Towards Regular Data Languages

Henrik Björklund and Thomas Schwentick

September 30, 2006

Outline

1. **Introduction** (data languages, class-memory automata (CMA), register automata)
2. **Expressiveness**
   - Register automata are strictly weaker than CMAs
   - Deterministic CMAs are incomparable to $FO^2(+1, <, \sim)$
   - CMAs w. reset are strictly stronger than CMAs
3. **Emptiness problem**
   - 2-way deterministic CMAs are undecidable
   - CMA w. reset are decidable
4. **Model checking** (word problem)

Introduction

Data Languages

Let Σ be a finite alphabet, and Δ an infinite data set.

Then any $L \subseteq (\Sigma \times \Delta)^*$ is a data language.

Motivations

Introduction

Data Languages

Let Σ be a finite alphabet, and Δ an infinite data set.

Then any $L \subseteq (\Sigma \times \Delta)^*$ is a data language.

Motivations

XML databases. Attribute values often belong to an infinite domain.

Introduction

Data Languages

Let Σ be a finite alphabet, and Δ an infinite data set.

Then any $L \subseteq (\Sigma \times \Delta)^*$ is a data language.

Motivations

XML databases. Attribute values often belong to an infinite domain.

Parameterized verification. If a protocol is parameterized by the number of processes involved, process IDs can be seen as values from an infinite domain.

Introduction

Data Languages

Let Σ be a finite alphabet, and Δ an infinite data set.

Then any $L \subseteq (\Sigma \times \Delta)^*$ is a data language.

Motivations

XML databases. Attribute values often belong to an infinite domain.

Parameterized verification. If a protocol is parameterized by the number of processes involved, process IDs can be seen as values from an infinite domain.

Regular model checking defines regular sets of global states, and checks reachability w.r.t. length-preserving transducers.

Introduction

Data Languages

Let Σ be a finite alphabet, and Δ an infinite data set.

Then any $L \subseteq (\Sigma \times \Delta)^*$ is a data language.

Motivations

XML databases. Attribute values often belong to an infinite domain.

Parameterized verification. If a protocol is parameterized by the number of processes involved, process IDs can be seen as values from an infinite domain.

Regular model checking defines regular sets of global states, and checks reachability w.r.t. length-preserving transducers.

When looking at computations as sequences of actions, we can instead check that the global sequence and the sequences belonging to individual processes fulfill regular properties.

Introduction

Data Languages

Let Σ be a finite alphabet, and Δ an infinite data set.

Then any $L \subseteq (\Sigma \times \Delta)^*$ is a data language.

Motivations

XML databases. Attribute values often belong to an infinite domain.

Parameterized verification. If a protocol is parameterized by the number of processes involved, process IDs can be seen as values from an infinite domain.

Regular model checking defines regular sets of global states, and checks reachability w.r.t. length-preserving transducers.

When looking at computations as sequences of actions, we can instead check that the global sequence and the sequences belonging to individual processes fulfill regular properties.

In other words, given regular expressions r_1 and r_2, we look at languages of the type $L(r_1) \cap L(r_2)^{\otimes}$.

Class-Memory Automata

A class-memory automaton (CMA) is a tuple $(Q, \Sigma, \Delta, \delta, q_I, F_1, F_2)$, where

1. $\delta : (Q \times \Sigma \times (Q \cup \{\perp\})) \rightarrow 2^Q$ is the transition function,
2. F_1 is the set of globally accepting states, and
3. F_2 is the set of locally accepting states.

Class-Memory Automata

A class-memory automaton (CMA) is a tuple $(Q, \Sigma, \Delta, \delta, q_I, F_1, F_2)$, where

1. $\delta : (Q \times \Sigma \times (Q \cup \{\perp\})) \to 2^Q$ is the transition function,
2. F_1 is the set of globally accepting states, and
3. F_2 is the set of locally accepting states.

A configuration is a pair (q, f), where $q \in Q$ and $f : \Delta \to (Q \cup \{\perp\})$.

Class-Memory Automata

A class-memory automaton (CMA) is a tuple $(Q, \Sigma, \Delta, \delta, q_I, F_1, F_2)$, where

1. $\delta : (Q \times \Sigma \times (Q \cup \{\perp\})) \rightarrow 2^Q$ is the transition function,
2. F_1 is the set of globally accepting states, and
3. F_2 is the set of locally accepting states.

A configuration is a pair (q, f), where $q \in Q$ and $f : \Delta \rightarrow (Q \cup \{\perp\})$.

The automaton can go from (q, f) to (q', f') when reading (a, d) if

1. $f(d) = q''$ and $q' \in \delta(q, a, q'')$,
2. $f'(d) = q'$, and
3. $f'(d') = f(d)$ for all $d' \neq d$.

Class-Memory Automata

A class-memory automaton (CMA) is a tuple $(Q, \Sigma, \Delta, \delta, q_I, F_1, F_2)$, where

1. $\delta : (Q \times \Sigma \times (Q \cup \{\perp\})) \rightarrow 2^Q$ is the transition function,
2. F_1 is the set of globally accepting states, and
3. F_2 is the set of locally accepting states.

A configuration is a pair (q, f), where $q \in Q$ and $f : \Delta \rightarrow (Q \cup \{\perp\})$.

The automaton can go from (q, f) to (q', f') when reading (a, d) if

1. $f(d) = q''$ and $q' \in \delta(q, a, q'')$,
2. $f'(d) = q'$, and
3. $f'(d') = f(d)$ for all $d' \neq d$.

The automaton accepts a word w if, after reading w, it is in a configuration (q, f) s.t.

1. $q \in F_1$, and
2. $\forall d \in \Delta : f(d) \in F_2 \cup \{\perp\}$.

Example

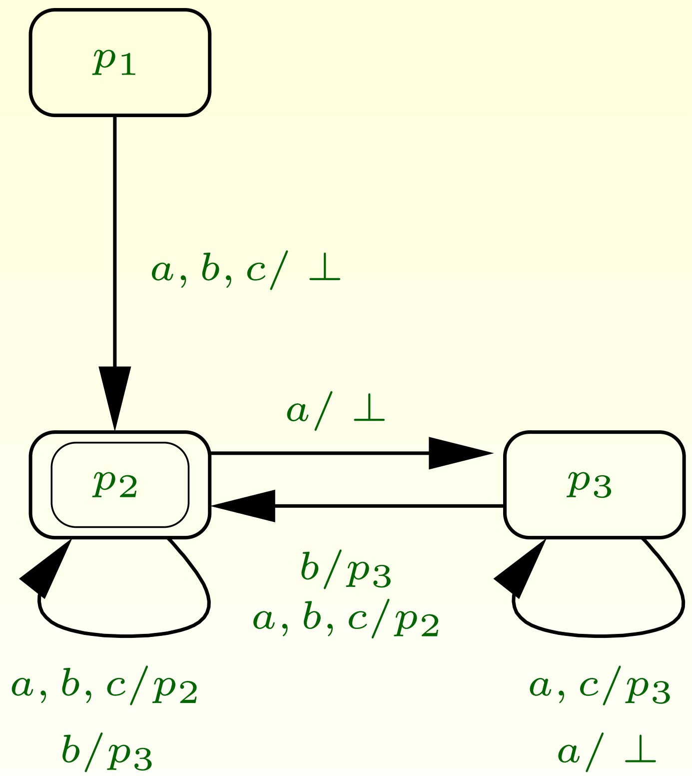
Consider the data language L such that

1. the class of the first position may have any form, and
2. all other classes must begin with an a and contain a b.

Example

Consider the data language L such that

1. the class of the first position may have any form, and
2. all other classes must begin with an a and contain a b.



Data Automata[Bojańczyk, David, Muscholl, Schwentick, Segoufin]

A data automaton D has two parts, the Base automaton A and the class automaton B.

A is a nondeterministic transducer, which reads marked string projections, and writes symbols from a finite alphabet Γ.

B is an NFA, which reads class strings from Γ^*.

Data Automata[Bojańczyk, David, Muscholl, Schwentick, Segoufin]

A data automaton D has two parts, the Base automaton A and the class automaton B.

A is a nondeterministic transducer, which reads marked string projections, and writes symbols from a finite alphabet Γ.

B is an NFA, which reads class strings from Γ^*.

Proposition. Class-memory automata and data automata are equivalent.

Example Languages

Example Languages

1. Data values appear at most once. Check that each class string has length one.

Example Languages

1. Data values appear at most once. Check that each class string has length one.
2. The first and last data values are the same. Mark the first and last position with special signs. Check that they are in the same class string.

Example Languages

1. Data values appear at most once. Check that each class string has length one.
2. The first and last data values are the same. Mark the first and last position with special signs. Check that they are in the same class string.
3. Every class begins with an a and has a b in it. Copy the string projection. Check that all class strings begin with a and contain a b.

Example Languages

1. Data values appear at most once. Check that each class string has length one.
2. The first and last data values are the same. Mark the first and last position with special signs. Check that they are in the same class string.
3. Every class begins with an a and has a b in it. Copy the string projection. Check that all class strings begin with a and contain a b.

A language that isn't recognized:

Between any two occurrences of b, no data value appears more than once.

Example Languages

1. Data values appear at most once. Check that each class string has length one.
2. The first and last data values are the same. Mark the first and last position with special signs. Check that they are in the same class string.
3. Every class begins with an a and has a b in it. Copy the string projection. Check that all class strings begin with a and contain a b.

A language that isn't recognized:

Between any two occurrences of b, no data value appears more than once.

This example also illustrates that the class of recognized languages is not closed under Kleene *.

Register automata [Kaminski & Frances, Neven & Schwentick & Vianu]

A register automaton is a tuple $(Q, \Sigma, \Delta, P, q_I, k, F)$, where

- k is the number of registers, and
- P is a set of transitions of the form $(i, p, a) \to q$ or $(p, a) \to (i, q)$, where $i \in \{1, \dots, k\}$, $a \in \Sigma$ and $p, q \in Q$.

Register automata [Kaminski & Frances, Neven & Schwentick & Vianu]

A register automaton is a tuple $(Q, \Sigma, \Delta, P, q_I, k, F)$, where

- k is the number of registers, and
- P is a set of transitions of the form $(i, p, a) \to q$ or $(p, a) \to (i, q)$, where $i \in \{1, \ldots, k\}$, $a \in \Sigma$ and $p, q \in Q$.

A configuration is a pair (p, f), where $p \in Q$ and $f : \{1, \ldots, k\} \to (\Delta \cup \{\perp\})$ is a register assignment.

Register automata [Kaminski & Frances, Neven & Schwentick & Vianu]

A register automaton is a tuple $(Q, \Sigma, \Delta, P, q_I, k, F)$, where

- k is the number of registers, and
- P is a set of transitions of the form $(i, p, a) \to q$ or $(p, a) \to (i, q)$, where $i \in \{1, \ldots, k\}$, $a \in \Sigma$ and $p, q \in Q$.

A configuration is a pair (p, f), where $p \in Q$ and $f : \{1, \ldots, k\} \to (\Delta \cup \{\perp\})$ is a register assignment.

We can go from (p, f) to (p', f') with transition $(i, a, p) \to p'$ when reading (a, d) if $f(i) = d$.

Register automata [Kaminski & Frances, Neven & Schwentick & Vianu]

A register automaton is a tuple $(Q, \Sigma, \Delta, P, q_I, k, F)$, where

- k is the number of registers, and
- P is a set of transitions of the form $(i, p, a) \to q$ or $(p, a) \to (i, q)$, where $i \in \{1, \ldots, k\}$, $a \in \Sigma$ and $p, q \in Q$.

A configuration is a pair (p, f), where $p \in Q$ and $f : \{1, \ldots, k\} \to (\Delta \cup \{\perp\})$ is a register assignment.

We can go from (p, f) to (p', f') with transition $(i, a, p) \to p'$ when reading (a, d) if $f(i) = d$.

We can go from (p, f) to (p', f') with transition $(p, a) \to (i, p')$ if $f(j) \neq d$, for all i, and $f' = f$ except for $f'(i) = d$.

Register automata [Kaminski & Frances, Neven & Schwentick & Vianu]

A register automaton is a tuple $(Q, \Sigma, \Delta, P, q_I, k, F)$, where

- k is the number of registers, and
- P is a set of transitions of the form $(i, p, a) \to q$ or $(p, a) \to (i, q)$, where $i \in \{1, \ldots, k\}$, $a \in \Sigma$ and $p, q \in Q$.

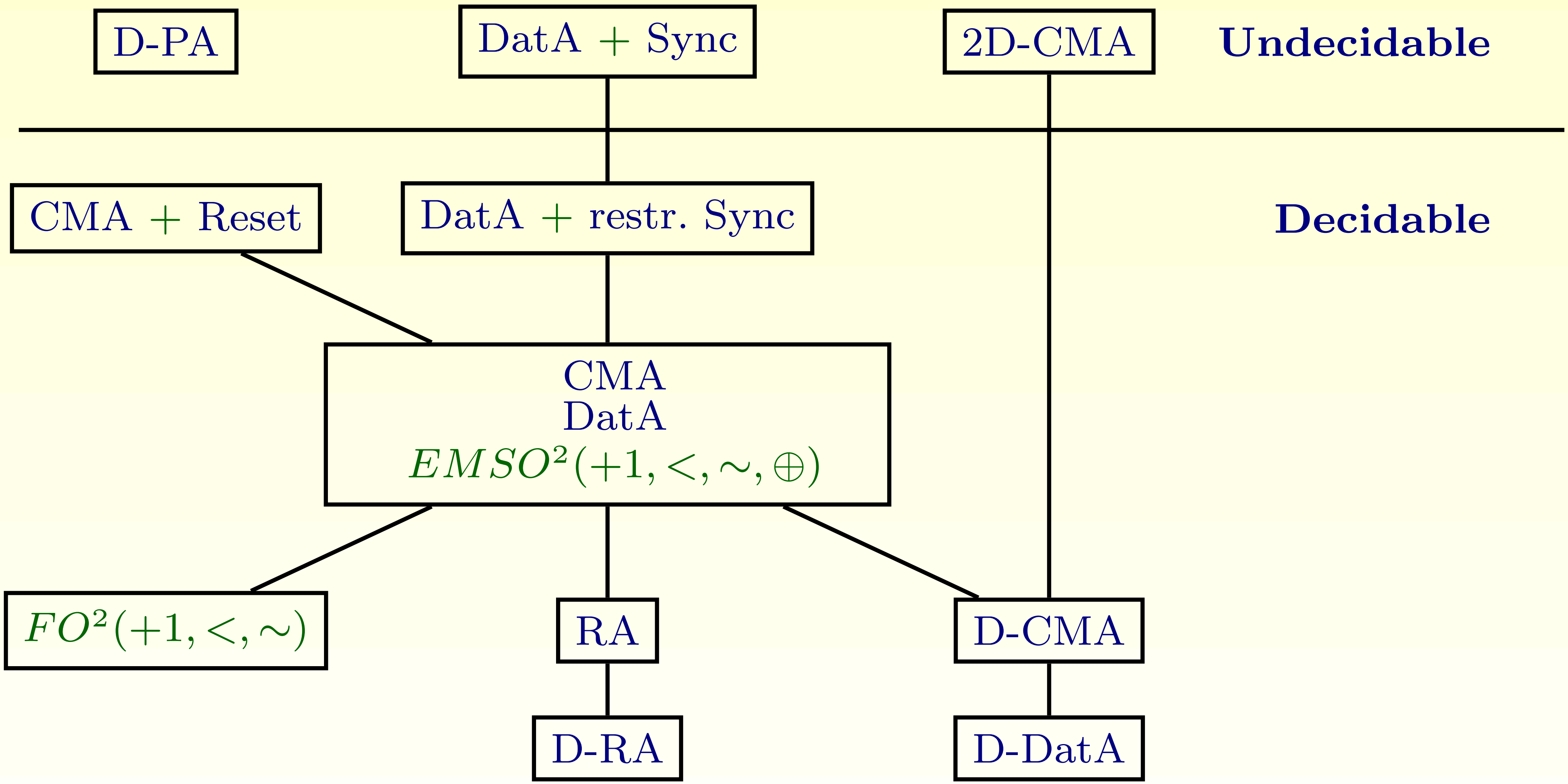
A configuration is a pair (p, f), where $p \in Q$ and $f : \{1, \ldots, k\} \to (\Delta \cup \{\perp\})$ is a register assignment.

We can go from (p, f) to (p', f') with transition $(i, a, p) \to p'$ when reading (a, d) if $f(i) = d$.

We can go from (p, f) to (p', f') with transition $(p, a) \to (i, p')$ if $f(j) \neq d$, for all i, and $f' = f$ except for $f'(i) = d$.

Weakness: Only remembers k data values. Cannot check all regular properties of class strings, e.g. "each class string has length one".

Schematic Picture



Register Automata to Class-Memory Automata

Theorem. Register automata are strictly weaker than class-memory automata.

Register Automata to Class-Memory Automata

Theorem. Register automata are strictly weaker than class-memory automata.

We are given an RA R.

- Associate every accepting run of R with a valid colored trace. (Shown here)
- Show that there is a CMA that determines, for each data word, whether it has a valid colored trace w.r.t. R. (Not shown here)

Register Automata to Class-Memory Automata

Theorem. Register automata are strictly weaker than class-memory automata.

We are given an RA R.

- Associate every accepting run of R with a valid colored trace. (Shown here)
- Show that there is a CMA that determines, for each data word, whether it has a valid colored trace w.r.t. R. (Not shown here)

For each $1 \leq i \leq k$, let $\Gamma_i = \{\langle i \rangle, i, \langle / i \rangle\}$ and $\Gamma = \prod_i \Gamma_i$.

Register Automata to Class-Memory Automata

Theorem. Register automata are strictly weaker than class-memory automata.

We are given an RA R.

- Associate every accepting run of R with a valid colored trace. (Shown here)
- Show that there is a CMA that determines, for each data word, whether it has a valid colored trace w.r.t. R. (Not shown here)

For each $1 \leq i \leq k$, let $\Gamma_i = \{\langle i \rangle, i, \langle / i \rangle\}$ and $\Gamma = \prod_i \Gamma_i$.

A trace is a string over $Q \times \Gamma$, such that for each i, the i-projection is of the form $(\langle i \rangle i^* \langle / i \rangle + \langle / i \rangle)^*$.

Register Automata to Class-Memory Automata

Theorem. Register automata are strictly weaker than class-memory automata.

We are given an RA R.

- Associate every accepting run of R with a valid colored trace. (Shown here)
- Show that there is a CMA that determines, for each data word, whether it has a valid colored trace w.r.t. R. (Not shown here)

For each $1 \leq i \leq k$, let $\Gamma_i = \{\langle i\rangle, i, \langle /i\rangle\}$ and $\Gamma = \prod_i \Gamma_i$.

A trace is a string over $Q \times \Gamma$, such that for each i, the i-projection is of the form $(\langle i\rangle i^* \langle /i\rangle + \langle /i\rangle)^*$.

Let $\rho = (q_0, \tau_0), \ldots, (q_n, \tau_n)$ be a run of R on $w = (a_1, d_1) \ldots (a_n, d_n)$.

Register Automata to Class-Memory Automata

Theorem. Register automata are strictly weaker than class-memory automata.

We are given an RA R.

- Associate every accepting run of R with a valid colored trace. (Shown here)
- Show that there is a CMA that determines, for each data word, whether it has a valid colored trace w.r.t. R. (Not shown here)

For each $1 \leq i \leq k$, let $\Gamma_i = \{\langle i \rangle, i, \langle /i \rangle\}$ and $\Gamma = \prod_i \Gamma_i$.

A trace is a string over $Q \times \Gamma$, such that for each i, the i-projection is of the form $(\langle i \rangle i^* \langle /i \rangle + \langle /i \rangle)^*$.

Let $\rho = (q_0, \tau_0), \ldots, (q_n, \tau_n)$ be a run of R on $w = (a_1, d_1) \ldots (a_n, d_n)$.

A step of ρ is closing if it affects i, for some i, and there is no later step affecting i, or the next such step is a write-transition.

We construct a trace $t(\rho) = t_1 \dots t_n$.

- If the jth step of ρ is a read transition $(i, p, a) \to q$, then t_j is $(q, \langle /i \rangle)$ if the step is closing, otherwise (q, i).
- If the jth step is a write transition $(p, a) \to (q, i)$, then t_j is $(q, \langle i \rangle)$ if the step is closing, otherwise $(q, \langle /i \rangle)$.

We construct a trace $t(\rho) = t_1 \ldots t_n$.

- If the jth step of ρ is a read transition $(i, p, a) \to q$, then t_j is $(q, \langle / i \rangle)$ if the step is closing, otherwise (q, i).
- If the jth step is a write transition $(p, a) \to (q, i)$, then t_j is $(q, \langle i \rangle)$ if the step is closing, otherwise $(q, \langle / i \rangle)$.

A colored trace is a trace where each position is colored by 0 or 1.

We construct a trace $t(\rho) = t_1 \dots t_n$.

- If the jth step of ρ is a read transition $(i, p, a) \to q$, then t_j is $(q, \langle / i \rangle)$ if the step is closing, otherwise (q, i).
- If the jth step is a write transition $(p, a) \to (q, i)$, then t_j is $(q, \langle i \rangle)$ if the step is closing, otherwise $(q, \langle / i \rangle)$.

A colored trace is a trace where each position is colored by 0 or 1.

Given $(q_1, s_1, c_1) \dots (q_n, s_n, c_n)$ and $(a_1, d_1) \dots (a_n, d_n)$, define (for each j)

- $v(j)$ as the maximal $l < j$ such that $d_j = d_l$ (if it exists)
- $r_i(j)$ as the maximal $l < j$ such that $s_l \in \Gamma_i$.

We construct a trace $t(\rho) = t_1 \ldots t_n$.

- If the jth step of ρ is a read transition $(i, p, a) \to q$, then t_j is $(q, \langle /i \rangle)$ if the step is closing, otherwise (q, i).
- If the jth step is a write transition $(p, a) \to (q, i)$, then t_j is $(q, \langle i \rangle)$ if the step is closing, otherwise $(q, \langle /i \rangle)$.

A colored trace is a trace where each position is colored by 0 or 1.

Given $(q_1, s_1, c_1) \ldots (q_n, s_n, c_n)$ and $(a_1, d_1) \ldots (a_n, d_n)$, define (for each j)

- $v(j)$ as the maximal $l < j$ such that $d_j = d_l$ (if it exists)
- $r_i(j)$ as the maximal $l < j$ such that $s_l \in \Gamma_i$.

A colored trace is valid w.r.t. R and w if there are appropriate transitions, q_n is accepting, and for each j, one of the following conditions holds.

We construct a trace $t(\rho) = t_1 \dots t_n$.

- If the jth step of ρ is a read transition $(i, p, a) \to q$, then t_j is $(q, \langle /i \rangle)$ if the step is closing, otherwise (q, i).
- If the jth step is a write transition $(p, a) \to (q, i)$, then t_j is $(q, \langle i \rangle)$ if the step is closing, otherwise $(q, \langle /i \rangle)$.

A colored trace is a trace where each position is colored by 0 or 1.

Given $(q_1, s_1, c_1) \dots (q_n, s_n, c_n)$ and $(a_1, d_1) \dots (a_n, d_n)$, define (for each j)

- $v(j)$ as the maximal $l < j$ such that $d_j = d_l$ (if it exists)
- $r_i(j)$ as the maximal $l < j$ such that $s_l \in \Gamma_i$.

A colored trace is valid w.r.t. R and w if there are appropriate transitions, q_n is accepting, and for each j, one of the following conditions holds.

1. $s_j = i$, for some i, $c_j = c_{v(j)}$, $s_{r_i(j)} \in \{\langle i \rangle, i\}$, and $s_{v(j)} \in \{\langle i \rangle\}$,

We construct a trace $t(\rho) = t_1 \dots t_n$.

- If the jth step of ρ is a read transition $(i, p, a) \to q$, then t_j is $(q, \langle/i\rangle)$ if the step is closing, otherwise (q, i).
- If the jth step is a write transition $(p, a) \to (q, i)$, then t_j is $(q, \langle i\rangle)$ if the step is closing, otherwise $(q, \langle/i\rangle)$.

A colored trace is a trace where each position is colored by 0 or 1.

Given $(q_1, s_1, c_1) \dots (q_n, s_n, c_n)$ and $(a_1, d_1) \dots (a_n, d_n)$, define (for each j)

- $v(j)$ as the maximal $l < j$ such that $d_j = d_l$ (if it exists)
- $r_i(j)$ as the maximal $l < j$ such that $s_l \in \Gamma_i$.

A colored trace is valid w.r.t. R and w if there are appropriate transitions, q_n is accepting, and for each j, one of the following conditions holds.

1. $s_j = i$, for some i, $c_j = c_{v(j)}$, $s_{r_i(j)} \in \{\langle i\rangle, i\}$, and $s_{v(j)} \in \{\langle i\rangle\}$,
2. $s_j = \langle i\rangle$, for some i, $v(j) = \perp$, and register i is closed,

We construct a trace $t(\rho) = t_1 \ldots t_n$.

- If the jth step of ρ is a read transition $(i, p, a) \to q$, then t_j is $(q, \langle/i\rangle)$ if the step is closing, otherwise (q, i).
- If the jth step is a write transition $(p, a) \to (q, i)$, then t_j is $(q, \langle i\rangle)$ if the step is closing, otherwise $(q, \langle/i\rangle)$.

A colored trace is a trace where each position is colored by 0 or 1.

Given $(q_1, s_1, c_1) \ldots (q_n, s_n, c_n)$ and $(a_1, d_1) \ldots (a_n, d_n)$, define (for each j)

- $v(j)$ as the maximal $l < j$ such that $d_j = d_l$ (if it exists)
- $r_i(j)$ as the maximal $l < j$ such that $s_l \in \Gamma_i$.

A colored trace is valid w.r.t. R and w if there are appropriate transitions, q_n is accepting, and for each j, one of the following conditions holds.

1. $s_j = i$, for some i, $c_j = c_{v(j)}$, $s_{r_i(j)} \in \{\langle i\rangle, i\}$, and $s_{v(j)} \in \{\langle i\rangle\}$,
2. $s_j = \langle i\rangle$, for some i, $v(j) = \perp$, and register i is closed,
3. $s_j = \langle i\rangle$, for some i, $s_{v(j)} = \langle p\rangle$, for some p, register i is closed, and $c_{v(j)} \neq c_{r_p(j)}$,

We construct a trace $t(\rho) = t_1 \dots t_n$.

- If the jth step of ρ is a read transition $(i, p, a) \to q$, then t_j is $(q, \langle /i \rangle)$ if the step is closing, otherwise (q, i).
- If the jth step is a write transition $(p, a) \to (q, i)$, then t_j is $(q, \langle i \rangle)$ if the step is closing, otherwise $(q, \langle /i \rangle)$.

A colored trace is a trace where each position is colored by 0 or 1.

Given $(q_1, s_1, c_1) \dots (q_n, s_n, c_n)$ and $(a_1, d_1) \dots (a_n, d_n)$, define (for each j)

- $v(j)$ as the maximal $l < j$ such that $d_j = d_l$ (if it exists)
- $r_i(j)$ as the maximal $l < j$ such that $s_l \in \Gamma_i$.

A colored trace is valid w.r.t. R and w if there are appropriate transitions, q_n is accepting, and for each j, one of the following conditions holds.

1. $s_j = i$, for some i, $c_j = c_{v(j)}$, $s_{r_i(j)} \in \{\langle i \rangle, i\}$, and $s_{v(j)} \in \{\langle i \rangle\}$,
2. $s_j = \langle i \rangle$, for some i, $v(j) = \perp$, and register i is closed,
3. $s_j = \langle i \rangle$, for some i, $s_{v(j)} = \langle p \rangle$, for some p, register i is closed, and $c_{v(j)} \neq c_{r_p(j)}$,
4. $s_j = \langle /i \rangle$, for some i, and one of (1)-(3) applies.

Claim. Data word w is accepted by R iff there is a valid colored trace for w.

Claim. Data word w is accepted by R iff there is a valid colored trace for w.

$(\Leftarrow)$ Straightforward. Important is that the coloring scheme makes sure that the automaton doesn't try to write a data letter into a register that already resides in another register.

Claim. Data word w is accepted by R iff there is a valid colored trace for w.

($\Leftarrow$) Straightforward. Important is that the coloring scheme makes sure that the automaton doesn't try to write a data letter into a register that already resides in another register.

($\Rightarrow$) (Sketch.) We have to show that every trace of an accepting run can be extended to a valid colored trace.

Claim. Data word w is accepted by R iff there is a valid colored trace for w.

($\Leftarrow$) Straightforward. Important is that the coloring scheme makes sure that the automaton doesn't try to write a data letter into a register that already resides in another register.

($\Rightarrow$) (Sketch.) We have to show that every trace of an accepting run can be extended to a valid colored trace.

Pass through the trace from right to left.

Claim. Data word w is accepted by R iff there is a valid colored trace for w.

($\Leftarrow$) Straightforward. Important is that the coloring scheme makes sure that the automaton doesn't try to write a data letter into a register that already resides in another register.

($\Rightarrow$) (Sketch.) We have to show that every trace of an accepting run can be extended to a valid colored trace.

Pass through the trace from right to left.

Positions with i or $\langle i\rangle$ get the same color as their corresponding closing tag.

Claim. Data word w is accepted by R iff there is a valid colored trace for w.

($\Leftarrow$) Straightforward. Important is that the coloring scheme makes sure that the automaton doesn't try to write a data letter into a register that already resides in another register.

($\Rightarrow$) (Sketch.) We have to show that every trace of an accepting run can be extended to a valid colored trace.

Pass through the trace from right to left.

Positions with i or $\langle i \rangle$ get the same color as their corresponding closing tag.

For positions j with $s_j = \langle /i \rangle$, let $l > j$ be minimal s.t. $d_j = d_l$. Set $c_j = 1 - c_l$.

Claim. Data word w is accepted by R iff there is a valid colored trace for w.

($\Leftarrow$) Straightforward. Important is that the coloring scheme makes sure that the automaton doesn't try to write a data letter into a register that already resides in another register.

($\Rightarrow$) (Sketch.) We have to show that every trace of an accepting run can be extended to a valid colored trace.

Pass through the trace from right to left.

Positions with i or $\langle i \rangle$ get the same color as their corresponding closing tag.

For positions j with $s_j = \langle /i \rangle$, let $l > j$ be minimal s.t. $d_j = d_l$. Set $c_j = 1 - c_l$.

This gives a valid colored trace.

CMAs with Reset

Class-memory automata cannot recognize the language "between every two a positions in the same class, there is a b".

CMAs with Reset

Class-memory automata cannot recognize the language "between every two a positions in the same class, there is a b".

This property is expressible in $LTL\downarrow_1(X, U)$ and by alternating 1-register automata. [Demri & Lazić]

CMAs with Reset

Class-memory automata cannot recognize the language "between every two a positions in the same class, there is a b".

This property is expressible in $LTL\downarrow_1(X, U)$ and by alternating 1-register automata. [Demri & Lazić]

We extend CMAs by allowing reset transitions, erasing the memory for every class (setting $f(d) = \perp$ for all $d \in \Delta$).

CMAs with Reset

Class-memory automata cannot recognize the language "between every two a positions in the same class, there is a b".

This property is expressible in $LTL\downarrow_1(X, U)$ and by alternating 1-register automata. [Demri & Lazić]

We extend CMAs by allowing reset transitions, erasing the memory for every class (setting $f(d) = \perp$ for all $d \in \Delta$).

Condition: The reset transitions may only be taken if $f(d) \in F_2 \cup \{\perp\}$ for all $d \in \Delta$.

CMAs with Reset

Class-memory automata cannot recognize the language "between every two a positions in the same class, there is a b".

This property is expressible in $LTL\downarrow_1(X, U)$ and by alternating 1-register automata. [Demri & Lazić]

We extend CMAs by allowing reset transitions, erasing the memory for every class (setting $f(d) = \perp$ for all $d \in \Delta$).

Condition: The reset transitions may only be taken if $f(d) \in F_2 \cup \{\perp\}$ for all $d \in \Delta$.

We preserve decidability by using priority multicounter automata. When taking a reset transition, empty the counters for F_2, then zero-check all counters at once.

CMAs with Reset

Class-memory automata cannot recognize the language "between every two a positions in the same class, there is a b".

This property is expressible in $LTL\downarrow_1(X, U)$ and by alternating 1-register automata. [Demri & Lazić]

We extend CMAs by allowing reset transitions, erasing the memory for every class (setting $f(d) = \perp$ for all $d \in \Delta$).

Condition: The reset transitions may only be taken if $f(d) \in F_2 \cup \{\perp\}$ for all $d \in \Delta$.

We preserve decidability by using priority multicounter automata. When taking a reset transition, empty the counters for F_2, then zero-check all counters at once.

Benefit: Can recognize the Kleene * of a data automaton language.

2-Way Deterministic CMA

2-Way Deterministic CMA

Theorem. Emptiness for 2-Way Deterministic CMA is undecidable.

2-Way Deterministic CMA

Theorem. Emptiness for 2-Way Deterministic CMA is undecidable.

The proof is by reduction from PCP.

The PCP

- Instances: $(x_1, y_1), \ldots, (x_n, y_n)$, where $x_i, y_i \in \{a, b\}^*$.
- Question: Is there a finite sequence $i_1, \ldots, i_m$ such that $x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m}$?

2-Way Deterministic CMA

Theorem. Emptiness for 2-Way Deterministic CMA is undecidable.

The proof is by reduction from PCP.

The PCP

- Instances: $(x_1, y_1), \ldots, (x_n, y_n)$, where $x_i, y_i \in \{a, b\}^*$.
- Question: Is there a finite sequence $i_1, \ldots, i_m$ such that $x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m}$?

Given instance I we construct a CMA A whose language is nonempty iff I has a solution.

Encoding solution

Alphabet: $\{a, b, \#\} \cup \{1, \ldots, n\}$.

Encoding solution

Alphabet: $\{a, b, \#\} \cup \{1, \ldots, n\}$.

Construct the string $i_1 x_{i_1} \ldots i_m x_{i_m} \# i_1 y_{i_1} \ldots i_m y_{i_m}$.

Encoding solution

Alphabet: $\{a, b, \#\} \cup \{1, \ldots, n\}$.

Construct the string $i_1 x_{i_1} \ldots i_m x_{i_m} \# i_1 y_{i_1} \ldots i_m y_{i_m}$.

- $\#$ gets a unique data value
- for $1 \leq j \leq m$, both occurrences of i_j get the same (unique) data value.
- two letter positions get the same data (unique) data value if they represent the same position in $x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m}$.

Encoding solution

Alphabet: $\{a, b, \#\} \cup \{1, \ldots, n\}$.

Construct the string $i_1 x_{i_1} \ldots i_m x_{i_m} \# i_1 y_{i_1} \ldots i_m y_{i_m}$.

- $\#$ gets a unique data value
- for $1 \leq j \leq m$, both occurrences of i_j get the same (unique) data value.
- two letter positions get the same data (unique) data value if they represent the same position in $x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m}$.

Constructing A

A should accept only correct encodings of solutions to I.

Encoding solution

Alphabet: $\{a, b, \#\} \cup \{1, \ldots, n\}$.

Construct the string $i_1 x_{i_1} \ldots i_m x_{i_m} \# i_1 y_{i_1} \ldots i_m y_{i_m}$.

- $\#$ gets a unique data value
- for $1 \leq j \leq m$, both occurrences of i_j get the same (unique) data value.
- two letter positions get the same data (unique) data value if they represent the same position in $x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m}$.

Constructing A

A should accept only correct encodings of solutions to I.

1. Check that each data value appears exactly twice (except the one for $\#$).

Encoding solution

Alphabet: $\{a, b, \#\} \cup \{1, \ldots, n\}$.

Construct the string $i_1 x_{i_1} \ldots i_m x_{i_m} \# i_1 y_{i_1} \ldots i_m y_{i_m}$.

- $\#$ gets a unique data value
- for $1 \leq j \leq m$, both occurrences of i_j get the same (unique) data value.
- two letter positions get the same data (unique) data value if they represent the same position in $x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m}$.

Constructing A

A should accept only correct encodings of solutions to I.

1. Check that each data value appears exactly twice (except the one for $\#$).
2. For each $1 \leq j \leq m$, check that the string following i_j is x_{i_j} (y_{i_j}).

Encoding solution

Alphabet: $\{a, b, \#\} \cup \{1, \ldots, n\}$.

Construct the string $i_1 x_{i_1} \ldots i_m x_{i_m} \# i_1 y_{i_1} \ldots i_m y_{i_m}$.

- $\#$ gets a unique data value
- for $1 \leq j \leq m$, both occurrences of i_j get the same (unique) data value.
- two letter positions get the same data (unique) data value if they represent the same position in $x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m}$.

Constructing A

A should accept only correct encodings of solutions to I.

1. Check that each data value appears exactly twice (except the one for $\#$).
2. For each $1 \leq j \leq m$, check that the string following i_j is x_{i_j} (y_{i_j}).
3. Check that the sequences of indices are the same on both sides of $\#$.

Encoding solution

Alphabet: $\{a, b, \#\} \cup \{1, \ldots, n\}$.

Construct the string $i_1 x_{i_1} \ldots i_m x_{i_m} \# i_1 y_{i_1} \ldots i_m y_{i_m}$.

- $\#$ gets a unique data value
- for $1 \leq j \leq m$, both occurrences of i_j get the same (unique) data value.
- two letter positions get the same data (unique) data value if they represent the same position in $x_{i_1} \ldots x_{i_m} = y_{i_1} \ldots y_{i_m}$.

Constructing A

A should accept only correct encodings of solutions to I.

1. Check that each data value appears exactly twice (except the one for $\#$).
2. For each $1 \leq j \leq m$, check that the string following i_j is x_{i_j} (y_{i_j}).
3. Check that the sequences of indices are the same on both sides of $\#$.
4. Check that the strings formed by the text positions are the same on both sides of $\#$.

Model Checking (Word Problem)

We consider the model checking problem for CMA and RA, w.r.t. data complexity and combined complexity.

Model Checking (Word Problem)

We consider the model checking problem for CMA and RA, w.r.t. data complexity and combined complexity.

For deterministic CMA and RA, model checking is polynomial.

Model Checking (Word Problem)

We consider the model checking problem for CMA and RA, w.r.t. data complexity and combined complexity.

For deterministic CMA and RA, model checking is polynomial.

Proposition. For CMA, the data complexity of model checking is NP-complete.

Proof is by reduction from 3-SAT.

Model Checking (Word Problem)

We consider the model checking problem for CMA and RA, w.r.t. data complexity and combined complexity.

For deterministic CMA and RA, model checking is polynomial.

Proposition. For CMA, the data complexity of model checking is NP-complete.

Proof is by reduction from 3-SAT.

Proposition. For RA, the data complexity of model checking is polynomial.

When reading input w, there are at most $|Q| \cdot \binom{|w|}{k} \cdot k!$

Proposition. For RA, the combined complexity of model checking is NP-complete.

Proof is by reduction from 3-SAT.

The complexity of model checking for RA depends strongly on the number of registers.

The complexity of model checking for RA depends strongly on the number of registers.

Proposition. Model checking for RA, parameterized by the number of registers, is $W[1]$-hard.

The complexity of model checking for RA depends strongly on the number of registers.

Proposition. Model checking for RA, parameterized by the number of registers, is $W[1]$-hard.

By reduction from k-Clique.

The complexity of model checking for RA depends strongly on the number of registers.

Proposition. Model checking for RA, parameterized by the number of registers, is $W[1]$-hard.

By reduction from k-Clique.

Given $G = (V, E)$, let $\Sigma = V$.

The complexity of model checking for RA depends strongly on the number of registers.

Proposition. Model checking for RA, parameterized by the number of registers, is $W[1]$-hard.

By reduction from k-Clique.

Given $G = (V, E)$, let $\Sigma = V$.

Encode G as a data word uv, where u enumerates the vertices, and v is enumerates the edges not in G as vertex pairs. Every time a vertex appears, it has the same (unique) data value.

The complexity of model checking for RA depends strongly on the number of registers.

Proposition. Model checking for RA, parameterized by the number of registers, is $W[1]$-hard.

By reduction from k-Clique.

Given $G = (V, E)$, let $\Sigma = V$.

Encode G as a data word uv, where u enumerates the vertices, and v is enumerates the edges not in G as vertex pairs. Every time a vertex appears, it has the same (unique) data value.

A RA with $k + 1$ registers can nondeterministically guess k vertices while reading u. It then checks that no pair represents an edge between two guessed vertices.

Open Problems

Open Problems

Closure properties. What are reasonable definitions of homomorphisms?

Open Problems

Closure properties. What are reasonable definitions of homomorphisms?

Algebraic properties

Open Problems

Closure properties. What are reasonable definitions of homomorphisms?

Algebraic properties

Alternative characterizations for deterministic CMAs

Open Problems

Closure properties. What are reasonable definitions of homomorphisms?

Algebraic properties

Alternative characterizations for deterministic CMAs

How hard is it to determine whether $L(r) \cap L(r')^{\otimes} = \emptyset$?