Synchronizing Data Words for Register Automata

Parvaneh Babari ¹, Karin Quaas ¹, Mahsa Shirmohammadi ²

¹ Universität Leipzig, ² University of Oxford

MFCS 2016
\(\Sigma\) ... a finite alphabet
\(D\) ... an infinite data domain, e.g., \(\mathbb{N}, \mathbb{R}_{\geq 0},\) ASCII strings

A data word is a finite sequence

\[(a_1, d_1)(a_2, d_2)\ldots(a_n, d_n)\]

over \(\Sigma \times D\).

A data language is a subset of \((\Sigma \times D)^*\).
Register Automata (RA)

\[
\begin{align*}
0 & \rightarrow 1 \quad a, = r_1 \quad a, = r_1 \vee = r_2 \\
0 & \rightarrow 1' \quad a, \neq r_1, r_1 \downarrow \\
0 & \rightarrow 2 \quad a, \neq r_1 \vee \neq r_2, r_1 \downarrow, r_2 \downarrow \\
0 & \rightarrow 2' \quad a, \neq r_1, r_2 \downarrow \\
1 & \rightarrow 1 \quad a, = r_1 \\
1 & \rightarrow 2 \quad a, \neq r_1, r_2 \downarrow \\
1 & \rightarrow 3 \quad a, r_1 \downarrow, r_2 \downarrow \\
1' & \rightarrow 1' \quad a, = r_1 \\
1' & \rightarrow 2' \quad a, \neq r_1, r_2 \downarrow \\
1' & \rightarrow 3 \quad a, r_1 \downarrow, r_2 \downarrow \\
2 & \rightarrow 2 \quad a, \neq r_1 \vee \neq r_2, r_1 \downarrow, r_2 \downarrow \\
2 & \rightarrow 3 \quad a, r_1 \downarrow, r_2 \downarrow \\
2' & \rightarrow 2' \quad a, = r_1 \vee = r_2 \\
2' & \rightarrow 3 \quad a, r_1 \downarrow, r_2 \downarrow \\
3 & \rightarrow 3 \quad a, r_1 \downarrow, r_2 \downarrow
\end{align*}
\]
Register Automata (RA)

$w = (a, d_1)(a, d_2)(a, d_1)(a, d_3)$

$(0, d_1, d_1) \xrightarrow{(a, d_1)} (1, d_1, d_1) \xrightarrow{(a, d_2)} (2, d_1, d_2) \xrightarrow{(a, d_1)} (2, d_1, d_2) \xrightarrow{(a, d_3)} (3, d_3, d_3)$
Register Automata (RA)

\[ w = (a, d_1)(a, d_2)(a, d_1)(a, d_3) \]

\[
(0, d_1, d_1) \xrightarrow{(a,d_1)} (1, d_1, d_1) \xrightarrow{(a,d_2)} (2, d_1, d_2) \xrightarrow{(a,d_1)} (2, d_1, d_2) \xrightarrow{(a,d_3)} (3, d_3, d_3)
\]
Register Automata (RA)

\[ w = (a, d_1)(a, d_2)(a, d_1)(a, d_3) \]

\[ (0, d_1, d_1) \xrightarrow{(a,d_1)} (1, d_1, d_1) \xrightarrow{(a,d_2)} (2, d_1, d_2) \xrightarrow{(a,d_1)} (2, d_1, d_2) \xrightarrow{(a,d_3)} (3, d_3, d_3) \]
Register Automata (RA)

State transitions:

\[ w = (a, \text{ } d_1)(a, \text{ } d_2)(a, d_1)(a, d_3) \]

\[ (0, d_1, d_1) \overset{(a,d_1)}\rightarrow (1, d_1, d_1) \overset{(a,d_2)}\rightarrow (2, d_1, d_2) \overset{(a,d_1)}\rightarrow (2, d_1, d_2) \overset{(a,d_3)}\rightarrow (3, d_3, d_3) \]
Register Automata (RA)

\[ w = (a, d_1)(a, d_2)(a, d_1)(a, d_3) \]

\[ (0, d_1, d_1) \xrightarrow{(a, d_1)} (1, d_1, d_1) \xrightarrow{(a, d_2)} (2, d_1, d_2) \xrightarrow{(a, d_1)} (2, d_1, d_2) \xrightarrow{(a, d_3)} (3, d_3, d_3) \]
Register Automata (RA)

\[ w = (a, d_1)(a, d_2)(a, d_1)(a, d_3) \]

\[(0, d_1, d_1) \xrightarrow{(a,d_1)} (1, d_1, d_1) \xrightarrow{(a,d_2)} (2, d_1, d_2) \xrightarrow{(a,d_1)} (2, d_1, d_2) \xrightarrow{(a,d_3)} (3, d_3, d_3) \]
Register Automata (RA)

\[ w = (a, d_1)(a, d_2)(a, d_1)(a, d_3) \]

\[ (0, d_1, d_1) \xrightarrow{(a,d_1)} (1, d_1, d_1) \xrightarrow{(a,d_2)} (2, d_1, d_2) \xrightarrow{(a,d_1)} (2, d_1, d_2) \xrightarrow{(a,d_3)} (3, d_3, d_3) \]
Facts about Register Automata

The Non-Emptiness Problem
Instance: A register automaton $A$.
Question: Does $A$ accept a data word?
The Non-Emptiness Problem
Instance: A register automaton $A$.
Question: Does $A$ accept a data word?

- PSPACE-complete (Demri & Lazic, 2008)
Facts about Register Automata

The Non-Emptiness Problem
Instance: A register automaton \( A \).
Question: Does \( A \) accept a data word?

▷ PSPACE-complete (Demri & Lazic, 2008)

The Non-Universality Problem
Instance: A register automaton \( A \).
Question: Does there exist some data word that \( A \) does not accept?
Facts about Register Automata

The Non-Emptiness Problem
Instance: A register automaton $\mathcal{A}$.
Question: Does $\mathcal{A}$ accept a data word?

- PSPACE-complete (Demri & Lazic, 2008)

The Non-Universality Problem
Instance: A register automaton $\mathcal{A}$.
Question: Does there exist some data word that $\mathcal{A}$ does not accept?

- PSPACE-complete for deterministic RA (DRA)
  (Demri & Lazic, 2008)
Facts about Register Automata

The Non-Emptiness Problem
Instance: A register automaton \( A \).
Question: Does \( A \) accept a data word?
  ▶ PSPACE-complete (Demri & Lazic, 2008)

The Non-Universality Problem
Instance: A register automaton \( A \).
Question: Does there exist some data word that \( A \) does not accept?
  ▶ PSPACE-complete for deterministic RA (DRA) (Demri & Lazic, 2008)
  ▶ undecidable for non-deterministic RA (NRA) (Neven, Schwentick & Vianu, 2004)
The Non-Emptiness Problem
Instance: A register automaton $A$.
Question: Does $A$ accept a data word?

- PSPACE-complete (Demri & Lazic, 2008)

The Non-Universality Problem
Instance: A register automaton $A$.
Question: Does there exist some data word that $A$ does not accept?

- PSPACE-complete for deterministic RA (DRA) (Demri & Lazic, 2008)
- undecidable for non-deterministic RA (NRA) (Neven, Schwentick & Vianu, 2004)
- decidable for NRA with one register (1-NRA), Ackermann-complete (Demri & Lazic, 2008)
Synchronizing Data Words for Register Automata
Let $\mathcal{A}$ be a RA with $k$ registers.

A data word $w$ is called **synchronizing for** $\mathcal{A}$ if there exists a configuration $(s, v_1, \ldots, v_k)$ such that $\mathcal{A}$ is in $(s, v_1, \ldots, v_k)$ after processing $w$, no matter from which configuration $c$ $\mathcal{A}$ starts from.
Let $\mathcal{A}$ be a RA with $k$ registers.

A data word $w$ is called **synchronizing for** $\mathcal{A}$ if there exists a configuration $(s, v_1, \ldots, v_k)$ such that $\mathcal{A}$ is in $(s, v_1, \ldots, v_k)$ after processing $w$, no matter from which configuration $c$ $\mathcal{A}$ starts from.
The data word $w = (a, d_1)(a, d_2)(a, d_3)(a, d_4)$ is synchronizing for $r \neq r$, $r \downarrow \neq r$, $r \downarrow \neq r$, $r \downarrow \neq r$, $r = r$, $r = r$ because this RA is in $(3, d_4)$ after processing $w$, no matter from which configuration $c \ A$ starts from.
The data word \( w = (a, d_1)(a, d_2)(a, d_3)(a, d_4) \) is synchronizing for 

because this RA is in \((3, d_4)\) after processing \( w \), no matter from which configuration \( c A \) starts from.
The data word $w = (a, d_1)(a, d_2)(a, d_3)(a, d_4)$ is synchronizing for $r \neq r, r \downarrow \neq r, r \downarrow \neq r, r \downarrow \neq r, r \downarrow$ because this RA is in $(3, d_4)$ after processing $w$, no matter from which configuration $c \ A$ starts from.
Example Synchronizing Data Words

The data word \( w = (a, d_1)(a, d_2)(a, d_3)(a, d_4) \) is synchronizing for

because this RA is in \((3, d_4)\) after processing \( w \), no matter from which configuration \( c A \) starts from.
The data word \( w = (a, d_1)(a, d_2)(a, d_3)(a, d_4) \) is synchronizing for \( r \neq r, r \uparrow \neq r, r \downarrow \neq r, r \downarrow \neq r, r \uparrow \neq r, r \downarrow \neq r \) because this RA is in \((3, d_4)\) after processing \( w \), no matter from which configuration \( c \ A \) starts from.
The data word \( w = (a, d_1)(a, d_2)(a, d_3)(a, d_4) \) is synchronizing for

\[
= r \\
\neq r, r \downarrow \\
\neq r, r \downarrow
\]

because this RA is in \((3, d_4)\) after processing \( w \), no matter from which configuration \( c \mathcal{A} \) starts from.
How to prove the Synchronizing Property

\[
\{0, 1, 2, 1', 2', 3\} \times D
\]

\[
\{(1, d_1), (1', d_1), (2, d_1), (3, d_1), (2', d_1)\} \cup \{(2', 3) \times D\setminus\{d_1\}\}
\]

\[
\{(2, d_2), (2', d_1), (3, d_2), (2', d_2), (3, d_1)\} \cup \{(3) \times D\setminus\{d_1, d_2\}\}
\]

\[
\{(3, d_1), (3, d_2), (3, d_3)\}
\]

\[
\{(3, d_4)\}
\]
How to prove the Synchronizing Property

\[
\{0, 1, 2, 1', 2', 3\} \times D
\]

\[
\{(1, d_1), (1', d_1), (2, d_1), (3, d_1), (2', d_1)\} \cup (\{2', 3\} \times D \setminus \{d_1\})
\]

\[
\{(2, d_2), (2', d_1), (3, d_2), (2', d_2), (3, d_1)\} \cup (\{3\} \times D \setminus \{d_1, d_2\})
\]

\[
\{(3, d_1), (3, d_2), (3, d_3)\}
\]

\[
\{(3, d_4)\}
\]
The Synchronizing Problem

Instance: A register automaton $\mathcal{A}$.

Question: Does there exist a synchronizing data word for $\mathcal{A}$?
Some Notes on the Synchronizing Problem

- Synchronizing words for finite automata (FA) (Černy 1964, Pin 1978)
Some Notes on the Synchronizing Problem

- Synchronizing words for finite automata (FA) (Černy 1964, Pin 1978)
- Synchronizing problem for deterministic FA is in PTIME
Some Notes on the Synchronizing Problem

- Synchronizing words for finite automata (FA) (Černy 1964, Pin 1978)
- Synchronizing problem for deterministic FA is in PTIME
- Černy Conjecture: The length of a shortest synchronizing word for deterministic FA with \( n \) states is at most \((n - 1)^2\).
Some Notes on the Synchronizing Problem

- Synchronizing words for finite automata (FA) (Černy 1964, Pin 1978)
- Synchronizing problem for deterministic FA is in PTIME
- Černy Conjecture: The length of a shortest synchronizing word for deterministic FA with $n$ states is at most $(n - 1)^2$.
- Overview article by Volkov, 2008.
Synchronizing Data Words for DRA
Synchronizing Data Words for DRA

The data word \( w = (a, d_1)(a, d_2)(a, d_3)(a, d_4) \) is synchronizing for because \( A \) is in \( (3, d_4) \) after processing \( w \), no matter from which configuration \( c \) \( A \) starts from.
Synchronizing Data Words for DRA

\[
\{0, 1, 2, 1', 2', 3\} \times D
\]

\[
\downarrow (a, d_1)
\]

\[
\{(1, d_1), (1', d_1), (2, d_1), (3, d_1), (2', d_1)\} \cup \{(2', 3) \times D \setminus \{d_1\}\}
\]

\[
\downarrow (a, d_2)
\]

\[
\{(2, d_2), (2', d_1), (3, d_2), (2', d_2), (3, d_1)\} \cup \{(3) \times D \setminus \{d_1, d_2\}\}
\]

\[
\downarrow (a, d_3)
\]

\[
\{(3, d_1), (3, d_2), (3, d_3)\}
\]

\[
\downarrow (a, d_4)
\]

\[
\{(3, d_4)\}
\]
The data word \( w = (a, d_1)(a, d_2)(a, d_3)(a, d_4) \) is synchronizing for

because \( \mathcal{A} \) is in \((3, d_4)\) after processing \( w \), no matter from which configuration \( c \) \( \mathcal{A} \) starts from.
The data word \( w = (a, d_1)(a, d_2)(a, d_3)(a, d_4) \) is synchronizing for \( A \) because \( A \) is in \((3, d_4)\) after processing \( w \), no matter from which configuration \( c \) \( A \) starts from.

The data word \((a, d_1)(a, d_1)(a, d_1)(a, d_2)(a, d_1)(a, d_2)(a, d_1)\) is also synchronizing for \( A \).
\{0, 1, 2, 1', 2', 3\} \times D

\{(1, d_1), (1', d_1), (2, d_1), (3, d_1), (2', d_1)\} \cup (\{2', 3\} \times D \setminus \{d_1\})

\{(1, d_1), (1', d_1), (2, d_1), (3, d_1), (2', d_1)\} \cup (\{3\} \times D \setminus \{d_1\})

\{(1, d_1), (1', d_1), (2, d_1), (3, d_1), (2', d_1)\}

\{(2, d_2), (2', d_1), (3, d_2), (3, d_1)\}

\{(2, d_2), (2', d_1), (3, d_2), (3, d_1)\}

\{(3, d_1), (2', d_1)\}

\{(3, d_1), (2', d_1)\}

\{(3, d_2), (3, d_1)\}

\{(3, d_2), (3, d_1)\}

\{(3, d_1)\}

\{(3, d_1)\}
Lemma

If for a $k$-DRA $A$ there exists some synchronizing data word, then there exists also some synchronizing data word for $A$ with at most $2k + 1$ distinct data values.
Lemma
If for a $k$-DRA $A$ there exists some synchronizing data word, then there exists also some synchronizing data word for $A$ with at most $2k + 1$ distinct data values.

Theorem
The synchronizing problem for DRA is PSPACE-complete.
Synchronizing Data Words for NRA
Theorem

The synchronizing problem for NRA is undecidable.

Proof Sketch.

By reduction of the non-universality problem for NRA (preserving the number of registers).
Theorem

The synchronizing problem for NRA is undecidable.

Proof Sketch.
By reduction of the non-universality problem for NRA (preserving the number of registers).

Theorem

For 1-NRA the synchronizing problem is Ackermann-complete.

Proof Sketch.
By reduction to the non-universality problem for 1-NRA.
The Length-Bounded Synchronizing Problem
Instance: An NRA $\mathcal{A}$, a natural number $N \in \mathbb{N}$.
Question: Does there exist a synchronizing data word for $\mathcal{A}$ with length at most $N$?
The Length-Bounded Synchronizing Problem
Instance: An NRA $\mathcal{A}$, a natural number $N \in \mathbb{N}$.
Question: Does there exist a synchronizing data word for $\mathcal{A}$ with length at most $N$?

Theorem
The length-bounded synchronizing problem for NRA is $\text{NEXPTIME}$-complete.

Proof Sketch.
The Length-Bounded Synchronizing Problem
Instance: An NRA $\mathcal{A}$, a natural number $N \in \mathbb{N}$.
Question: Does there exist a synchronizing data word for $\mathcal{A}$ with length at most $N$?

Theorem
The length-bounded synchronizing problem for NRA is NEXPTIME-complete.

Proof Sketch.
NEXPTIME-hardness by reduction from the bounded universality problem for regular-like expressions (built from atomic expressions and $\cdot$, $+$, and $^2$).
The synchronizing problem is

- PSPACE-complete for DRA
- undecidable for NRA
- Ackermann-complete for 1-NRA
The synchronizing problem is

- \( \text{PSPACE-complete for DRA} \)
- \( \text{undecidable for NRA} \)
- \( \text{Ackermann-complete for 1-NRA} \)

The length-bounded synchronizing problem for NRA is \( \text{NEXPTIME-complete} \).
The synchronizing problem is

- \( \text{PSPACE-complete for DRA} \)
- \( \text{undecidable for NRA} \)
- \( \text{Ackermann-complete for 1-NRA} \)

The length-bounded synchronizing problem for NRA is \( \text{NEXPTIME-complete} \).

Open: Precise complexity for the length-bounded synchronizing problem for DRA.