\omega\omega}$*injective* iff for any bi-infinite word $w \in {}^{\omega}\Sigma^{\omega}$ there exists at most one path labeled by *w*;
- $^{\omega\omega}$*surjective* iff for any bi-infinite word $w \in {}^{\omega}\Sigma^{\omega}$ there exists at least one path labeled by *w*;
- $^{\omega\omega}$*bijective* iff for any bi-infinite word $w \in {}^{\omega}\Sigma^{\omega}$ there exists exactly one path labeled by *w*;

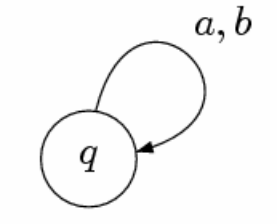
$^{\omega\omega}$Automata

We say that a finite automaton $\mathcal{A} = (Q, \Sigma, \delta)$ is

- $^{\omega\omega}$*injective* iff for any bi-infinite word $w \in {}^{\omega}\Sigma^{\omega}$ there exists at most one path labeled by *w*;
- $^{\omega\omega}$*surjective* iff for any bi-infinite word $w \in {}^{\omega}\Sigma^{\omega}$ there exists at least one path labeled by *w*;
- $^{\omega\omega}$*bijective* iff for any bi-infinite word $w \in {}^{\omega}\Sigma^{\omega}$ there exists exactly one path labeled by *w*;
  - **Obs**. We may restrict ourselves to the non-transient part of the automaton (all states are part of some bi-infinite path)

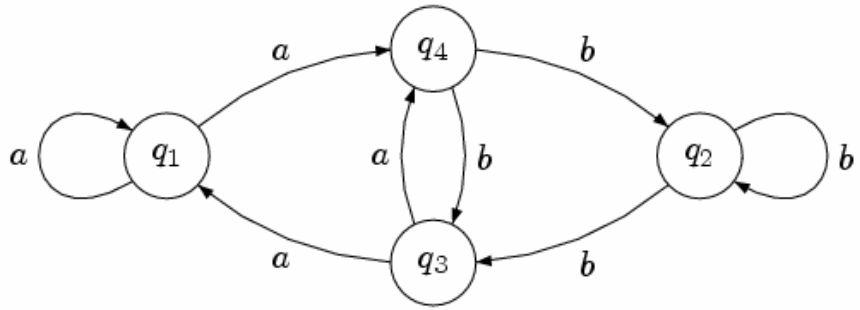
$\omega\omega$Automata

Example:



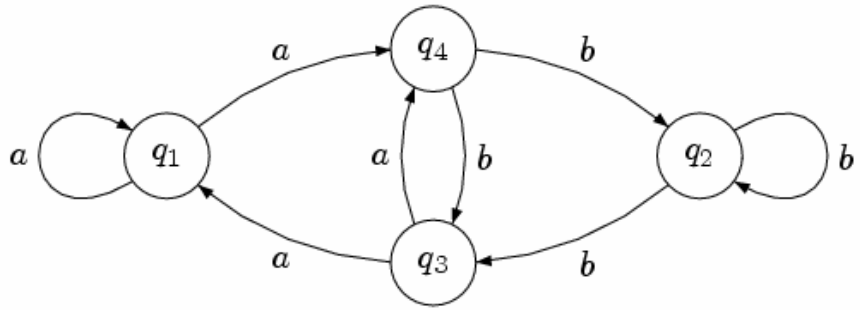
$\omega\omega$Automata

Example:



$\omega\omega$Automata

Example:



An automaton is $\omega\omega$ injective iff it is local. Moreover, the class of $\omega\omega$ bijective automata is strictly included in the class of local automata.

Decision algorithms

Given an automaton $\mathcal{A} = (Q, \Sigma, \delta)$ how efficiently can we decide whether:

- $\mathcal{A}$ is $\omega\omega$ injective
- $\mathcal{A}$ is $\omega\omega$ surjective
- $\mathcal{A}$ is $\omega\omega$ bijective

Decision algorithms

Given an automaton $\mathcal{A} = (Q, \Sigma, \delta)$ how efficiently can we decide whether:

- $\mathcal{A}$ is $\omega\omega$ injective (quadratic time compexity)
- $\mathcal{A}$ is $\omega\omega$ surjective
- $\mathcal{A}$ is $\omega\omega$ bijective

Decision algorithms

Given an automaton $\mathcal{A} = (Q, \Sigma, \delta)$ how efficiently can we decide whether:

- $\mathcal{A}$ is $\omega\omega$ injective (quadratic time compexity)
- $\mathcal{A}$ is $\omega\omega$ surjective (PSPACE-complete)
- $\mathcal{A}$ is $\omega\omega$ bijective

Decision algorithms

Given an automaton $\mathcal{A} = (Q, \Sigma, \delta)$ how efficiently can we decide whether:

- $\mathcal{A}$ is $\omega\omega$ injective (quadratic time compexity)
- $\mathcal{A}$ is $\omega\omega$ surjective (PSPACE-complete)
- $\mathcal{A}$ is $\omega\omega$ bijective (polynomial time complexity)

Decision algorithms

Given an automaton $\mathcal{A} = (Q, \Sigma, \delta)$ how efficiently can we decide whether:

- $\mathcal{A}$ is $\omega\omega$ injective (quadratic time complexity)
- $\mathcal{A}$ is $\omega\omega$ surjective (PSPACE-complete)
- $\mathcal{A}$ is $\omega\omega$ bijective (polynomial time complexity)
  - Given two $\omega\omega$ injective automata, decide whether the two languages of bi-infinite words obtained as labels of bi-infinite paths are equal, i.e. the two automata are $\omega\omega$ equivalent.

Sofic Systems

Sofic Systems

Given Σ an alphabeth, ${}^{\omega}\Sigma^{\omega}$ is called the *full shift*.

Sofic Systems

Given Σ an alphabeth, ${}^{\omega}\Sigma^{\omega}$ is called the *full shift*.

$S \subseteq {}^{\omega}\Sigma^{\omega}$ is a *subshift* iff it is closed and shift invariant.

Sofic Systems

Given Σ an alphabeth, ${}^{\omega}\Sigma^{\omega}$ is called the *full shift*.

$S \subseteq {}^{\omega}\Sigma^{\omega}$ is a *subshift* iff it is closed and shift invariant.

Given a finite automaton $\mathcal{A} = (Q, \Sigma, \delta)$, the set of bi-infinite words obtained as labels of all bi-infinite paths in that automaton, is a subshift.

Sofic Systems

Given Σ an alphabeth, ${}^{\omega}\Sigma^{\omega}$ is called the *full shift*.

$S \subseteq {}^{\omega}\Sigma^{\omega}$ is a *subshift* iff it is closed and shift invariant.

Given a finite automaton $\mathcal{A} = (Q, \Sigma, \delta)$, the set of bi-infinite words obtained as labels of all bi-infinite paths in that automaton, is a subshift.

- Any subshift obtained in this way is called a sofic system, while the automaton generating it is called a presentation.

Sofic Systems

Given Σ an alphabeth, ${}^{\omega}\Sigma^{\omega}$ is called the *full shift*.

$S \subseteq {}^{\omega}\Sigma^{\omega}$ is a *subshift* iff it is closed and shift invariant.

Given a finite automaton $\mathcal{A} = (Q, \Sigma, \delta)$, the set of bi-infinite words obtained as labels of all bi-infinite paths in that automaton, is a subshift.

- Any subshift obtained in this way is called a sofic system, while the automaton generating it is called a presentation.
- **Obs**. A sofic system may have more than one presentation.

Sofic Systems

A subshift S is called of *finite type* iff all bi-infinite words from S avoid a finite number of factors.

Sofic Systems

A subshift S is called of *finite type* iff all bi-infinite words from S avoid a finite number of factors.

A subshift is of finite type iff it has a local presentation

Decision algorithms

Given an automaton $\mathcal{A} = (Q, \Sigma, \delta)$ how efficiently can we decide whether:

- $\mathcal{A}$ is $\omega\omega$ injective (quadratic time complexity)
- $\mathcal{A}$ is $\omega\omega$ surjective (PSPACE-complete)
- $\mathcal{A}$ is $\omega\omega$ bijective (polynomial time complexity)
  - Given two $\omega\omega$ injective automata, decide whether the two languages of bi-infinite words obtained as labels of bi-infinite paths are equal, i.e. the two automata are $\omega\omega$ equivalent.

Decision algorithms

Given an automaton $\mathcal{A} = (Q, \Sigma, \delta)$ how efficiently can we decide whether:

- $\mathcal{A}$ is $\omega\omega$ injective (quadratic time complexity)
- $\mathcal{A}$ is $\omega\omega$ surjective (PSPACE-complete)
- $\mathcal{A}$ is $\omega\omega$ bijective (polynomial time complexity)
  - Given two $\omega\omega$ injective automata, decide whether the two languages of bi-infinite words obtained as labels of bi-infinite paths are equal, i.e. the two automata are $\omega\omega$ equivalent.
  - Decide whether two subshifts of finite type are equal, given their presentation as $\omega\omega$ injective (local) automata.

Decision algorithms- $\omega\omega$injectivity

Quadratic in the number of states of the automaton (M.-P. Beal, Codage Symbolique, 1993)

Decision algorithms-$\omega\omega$injectivity

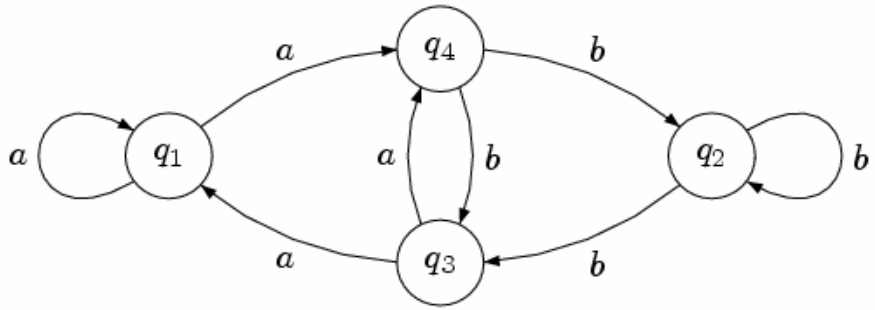
Quadratic in the number of states of the automaton (M.-P. Beal, Codage Symbolique, 1993)

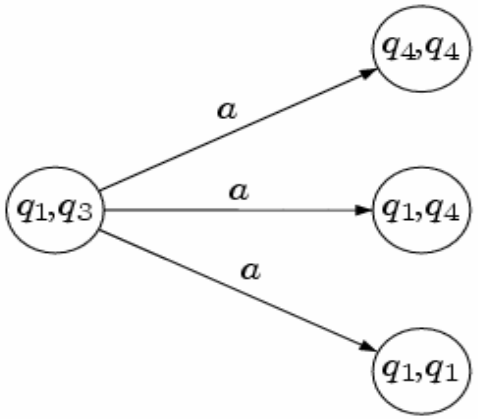
- Construct the pair graph

Decision algorithms- $\omega\omega$injectivity

Quadratic in the number of states of the automaton (M.-P. Beal, Codage Symbolique, 1993)

- Construct the pair graph

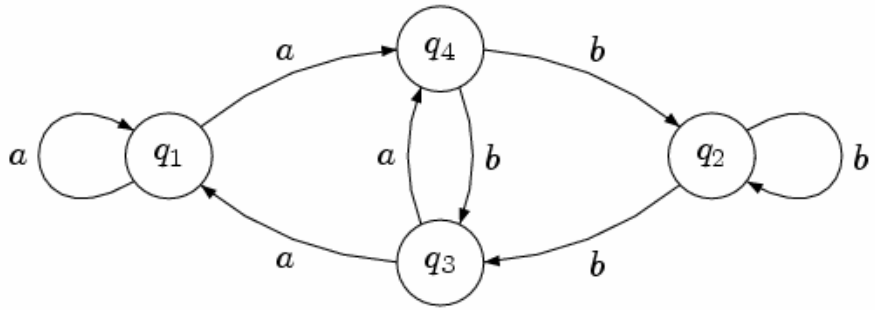


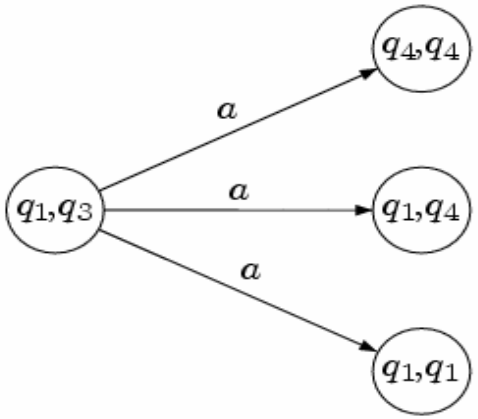


Decision algorithms- $\omega\omega$injectivity

Quadratic in the number of states of the automaton (M.-P. Beal, Codage Symbolique, 1993)

- Construct the pair graph





- Test for the existence of a loop containing a vertex of the form (q_i, q_j) , with $q_i \neq q_j$.

Decision algorithms- $\omega\omega$surjectivity

Testing whether a finite automaton is $\omega\omega$surjective is PSPACE-complete

Decision algorithms- $\omega\omega$surjectivity

Testing whether a finite automaton is $\omega\omega$surjective is PSPACE-complete

- Given a finite automaton $\mathcal{A} = (Q, \Sigma, I, \delta, F)$, we construct $\mathcal{A}' = (Q, \Sigma \cup \{\#\}, \delta')$ such that for all $q_f \in F$ and $q_s \in I$, we add a transition $q_s \in \delta'(q_f, \#)$.

Decision algorithms- $\omega\omega$surjectivity

Testing whether a finite automaton is $\omega\omega$surjective is PSPACE-complete

- Given a finite automaton $\mathcal{A} = (Q, \Sigma, I, \delta, F)$, we construct $\mathcal{A}' = (Q, \Sigma \cup \{\#\}, \delta')$ such that for all $q_f \in F$ and $q_s \in I$, we add a transition $q_s \in \delta'(q_f, \#)$.
  - Claim: $\mathcal{A}$ recognizes Σ^* iff $\mathcal{A}'$ is $\omega\omega$ surjective.

Decision algorithms- $\omega\omega$surjectivity

Testing whether a finite automaton is $\omega\omega$surjective is PSPACE-complete

- Given a finite automaton $\mathcal{A} = (Q, \Sigma, I, \delta, F)$, we construct $\mathcal{A}' = (Q, \Sigma \cup \{\#\}, \delta')$ such that for all $q_f \in F$ and $q_s \in I$, we add a transition $q_s \in \delta'(q_f, \#)$.
  - Claim: $\mathcal{A}$ recognizes Σ^* iff $\mathcal{A}'$ is $\omega\omega$ surjective.

- Testing whether a finite automaton is $\omega\omega$surjective is similar to testing whether it recognize the language Σ^* , when all the states are both initial and final.

Decision algorithms- $\omega\omega$ equivalence

Testing whether two automata are $\omega\omega$ equivalent can be done using a polynomial time algorithm.

Welch sets

Given a finite automaton $\mathcal{A} = (Q, \Sigma, \delta)$

- For any word $w \in \Sigma^{\omega}$ define the *left Welch set* of *w* as the set of states from which a path labeled by *w* can start.

$$L(w) = \{q \in Q \mid (q, q_1, q_2, \ldots) \text{ is a path in } \mathcal{A} \text{ labeled by } w\}$$

Welch sets

Given a finite automaton $\mathcal{A} = (Q, \Sigma, \delta)$

- For any word $w \in \Sigma^{\omega}$ define the *left Welch set* of *w* as the set of states from which a path labeled by *w* can start.

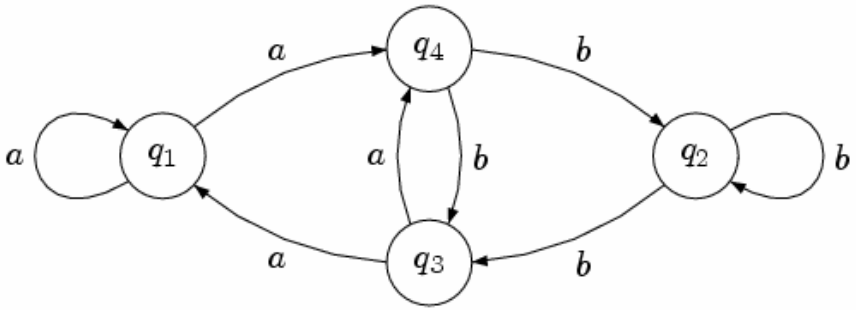
$$L(w) = \{q \in Q \mid (q, q_1, q_2, \ldots) \text{ is a path in } \mathcal{A} \text{ labeled by } w\}$$

- For any word $w \in {}^{\omega}\Sigma$ define the *right Welch set* of *w* as the set of states in which a path labeled by *w* can stop.

$$R(w) = \{q \in Q \mid (\ldots, q_{-2}, q_{-1}, q) \text{ is a path in } \mathcal{A} \text{ labeled by } w\}$$

Welch sets

Example:

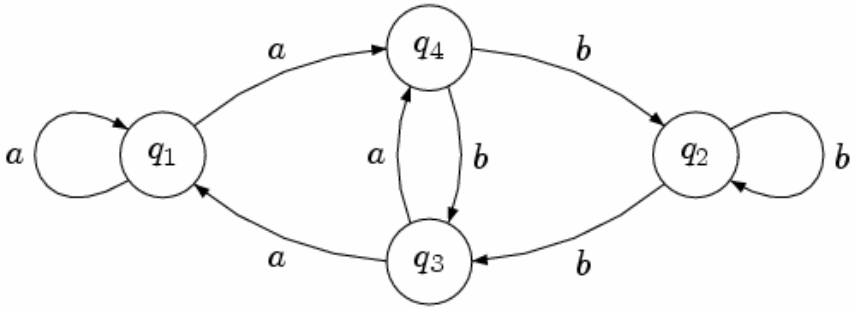


$$L(a^{\omega}) = \{q_1, q_3\}$$

$$R({}^{\omega}b) = \{q_2, q_3\}$$

Welch sets

Example:

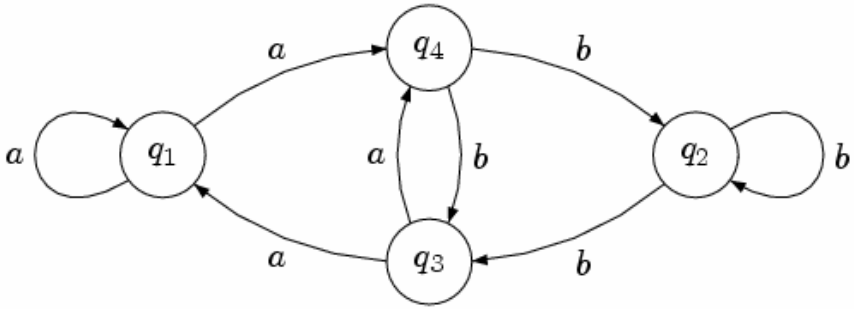


$$L(a^{\omega}) = \{q_1, q_3\} \qquad \vec{L} = (1, 0, 1, 0)$$

$$R({}^{\omega}b) = \{q_2, q_3\} \qquad \vec{R} = (0, 1, 1, 0)$$

Welch sets

Example:



$$L(a^{\omega}) = \{q_1, q_3\} \qquad \vec{L} = (1,0,1,0)$$

$$R({}^{\omega}b) = \{q_2, q_3\} \qquad \vec{R} = (0,1,1,0)$$

- Given an $\omega\omega$injective automaton $\mathcal{A}$, if $\vec{L}_1$ *and* $\vec{R}_1$ *are two left/right Welch vectors, any other Welch vector can be obtained by applying some suitable linear functions.*

Extended Welch vectors

Given two finite autonata $\mathcal{A} = (Q, \Sigma, \delta)$ and $\mathcal{A}' = (Q', \Sigma, \delta')$

- For any word $w \in \Sigma^{\omega}$, based on the two left Welch sets *L* and *L'* corresponding to *w* on both automata, we define the *extended left Welch vector* of *w* as:

$$\vec{L}^{cc}(w) = \left(-\vec{L}, \vec{L}'\right)$$

Extended Welch vectors

Given two finite autonata $\mathcal{A} = (Q, \Sigma, \delta)$ and $\mathcal{A}' = (Q', \Sigma, \delta')$

- For any word $w \in \Sigma^{\omega}$, based on the two left Welch sets *L* and *L'* corresponding to *w* on both automata, we define the *extended left Welch vector* of *w* as:

$$\vec{L}^{cc}(w) = \left(-\vec{L}, \vec{L}'\right)$$

- For any word $w \in {}^{\omega}\Sigma$, based on the two right Welch sets *R* and *R'* corresponding to *w* on both automata, we define the *extended right Welch vector* of *w* as:

$$\vec{R}^{cc}(w) = \left(\vec{R}, \vec{R}'\right)$$

Decision algorithms- $\omega\omega$ equivalence

Two $\omega\omega$ injective automata are $\omega\omega$ equivalent iff for every $w \in \Sigma^{\omega}$ and $v \in {}^{\omega}\Sigma$

$$\vec{L}^{cc}(w) \cdot \vec{R}^{cc}(v) = 0$$

Decision algorithms- $\omega\omega$ equivalence

Two $\omega\omega$ injective automata are $\omega\omega$ equivalent iff for every $w \in \Sigma^{\omega}$ and $v \in {}^{\omega}\Sigma$

$$\vec{L}^{cc}(w) \cdot \vec{R}^{cc}(v) = 0$$

We would like to test the above property only on some "representative" Welch vectors, i.e., a set of linear independent left/right Welch vectors.

Decision algorithms- $\omega\omega$equivalence

Given two $\omega\omega$ injective automata $\mathcal{A}$ and $\mathcal{A}'$ with n, respectively n' states, if $\{\vec{L}_1^{cc}, \ldots, \vec{L}_{n+n'}^{cc}\}$ is a <u>well selected</u> set of $n+n'$ extended left Welch vectors, then:

Decision algorithms- $\omega\omega$equivalence

Given two $\omega\omega$ injective automata $\mathcal{A}$ and $\mathcal{A}'$ with *n*, respectively *n'* states, if $\{\vec{L}_1^{cc}, \ldots, \vec{L}_{n+n'}^{cc}\}$ is a well selected set of *n+n'* extended left Welch vectors, then:

- either *any other extended left Welch vector can be obtained by applying some suitable linear functions,*

Decision algorithms- $\omega\omega$equivalence

Given two $\omega\omega$ injective automata $\mathcal{A}$ and $\mathcal{A}'$ with *n*, respectively *n'* states, if $\{\vec{L}_1^{cc}, \ldots, \vec{L}_{n+n'}^{cc}\}$ is a <u>well selected</u> set of *n+n'* extended left Welch vectors, then:

- <u>either</u> *any other extended left Welch vector can be obtained by applying some suitable linear functions,*
- <u>*or*</u> *there exists* $u \in \Sigma^*$ *and* $1 \leq i \leq n+n'$ *such that* $h_u(\vec{L}_i^{cc}) = (-\vec{L}, \vec{0})$ *or* $h_u(\vec{L}_i^{cc}) = (\vec{0}, \vec{L}')$.

($\mathcal{A}$ *and* $\mathcal{A}'$ *are not* $\omega\omega$ *equivalent) .*

Decision algorithms- $\omega\omega$equivalence

Given two $\omega\omega$ injective automata $\mathcal{A}$ and $\mathcal{A}'$ with *n*, respectively *n'* states, if $\{\vec{L}_1^{cc}, \ldots, \vec{L}_{n+n'}^{cc}\}$ is a <u>well selected</u> set of *n+n'* extended left Welch vectors, then:

- <u>either</u> *any other extended left Welch vector can be obtained by applying some suitable linear functions,*
- <u>*or*</u> *there exists* $u \in \Sigma^*$ *and* $1 \leq i \leq n+n'$ *such that* $h_u(\vec{L}_i^{cc}) = (-\vec{L}, \vec{0})$ *or* $h_u(\vec{L}_i^{cc}) = (\vec{0}, \vec{L}')$.

($\mathcal{A}$ *and* $\mathcal{A}'$ *are not* $\omega\omega$ *equivalent) .*

Based on these *n+n'* vectors we compute a basis $\{\vec{L}_1^{cc}, \ldots, \vec{L}_k^{cc}\}$ of the vector space:

$$\langle \{h_u(\vec{L}_i^{cc}) \mid u \in \Sigma^*,\ 1 \leq i \leq n+n'\} \rangle$$

Decision algorithms- $\omega\omega$equivalence

Similarly, we obtain a basis $\{\vec{R}_1^{cc}, \ldots, \vec{R}_l^{cc}\}$ for

$$\langle \{ g_u(\vec{R}_i^{cc}) \mid u \in \Sigma^*,\ 1 \leq i \leq n + n' \} \rangle$$

Decision algorithms- $\omega\omega$equivalence

Similarly, we obtain a basis $\{\vec{R}_1^{cc}, \ldots, \vec{R}_l^{cc}\}$ for

$$\langle \{ g_u(\vec{R}_i^{cc}) \mid u \in \Sigma^*,\ 1 \leq i \leq n + n' \} \rangle$$

The two $\omega\omega$injective automata are $\omega\omega$equivalent iff for all $1 \leq i \leq k$ and $1 \leq j \leq l$, $\vec{L}_i^{cc} \cdot \vec{R}_j^{cc} = 0$.

Decision algorithms- $\omega\omega$equivalence

The Algorithm:

Decision algorithms- $\omega\omega$equivalence

The Algorithm:

- Compute the sets $\{\vec{L}_1^{cc}, \ldots, \vec{L}_{n+n'}^{cc}\}$ and $\{\vec{R}_1^{cc}, \ldots, \vec{R}_{n+n'}^{cc}\}$;

Decision algorithms- $\omega\omega$equivalence

The Algorithm:

- Compute the sets $\{\vec{L}_1^{cc}, \ldots, \vec{L}_{n+n'}^{cc}\}$ and $\{\vec{R}_1^{cc}, \ldots, \vec{R}_{n+n'}^{cc}\}$;
- Compute the basses $\{\vec{L}_1^{cc}, \ldots, \vec{L}_k^{cc}\}$ and $\{\vec{R}_1^{cc}, \ldots, \vec{R}_l^{cc}\}$;

Decision algorithms- $\omega\omega$equivalence

The Algorithm:

- Compute the sets $\{\vec{L}_1^{cc}, \ldots, \vec{L}_{n+n'}^{cc}\}$ and $\{\vec{R}_1^{cc}, \ldots, \vec{R}_{n+n'}^{cc}\}$;
- Compute the basses $\{\vec{L}_1^{cc}, \ldots, \vec{L}_k^{cc}\}$ and $\{\vec{R}_1^{cc}, \ldots, \vec{R}_l^{cc}\}$;
- Test whether for all $1 \leq i \leq k$ and $1 \leq j \leq l$, $\vec{L}_i^{cc} \cdot \vec{R}_j^{cc} = 0$.

Open Problem

Open Problem

- Decide whether two sofic systems are equal, given their presentation as deterministic automata. (polynomial time complexity – N. Jonoska)

Open Problem

- Decide whether two sofic systems are equal, given their presentation as deterministic automata. (polynomial time complexity – N. Jonoska)

- Decide whether two subshifts of finite type are equal, given their presentation as $^{\omega\omega}$injective automata. (polynomial time complexity – here)

Open Problem

- Decide whether two sofic systems are equal, given their presentation as deterministic automata. (polynomial time complexity – N. Jonoska)

- Decide whether two subshifts of finite type are equal, given their presentation as $^{\omega\omega}$injective automata. (polynomial time complexity – here)

- Does there exists a polynomial time algorithm deciding whether two subshifts of finite type are equal, given an arbitrary presentations for these subshifts, i.e two (not necessarily local) finite automata?

Thank you !