Tree-automata, interpretations and Courcelle's theorem

Matthieu Rosenfeld

December 14, 2023

Courcelle's theorem as a generalization

Theorem (Courcelle, 1990)

Every graph property definable in MSOL can be decided in linear time on graphs of bounded treewidth.

Courcelle's theorem as a generalization

Theorem (Courcelle, 1990)

Every graph property definable in MSOL can be decided in linear time on graphs of bounded treewidth.

Theorem (Rabin, 1969)

Every tree property definable in MSOL can be decided in linear time on trees.

Courcelle's theorem as a generalization

Theorem (Courcelle, 1990)

Every graph property definable in MSOL can be decided in linear time on graphs of bounded treewidth.

Theorem (Rabin, 1969)

Every tree property definable in MSOL can be decided in linear time on trees.

Theorem (Büchi–Elgot–Trakhtenbrot, 1960)

Every word property definable in MSOL can be decided in linear time.

Courcelle's theorem as a generalization

Theorem (Courcelle, 1990)

Every graph property definable in MSOL can be decided in linear time on graphs of bounded treewidth.

Theorem (Rabin, 1969)

Every tree property definable in MSOL can be decided in linear time on trees.

Theorem (Büchi–Elgot–Trakhtenbrot, 1960)

Every word property definable in MSOL can be decided in linear time.

The set of words defined by any given MSO formula is recognized by an automaton.

Courcelle's theorem as a generalization

Theorem (Courcelle, 1990)

Every graph property definable in MSOL can be decided in linear time on graphs of bounded treewidth.

Theorem (Rabin, 1969)

Every tree property definable in MSOL can be decided in linear time on trees.

The set of trees defined by any given MSO formula is recognized by a tree-automaton.

Theorem (Büchi–Elgot–Trakhtenbrot, 1960)

Every word property definable in MSOL can be decided in linear time.

The set of words defined by any given MSO formula is recognized by an automaton.

Courcelle's theorem as a generalization

Theorem (Courcelle, 1990)

Every graph property definable in MSOL can be decided in linear time on graphs of bounded treewidth.

The tree-decomposition of graphs of bounded treewidth defined by any given MSO formula is recognized by a tree-automaton.

Theorem (Rabin, 1969)

Every tree property definable in MSOL can be decided in linear time on trees.

The set of trees defined by any given MSO formula is recognized by a tree-automaton.

Theorem (Büchi–Elgot–Trakhtenbrot, 1960)

Every word property definable in MSOL can be decided in linear time.

The set of words defined by any given MSO formula is recognized by an automaton.

Automaton and MSO

DFA

Definition (Deterministic Finite Automaton (DFA))

A deterministic finite automaton $\mathcal{A}$ is a 5-tuple, $(\Sigma, Q, \delta, q_0, F)$, consisting of

DFA

Definition (Deterministic Finite Automaton (DFA))

A deterministic finite automaton $\mathcal{A}$ is a 5-tuple, $(\Sigma, Q, \delta, q_0, F)$, consisting of

- an alphabet Σ,

$$\Sigma = \{0, 1\}$$

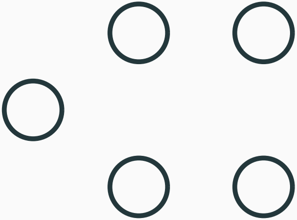
DFA

Definition (Deterministic Finite Automaton (DFA))

A deterministic finite automaton $\mathcal{A}$ is a 5-tuple, $(\Sigma, Q, \delta, q_0, F)$, consisting of

- an alphabet Σ,
- a finite set of states Q,

$$\Sigma = \{0, 1\}$$



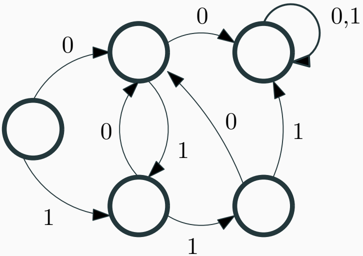
DFA

Definition (Deterministic Finite Automaton (DFA))

A deterministic finite automaton $\mathcal{A}$ is a 5-tuple, $(\Sigma, Q, \delta, q_0, F)$, consisting of

- an alphabet Σ,
- a finite set of states Q,
- a transition function $\delta : Q \times \Sigma \to Q$,

$\Sigma = \{0, 1\}$



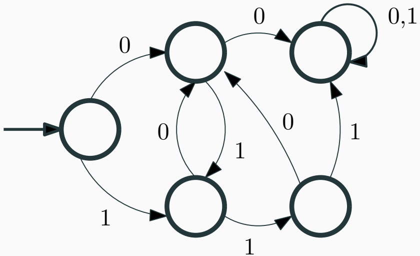
DFA

Definition (Deterministic Finite Automaton (DFA))

A deterministic finite automaton $\mathcal{A}$ is a 5-tuple, $(\Sigma, Q, \delta, q_0, F)$, consisting of

- an alphabet Σ,
- a finite set of states Q,
- a transition function $\delta : Q \times \Sigma \to Q$,
- an initial state $q_0 \in Q$,

$\Sigma = \{0, 1\}$



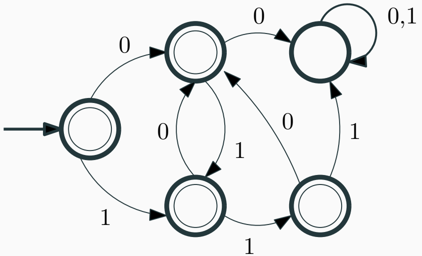
DFA

Definition (Deterministic Finite Automaton (DFA))

A deterministic finite automaton $\mathcal{A}$ is a 5-tuple, $(\Sigma, Q, \delta, q_0, F)$, consisting of

- an alphabet Σ,
- a finite set of states Q,
- a transition function $\delta : Q \times \Sigma \to Q$,
- an initial state $q_0 \in Q$,
- a set of accepting states $F \subseteq Q$.

$\Sigma = \{0, 1\}$



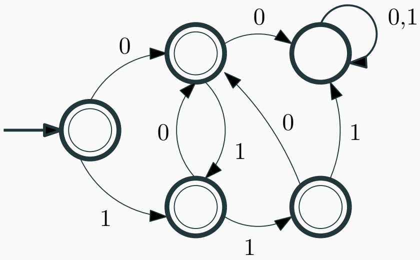
Language of a DFA

Definition (Recognized word)

We say that a DFA $(\Sigma, Q, \delta, q_0, F)$ accepts a word $w_1 \dots w_n \in Q^n$, if there exists a sequence $q_1, \dots, q_n$ such that:

- for all $i \in \{1, \dots, n\}$, $q_i = \delta(q_{i-1}, w_i)$,
- $q_n \in F$.

$\Sigma = \{0, 1\}$



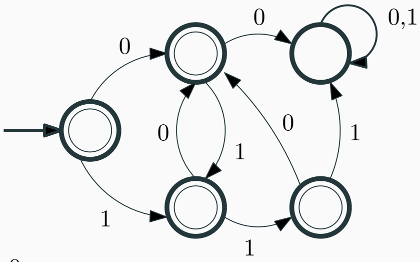
Language of a DFA

Definition (Recognized word)

We say that a DFA $(\Sigma, Q, \delta, q_0, F)$ accepts a word $w_1 \dots w_n \in Q^n$, if there exists a sequence $q_1, \dots, q_n$ such that:

- for all $i \in \{1, \dots, n\}$, $q_i = \delta(q_{i-1}, w_i)$,
- $q_n \in F$.

$\Sigma = \{0, 1\}$



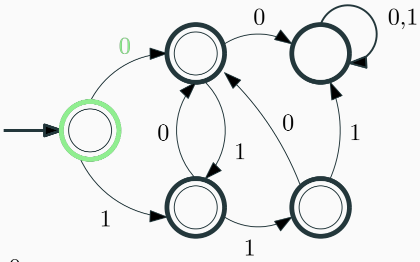
$w = 0\ 1\ 0\ 1\ 1\ 0$

Definition (Recognized word)

We say that a DFA $(\Sigma, Q, \delta, q_0, F)$ accepts a word $w_1 \ldots w_n \in Q^n$, if there exists a sequence $q_1, \ldots, q_n$ such that:

- for all $i \in \{1, \ldots, n\}$, $q_i = \delta(q_{i-1}, w_i)$,
- $q_n \in F$.

$\Sigma = \{0, 1\}$



$w = 0\ 1\ 0\ 1\ 1\ 0$

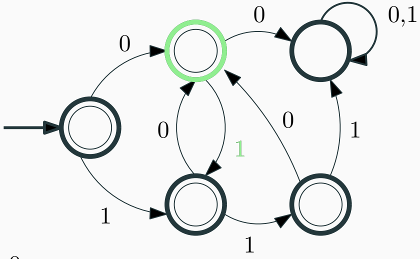
Language of a DFA

Definition (Recognized word)

We say that a DFA $(\Sigma, Q, \delta, q_0, F)$ accepts a word $w_1 \ldots w_n \in Q^n$, if there exists a sequence $q_1, \ldots, q_n$ such that:

- for all $i \in \{1, \ldots, n\}$, $q_i = \delta(q_{i-1}, w_i)$,
- $q_n \in F$.

$\Sigma = \{0, 1\}$



$w = \boxed{0}\ 1\ 0\ 1\ 1\ 0$

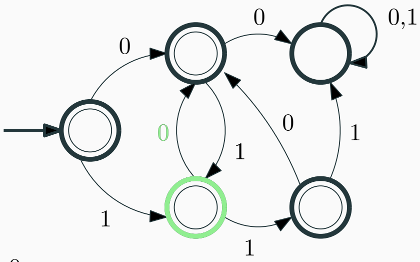
Language of a DFA

Definition (Recognized word)

We say that a DFA $(\Sigma, Q, \delta, q_0, F)$ accepts a word $w_1 \dots w_n \in Q^n$, if there exists a sequence $q_1, \dots, q_n$ such that:

- for all $i \in \{1, \dots, n\}$, $q_i = \delta(q_{i-1}, w_i)$,
- $q_n \in F$.

$\Sigma = \{0, 1\}$



$w = \boxed{0\ 1}\ 0\ 1\ 1\ 0$

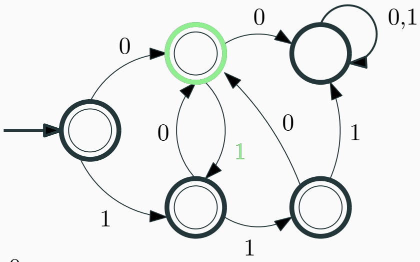
Language of a DFA

Definition (Recognized word)

We say that a DFA $(\Sigma, Q, \delta, q_0, F)$ accepts a word $w_1 \dots w_n \in Q^n$, if there exists a sequence $q_1, \dots, q_n$ such that:

- for all $i \in \{1, \dots, n\}$, $q_i = \delta(q_{i-1}, w_i)$,
- $q_n \in F$.

$\Sigma = \{0, 1\}$



$w = \boxed{0\ 1\ 0}\ 1\ 1\ 0$

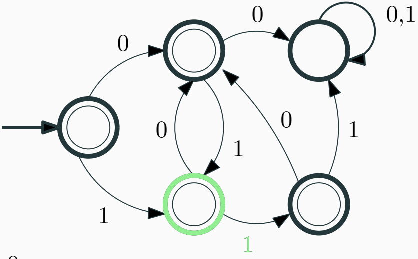
Language of a DFA

Definition (Recognized word)

We say that a DFA $(\Sigma, Q, \delta, q_0, F)$ accepts a word $w_1 \dots w_n \in Q^n$, if there exists a sequence $q_1, \dots, q_n$ such that:

- for all $i \in \{1, \dots, n\}$, $q_i = \delta(q_{i-1}, w_i)$,
- $q_n \in F$.

$\Sigma = \{0, 1\}$



$w = \boxed{0\ 1\ 0\ 1}\ 1\ 0$

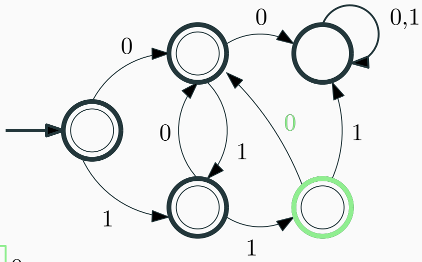
Language of a DFA

Definition (Recognized word)

We say that a DFA $(\Sigma, Q, \delta, q_0, F)$ accepts a word $w_1 \dots w_n \in Q^n$, if there exists a sequence $q_1, \dots, q_n$ such that:

- for all $i \in \{1, \dots, n\}$, $q_i = \delta(q_{i-1}, w_i)$,
- $q_n \in F$.

$\Sigma = \{0, 1\}$



$w = \boxed{0\ 1\ 0\ 1\ 1}\ 0$

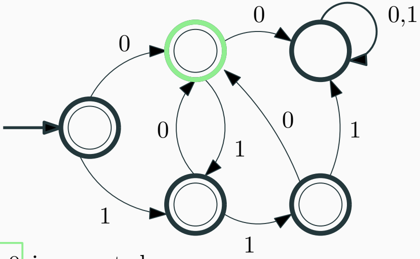
Language of a DFA

Definition (Recognized word)

We say that a DFA $(\Sigma, Q, \delta, q_0, F)$ accepts a word $w_1 \dots w_n \in Q^n$, if there exists a sequence $q_1, \dots, q_n$ such that:

- for all $i \in \{1, \dots, n\}$, $q_i = \delta(q_{i-1}, w_i)$,
- $q_n \in F$.

$\Sigma = \{0, 1\}$



$w = \boxed{0\ 1\ 0\ 1\ 1\ 0}$ is accepted

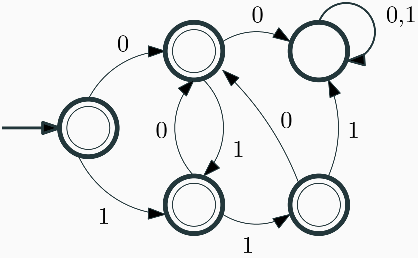
Language of a DFA

Definition (Recognized word)

We say that a DFA $(\Sigma, Q, \delta, q_0, F)$ accepts a word $w_1 \dots w_n \in Q^n$, if there exists a sequence $q_1, \dots, q_n$ such that:

- for all $i \in \{1, \dots, n\}$, $q_i = \delta(q_{i-1}, w_i)$,
- $q_n \in F$.

$\Sigma = \{0, 1\}$



$w = 1\ 0\ 0\ 1$

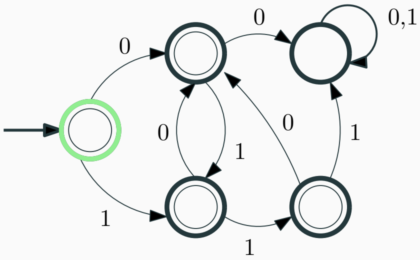
Language of a DFA

Definition (Recognized word)

We say that a DFA $(\Sigma, Q, \delta, q_0, F)$ accepts a word $w_1 \dots w_n \in Q^n$, if there exists a sequence $q_1, \dots, q_n$ such that:

- for all $i \in \{1, \dots, n\}$, $q_i = \delta(q_{i-1}, w_i)$,
- $q_n \in F$.

$\Sigma = \{0, 1\}$



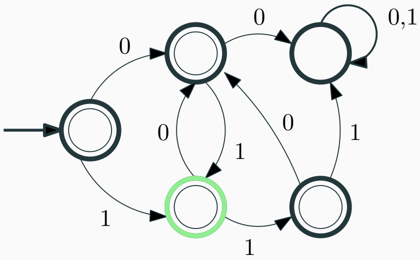
$w = 1\ 0\ 0\ 1$

Definition (Recognized word)

We say that a DFA $(\Sigma, Q, \delta, q_0, F)$ accepts a word $w_1 \dots w_n \in Q^n$, if there exists a sequence $q_1, \dots, q_n$ such that:

- for all $i \in \{1, \dots, n\}$, $q_i = \delta(q_{i-1}, w_i)$,
- $q_n \in F$.

$\Sigma = \{0, 1\}$



$w = \boxed{1}\ 0\ 0\ 1$

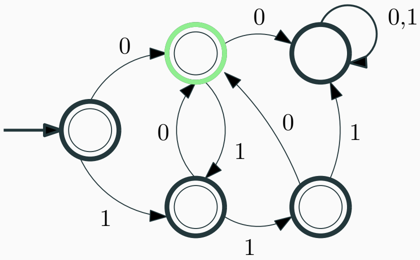
Language of a DFA

Definition (Recognized word)

We say that a DFA $(\Sigma, Q, \delta, q_0, F)$ accepts a word $w_1 \dots w_n \in Q^n$, if there exists a sequence $q_1, \dots, q_n$ such that:

- for all $i \in \{1, \dots, n\}$, $q_i = \delta(q_{i-1}, w_i)$,
- $q_n \in F$.

$\Sigma = \{0, 1\}$



$w = \boxed{1\ 0}\ 0\ 1$

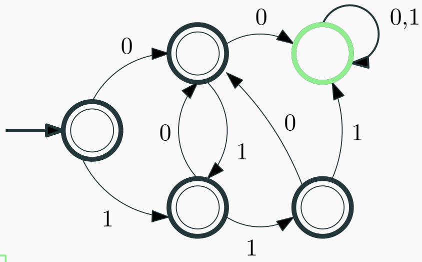
Language of a DFA

Definition (Recognized word)

We say that a DFA $(\Sigma, Q, \delta, q_0, F)$ accepts a word $w_1 \dots w_n \in Q^n$, if there exists a sequence $q_1, \dots, q_n$ such that:

- for all $i \in \{1, \dots, n\}$, $q_i = \delta(q_{i-1}, w_i)$,
- $q_n \in F$.

$\Sigma = \{0, 1\}$



$w = \boxed{1\ 0\ 0}\ 1$

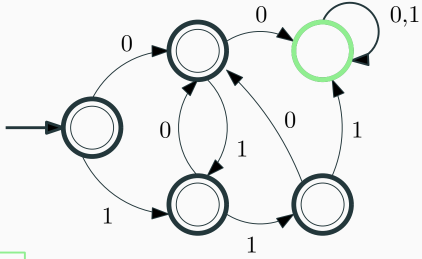
Language of a DFA

Definition (Recognized word)

We say that a DFA $(\Sigma, Q, \delta, q_0, F)$ accepts a word $w_1 \dots w_n \in Q^n$, if there exists a sequence $q_1, \dots, q_n$ such that:

- for all $i \in \{1, \dots, n\}$, $q_i = \delta(q_{i-1}, w_i)$,
- $q_n \in F$.

$\Sigma = \{0, 1\}$



$w = \boxed{1\ 0\ 0\ 1}$ is not accepted

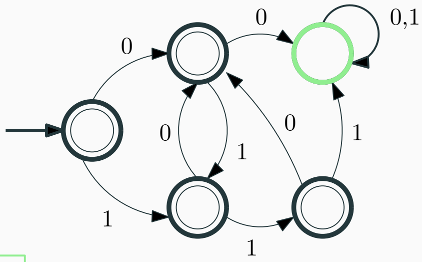
Language of a DFA

Definition (Recognized word)

We say that a DFA $(\Sigma, Q, \delta, q_0, F)$ accepts a word $w_1 \dots w_n \in Q^n$, if there exists a sequence $q_1, \dots, q_n$ such that:

- for all $i \in \{1, \dots, n\}$, $q_i = \delta(q_{i-1}, w_i)$,
- $q_n \in F$.

$\Sigma = \{0, 1\}$



$w = \boxed{1\ 0\ 0\ 1}$ is not accepted

Note: For a fixed DFA $\mathcal{A}$, testing if w is recognized by $\mathcal{A}$ is linear in $|w|$.

MSO for words

The variables are positions $(x, y, \ldots)$ and sets of positions $(X, Y, \ldots)$.

MSO for words

The variables are positions $(x, y, \ldots)$ and sets of positions $(X, Y, \ldots)$.

For all letter α, we have a predicate Q_α which is interpreted as

$$Q_\alpha(x) := \text{“Position x is labelled } \alpha\text{”}$$

MSO for words

The variables are positions $(x, y, \ldots)$ and sets of positions $(X, Y, \ldots)$.

For all letter α, we have a predicate Q_α which is interpreted as

$$Q_\alpha(x) := \text{"Position x is labelled } \alpha\text{"}$$

The syntax of *MSO* on words.

$$\begin{aligned}\phi := \quad & x = y + 1 \mid x = y \mid Q_\alpha(x) \mid X = Y \mid x \in X \\ & \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \implies \phi_2 \\ & \mid \forall x.\phi \mid \forall X.\phi \mid \exists x.\phi \mid \exists X.\phi,\end{aligned}$$

MSO for words

The variables are positions $(x, y, \ldots)$ and sets of positions $(X, Y, \ldots)$.

For all letter α, we have a predicate Q_α which is interpreted as

$$Q_\alpha(x) := \text{“Position x is labelled } \alpha\text{”}$$

The syntax of *MSO* on words.

$$\begin{aligned}\phi := \quad & x = y + 1 \mid x = y \mid Q_\alpha(x) \mid X = Y \mid x \in X \\ & \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \implies \phi_2 \\ & \mid \forall x.\phi \mid \forall X.\phi \mid \exists x.\phi \mid \exists X.\phi,\end{aligned}$$

$$\forall x, Q_a(x) \implies (\forall y, y = x + 1 \implies Q_b(y))$$

MSO for words

The variables are positions $(x, y, \ldots)$ and sets of positions $(X, Y, \ldots)$.

For all letter α, we have a predicate Q_α which is interpreted as

$$Q_\alpha(x) := \text{“Position x is labelled } \alpha\text{”}$$

The syntax of *MSO* on words.

$$\begin{aligned} \phi := \quad & x = y + 1 \mid x = y \mid Q_\alpha(x) \mid X = Y \mid x \in X \\ & \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \implies \phi_2 \\ & \mid \forall x.\phi \mid \forall X.\phi \mid \exists x.\phi \mid \exists X.\phi, \end{aligned}$$

$$\forall x, Q_a(x) \implies (\forall y, y = x + 1 \implies Q_b(y))$$

“Every *a* is followed by *b*”

MSO and automaton

Theorem (Büchi–Elgot–Trakhtenbrot, 1960)

For any MSO formula ϕ, there exists a DFA $\mathcal{A}(\phi)$ that recognizes exactly the words accepted by ϕ.

MSO and automaton

Theorem (Büchi–Elgot–Trakhtenbrot, 1960)

For any MSO formula ϕ, there exists a DFA $\mathcal{A}(\phi)$ that recognizes exactly the words accepted by ϕ.

Corollary

For any MSO formula ϕ, there exists a linear time algorithm that decide if any given word w is accepted by ϕ.

Idea of the proof - I

The syntax of *MSO* on words can be reduced to:

$$\begin{aligned} \phi := \quad & x = y + 1 \mid Q_\alpha(x) \mid x \in X \mid X = Y \mid x = y \\ & \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \implies \phi_2 \\ & \mid \exists x.\phi \mid \exists X.\phi \mid \forall x.\phi \mid \forall X.\phi, \end{aligned}$$

Idea of the proof - I

The syntax of *MSO* on words can be reduced to:

$$
\begin{aligned}
\phi := \quad & x = y + 1 \mid Q_\alpha(x) \mid x \in X \mid X = Y \mid x = y \\
& \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \implies \phi_2 \\
& \mid \exists x.\phi \mid \exists X.\phi \mid \forall x.\phi \mid \forall X.\phi,
\end{aligned}
$$

We will prove the result by induction on the MSO formula.

Idea of the proof - II

The proof is by induction on the MSO formula for this we need two notions:

Idea of the proof - II

The proof is by induction on the MSO formula for this we need two notions:

MSO formula with a free variable. *ex:* $\phi(S) := \quad \forall y, y \in S \implies Q_a(y)$

Idea of the proof - II

The proof is by induction on the MSO formula for this we need two notions:

MSO formula with a free variable. *ex:* $\phi(S) := \quad \forall y, y \in S \implies Q_a(y)$

How do we give S to the automaton ??

Idea of the proof - II

The proof is by induction on the MSO formula for this we need two notions:

MSO formula with a free variable. *ex:* $\phi(S) := \quad \forall y, y \in S \implies Q_a(y)$

How do we give S to the automaton ??

Instead of $\{a, b\}$, we use the alphabet $\left\{ \begin{pmatrix} a \\ 0 \end{pmatrix}, \begin{pmatrix} a \\ 1 \end{pmatrix}, \begin{pmatrix} b \\ 0 \end{pmatrix}, \begin{pmatrix} b \\ 1 \end{pmatrix} \right\}$

Idea of the proof - II

The proof is by induction on the MSO formula for this we need two notions:

MSO formula with a free variable. *ex:* $\phi(S) := \quad \forall y, y \in S \implies Q_a(y)$

How do we give S to the automaton ??

Instead of $\{a, b\}$, we use the alphabet $\left\{ \begin{pmatrix} a \\ 0 \end{pmatrix}, \begin{pmatrix} a \\ 1 \end{pmatrix}, \begin{pmatrix} b \\ 0 \end{pmatrix}, \begin{pmatrix} b \\ 1 \end{pmatrix} \right\}$

ex: $w = ababaab$ and $S = \{0, 2, 5\}$ gives:

$$w, S = \begin{pmatrix} a \\ 1 \end{pmatrix} \begin{pmatrix} b \\ 0 \end{pmatrix} \begin{pmatrix} a \\ 1 \end{pmatrix} \begin{pmatrix} b \\ 0 \end{pmatrix} \begin{pmatrix} a \\ 0 \end{pmatrix} \begin{pmatrix} a \\ 1 \end{pmatrix} \begin{pmatrix} b \\ 0 \end{pmatrix}$$

Idea of the proof - II

The proof is by induction on the MSO formula for this we need two notions:

MSO formula with a free variable. *ex:* $\phi(S) := \quad \forall y, y \in S \implies Q_a(y)$

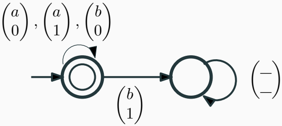
How do we give S to the automaton ??

Instead of $\{a, b\}$, we use the alphabet $\left\{ \begin{pmatrix} a \\ 0 \end{pmatrix}, \begin{pmatrix} a \\ 1 \end{pmatrix}, \begin{pmatrix} b \\ 0 \end{pmatrix}, \begin{pmatrix} b \\ 1 \end{pmatrix} \right\}$

ex: $w = ababaab$ and $S = \{0, 2, 5\}$ gives:

$$w, S = \begin{pmatrix} a \\ 1 \end{pmatrix} \begin{pmatrix} b \\ 0 \end{pmatrix} \begin{pmatrix} a \\ 1 \end{pmatrix} \begin{pmatrix} b \\ 0 \end{pmatrix} \begin{pmatrix} a \\ 0 \end{pmatrix} \begin{pmatrix} a \\ 1 \end{pmatrix} \begin{pmatrix} b \\ 0 \end{pmatrix}$$

ex:



The proof by induction:

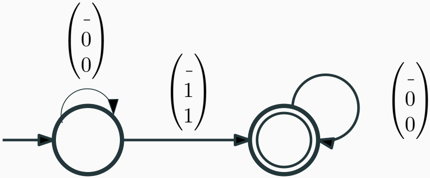
$$\mathcal{A}(x = y)$$

$$\begin{pmatrix} \Sigma \\ x \\ y \end{pmatrix}$$

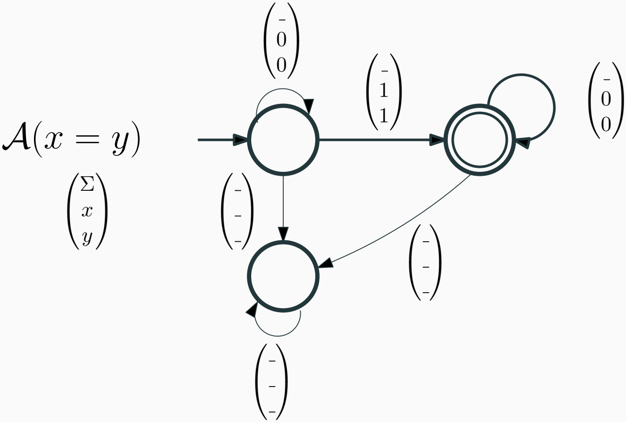
The proof by induction:

$\mathcal{A}(x = y)$

$\begin{pmatrix}\Sigma \\ x \\ y\end{pmatrix}$



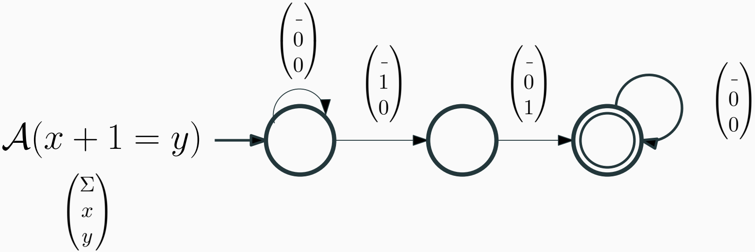
The proof by induction:



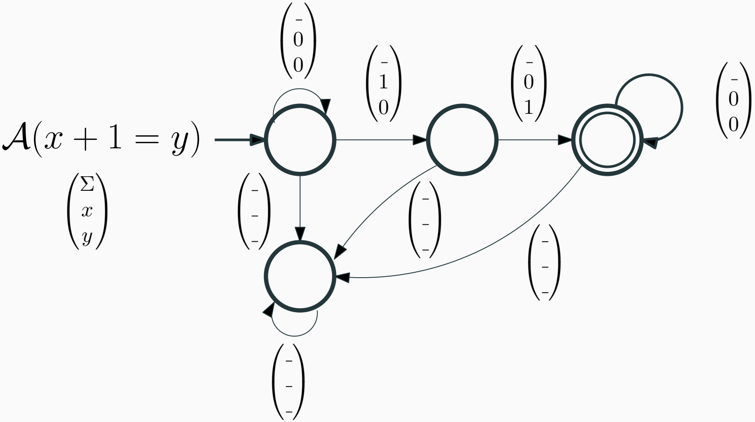
The proof by induction:

$$\mathcal{A}(x + 1 = y) \begin{pmatrix} \Sigma \\ x \\ y \end{pmatrix}$$

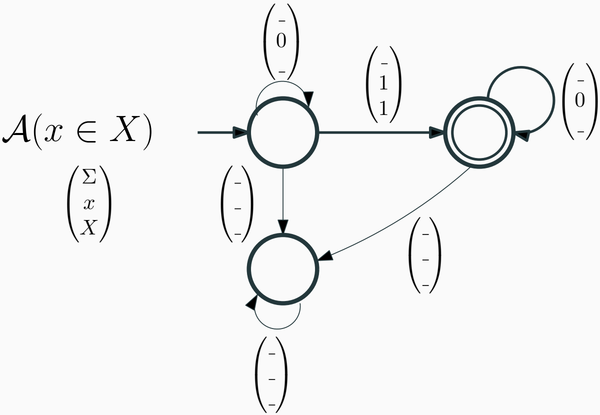
The proof by induction:



The proof by induction:



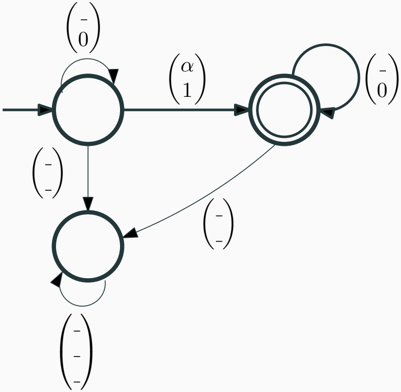
The proof by induction:



The proof by induction:

$\mathcal{A}(Q_\alpha(x))$

$\binom{\Sigma}{x}$



The proof by induction:

$$\mathcal{A}(\phi \wedge \psi) = \mathcal{A}(\mathcal{L}(\phi) \cap \mathcal{L}(\psi))$$

The proof by induction:

$\mathcal{A}(\neg\psi)$

The proof by induction:

$$\mathcal{A}(\psi) = (\Sigma, Q, \delta, q_0, F)$$

$$\mathcal{A}(\neg\psi) \;= \mathcal{A}(\overline{\mathcal{L}(\psi)}) = (\Sigma, Q, \delta, q_0, Q \setminus F)$$

The proof by induction:

$$\mathcal{A}(\exists X, \phi(X))$$

The proof by induction:

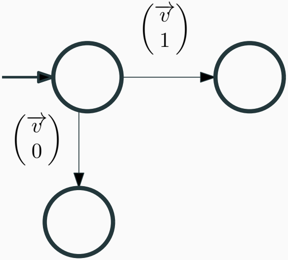
$$\mathcal{A}(\phi(X)) = (\Sigma, Q, \delta, q_0, F)$$

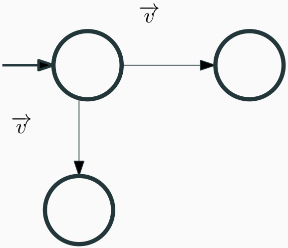
$$\mathcal{A}(\exists X, \phi(X))$$

The proof by induction:

$$\mathcal{A}(\phi(X)) = (\Sigma, Q, \delta, q_0, F)$$

$$\mathcal{A}(\exists X, \phi(X))$$



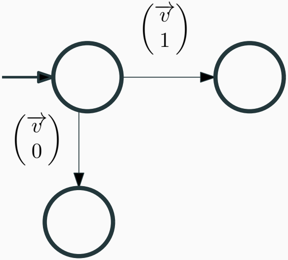


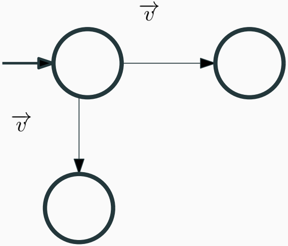
The proof by induction:

$$\mathcal{A}(\phi(X)) = (\Sigma, Q, \delta, q_0, F)$$

$$\mathcal{A}(\exists X, \phi(X)) \ = Determinize(\Sigma, Q, \delta', q_0, F)$$

Where for all $q \in Q$:





Some comments

Determinization can increase exponentially the size of the automaton

Some comments

Determinization can increase exponentially the size of the automaton

$$|\mathcal{A}(\psi)| \leq 2^{\underbrace{2^{2^{\cdots|\psi|}}}_{k \text{ times}}}$$

where k is the number of determinizations

Some comments

Determinization can increase exponentially the size of the automaton

$$|\mathcal{A}(\psi)| \leq 2^{\underbrace{2^{2^{\cdots|\psi|}}}_{k \text{ times}}}$$

where k is the number of determinizations $\approx$ "number of quantifier alternation"

Some comments

Determinization can increase exponentially the size of the automaton

$$|\mathcal{A}(\psi)| \leq 2^{\underbrace{2^{2^{\cdots|\psi|}}}_{k \text{ times}}}$$

where k is the number of determinizations $\approx$ "number of quantifier alternation"

Theorem (The other direction holds)

A language is regular, if and only if it is defined by an MSO formula.

Some comments

Determinization can increase exponentially the size of the automaton

$$|\mathcal{A}(\psi)| \leq 2^{\underbrace{2^{2^{\cdots|\psi|}}}_{k \text{ times}}}$$

where k is the number of determinizations $\approx$ "number of quantifier alternation"

Theorem (The other direction holds)

A language is regular, if and only if it is defined by an MSO formula.

Corollary (Presburger, 1929)

Presburger arithmetic is decidable.

Binary tree-automata and MSO on binary trees

An inductive view of automata

A word is either:

An inductive view of automata

A word is either:

- ε empty,

An inductive view of automata

A word is either:

- ε empty,
- $w\alpha$ for some word w and letter α.

An inductive view of automata

A word is either:

- ε empty,
- $w\alpha$ for some word w and letter α.

Given a DFA $(\Sigma, Q, \delta, q_0, F)$, let $\sigma : \Sigma^* \to Q$ be the function that computes the state of any given word then:

An inductive view of automata

A word is either:

- ε empty,
- $w\alpha$ for some word w and letter α.

Given a DFA $(\Sigma, Q, \delta, q_0, F)$, let $\sigma : \Sigma^* \to Q$ be the function that computes the state of any given word then:

- $\sigma(\varepsilon) = q_0$,

An inductive view of automata

A word is either:

- ε empty,
- $w\alpha$ for some word w and letter α.

Given a DFA $(\Sigma, Q, \delta, q_0, F)$, let $\sigma : \Sigma^* \to Q$ be the function that computes the state of any given word then:

- $\sigma(\varepsilon) = q_0$,
- $\sigma(wa) = \delta(\sigma(w), a)$.

An inductive view of automata

A word is either:

- ε empty,
- $w\alpha$ for some word w and letter α.

Given a DFA $(\Sigma, Q, \delta, q_0, F)$, let $\sigma : \Sigma^* \to Q$ be the function that computes the state of any given word then:

- $\sigma(\varepsilon) = q_0$,
- $\sigma(wa) = \delta(\sigma(w), a)$.

A word w is accepted iff $\sigma(w) \in F$.

LOB tree

A **labeled ordered binary tree** is either:

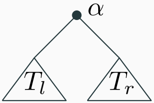
LOB tree

A **labeled ordered binary tree** is either:

- τ_0 the empty tree,

LOB tree

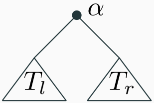
A **labeled ordered binary tree** is either:

- τ_0 the empty tree,
- 

for some letter α and trees T_l and T_r.

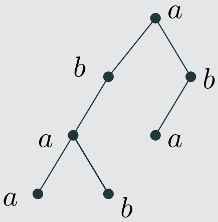
LOB tree

A **labeled ordered binary tree** is either:

- τ_0 the empty tree,
- 

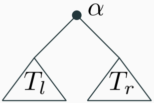
for some letter α and trees T_l and T_r.

Example



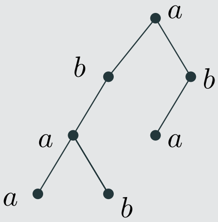
LOB tree

A **labeled ordered binary tree** is either:

- τ_0 the empty tree,
- 

  for some letter α and trees T_l and T_r.

Example



These two trees are different:

Deterministic binary-tree automaton

A DTFA $\mathcal{A}$ is a 5-tuple, $(\Sigma, Q, \delta, q_0, F)$, consisting of

Deterministic binary-tree automaton

A DTFA $\mathcal{A}$ is a 5-tuple, $(\Sigma, Q, \delta, q_0, F)$, consisting of

- an alphabet Σ, a finite set of states Q, an initial state $q_0 \in Q$, a set of accepting states $F \subseteq Q$,

Deterministic binary-tree automaton

A DTFA $\mathcal{A}$ is a 5-tuple, $(\Sigma, Q, \delta, q_0, F)$, consisting of

- an alphabet Σ, a finite set of states Q, an initial state $q_0 \in Q$, a set of accepting states $F \subseteq Q$,
- a transition function $\delta : Q \times Q \times \Sigma \to Q$.

Deterministic binary-tree automaton

A DTFA $\mathcal{A}$ is a 5-tuple, $(\Sigma, Q, \delta, q_0, F)$, consisting of

- an alphabet Σ, a finite set of states Q, an initial state $q_0 \in Q$, a set of accepting states $F \subseteq Q$,
- a transition function $\delta : Q \times Q \times \Sigma \to Q$.

Deterministic binary-tree automaton

A DTFA $\mathcal{A}$ is a 5-tuple, $(\Sigma, Q, \delta, q_0, F)$, consisting of

- an alphabet Σ, a finite set of states Q, an initial state $q_0 \in Q$, a set of accepting states $F \subseteq Q$,
- a transition function $\delta : Q \times Q \times \Sigma \to Q$.

Let σ be the function that computes the state of any given LOB-tree:

Deterministic binary-tree automaton

A DTFA $\mathcal{A}$ is a 5-tuple, $(\Sigma, Q, \delta, q_0, F)$, consisting of

- an alphabet Σ, a finite set of states Q, an initial state $q_0 \in Q$, a set of accepting states $F \subseteq Q$,
- a transition function $\delta : Q \times Q \times \Sigma \to Q$.

Let σ be the function that computes the state of any given LOB-tree:

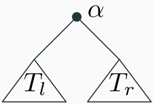
- $\sigma(\tau_0) = q_0$,

Deterministic binary-tree automaton

A DTFA $\mathcal{A}$ is a 5-tuple, $(\Sigma, Q, \delta, q_0, F)$, consisting of

- an alphabet Σ, a finite set of states Q, an initial state $q_0 \in Q$, a set of accepting states $F \subseteq Q$,
- a transition function $\delta : Q \times Q \times \Sigma \to Q$.

Let σ be the function that computes the state of any given LOB-tree:

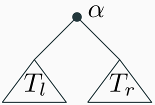
- $\sigma(\tau_0) = q_0$,
- $\sigma($ 

$) = \delta(\sigma(T_l), \sigma(T_r), \alpha)$

A DTFA $\mathcal{A}$ is a 5-tuple, $(\Sigma, Q, \delta, q_0, F)$, consisting of

- an alphabet Σ, a finite set of states Q, an initial state $q_0 \in Q$, a set of accepting states $F \subseteq Q$,
- a transition function $\delta : Q \times Q \times \Sigma \to Q$.

Let σ be the function that computes the state of any given LOB-tree:

- $\sigma(\tau_0) = q_0$,
- $\sigma($ 

$) = \delta(\sigma(T_l), \sigma(T_r), \alpha)$

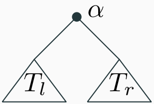
A tree $\mathcal{T}$ is accepted iff $\sigma(\mathcal{T}) \in F$.

Deterministic binary-tree automaton

A DTFA $\mathcal{A}$ is a 5-tuple, $(\Sigma, Q, \delta, q_0, F)$, consisting of

- an alphabet Σ, a finite set of states Q, an initial state $q_0 \in Q$, a set of accepting states $F \subseteq Q$,
- a transition function $\delta : Q \times Q \times \Sigma \to Q$.

Let σ be the function that computes the state of any given LOB-tree:

- $\sigma(\tau_0) = q_0$,
- $\sigma($ 

$) = \delta(\sigma(T_l), \sigma(T_r), \alpha)$

A tree $\mathcal{T}$ is accepted iff $\sigma(\mathcal{T}) \in F$.

Note: Testing if $\mathcal{T}$ is recognized by $\mathcal{A}$ is linear in $|\mathcal{T}|$.

MSO on binary trees - The symbols

The variables are vertices $(x, y, \dots)$ and sets of vertices $(X, Y, \dots)$.

MSO on binary trees - The symbols

The variables are vertices $(x, y, \ldots)$ and sets of vertices $(X, Y, \ldots)$.

For all letter α, we have a predicate Q_α which is interpreted as

$$Q_\alpha(x) := \text{“Vertex x is labelled } \alpha\text{”}$$

MSO on binary trees - The symbols

The variables are vertices $(x, y, \ldots)$ and sets of vertices $(X, Y, \ldots)$.

For all letter α, we have a predicate Q_α which is interpreted as

$$Q_\alpha(x) := \text{"Vertex x is labelled } \alpha\text{"}$$

For any vertex v different from τ_0,

- $v.l$:= left child of v,
- $v.r$:= right child of v,
- $v.p$:= parent of v (v itself if v is the root).

MSO on binary trees - The syntax

The syntax of *MSO* on LOB trees.

$$\begin{aligned}\phi := \quad & x = y.l \mid x = y.r \mid x = y.p \mid x = y \mid Q_\alpha(x) \mid x \in X \mid X = Y \\ & \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \implies \phi_2 \\ & \mid \forall x.\phi \mid \forall X.\phi \mid \exists x.\phi \mid \exists X.\phi.\end{aligned}$$

MSO on binary trees - The syntax

The syntax of *MSO* on LOB trees.

$$
\begin{aligned}
\phi := \quad & x = y.l \mid x = y.r \mid x = y.p \mid x = y \mid Q_\alpha(x) \mid x \in X \mid X = Y \\
& \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \implies \phi_2 \\
& \mid \forall x.\phi \mid \forall X.\phi \mid \exists x.\phi \mid \exists X.\phi.
\end{aligned}
$$

ex:

$$\psi(D) := \forall v, \exists d \in D, (v = d \vee v = d.p \vee d = v.p)$$

MSO on binary trees - The syntax

The syntax of *MSO* on LOB trees.

$$\begin{aligned}\phi := \quad & x = y.l \mid x = y.r \mid x = y.p \mid x = y \mid Q_\alpha(x) \mid x \in X \mid X = Y \\ & \mid \neg\phi \mid \phi_1 \land \phi_2 \mid \phi_1 \lor \phi_2 \mid \phi_1 \implies \phi_2 \\ & \mid \forall x.\phi \mid \forall X.\phi \mid \exists x.\phi \mid \exists X.\phi.\end{aligned}$$

ex:

$$\psi(D) := \forall v, \exists d \in D, (v = d \lor v = d.p \lor d = v.p)$$

"D is a dominating set"

MSO on binary trees - The syntax

The syntax of *MSO* on LOB trees.

$$\begin{aligned}\phi := \quad & x = y.l \mid x = y.r \mid x = y.p \mid x = y \mid Q_\alpha(x) \mid x \in X \mid X = Y \\ & \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \implies \phi_2 \\ & \mid \forall x.\phi \mid \forall X.\phi \mid \exists x.\phi \mid \exists X.\phi.\end{aligned}$$

ex:

$$\psi(D) := \forall v, \exists d \in D, (v = d \vee v = d.p \vee d = v.p)$$

"D is a dominating set"

MSO on binary trees - The syntax

The syntax of *MSO* on LOB trees.

$$
\begin{aligned}
\phi := \quad & x = y.l \mid x = y.r \mid x = y.p \mid x = y \mid Q_\alpha(x) \mid x \in X \mid X = Y \\
& \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \implies \phi_2 \\
& \mid \forall x.\phi \mid \forall X.\phi \mid \exists x.\phi \mid \exists X.\phi.
\end{aligned}
$$

ex:

$$\psi(D) := \forall v, \exists d \in D, (v = d \vee v = d.p \vee d = v.p)$$

"D is a dominating set"

MSO on binary trees - The syntax

The syntax of *MSO* on LOB trees.

$$\begin{aligned}\phi := \quad & x = y.l \mid x = y.r \mid x = y.p \mid x = y \mid Q_\alpha(x) \mid x \in X \mid X = Y \\ & \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \implies \phi_2 \\ & \mid \forall x.\phi \mid \forall X.\phi \mid \exists x.\phi \mid \exists X.\phi.\end{aligned}$$

ex:

$$\psi(D) := \forall v, \exists d \in D, (v = d \vee v = d.p \vee d = v.p)$$

"D is a dominating set"

Rabin's theorem

Theorem (Rabin, 1969)

For any MSO formula ϕ, there exists a DTFA $\mathcal{A}(\phi)$ that recognizes exactly the LOB trees accepted by ϕ.

Rabin's theorem

Theorem (Rabin, 1969)

For any MSO formula ϕ, there exists a DTFA $\mathcal{A}(\phi)$ that recognizes exactly the LOB trees accepted by ϕ.

Corollary

Every LOB tree property definable in MSOL can be decided in linear time on LOB trees.

Rabin's theorem

Theorem (Rabin, 1969)

For any MSO formula ϕ, there exists a DTFA $\mathcal{A}(\phi)$ that recognizes exactly the LOB trees accepted by ϕ.

Corollary

Every LOB tree property definable in MSOL can be decided in linear time on LOB trees.

Corollary

WS2S is decidable.

The proof

It uses exactly the same idea as for MSO on words and DFA. $\square$

Remark: Again, the only thing that really increases the size of the automaton is determinization.

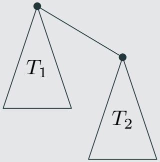
The final ingredient: interpretation

Encoding rooted trees in ordered binary trees

Definition

An rooted tree is either:

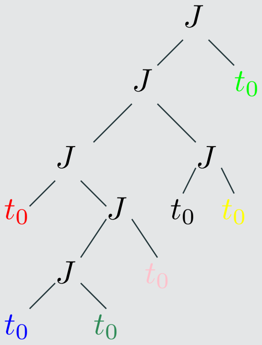
- a root and no other node (t_0),
- or the join $J(T_1, T_2)$ of two rooted trees T_1 and T_2.



Encoding rooted trees in ordered binary trees - example

Example

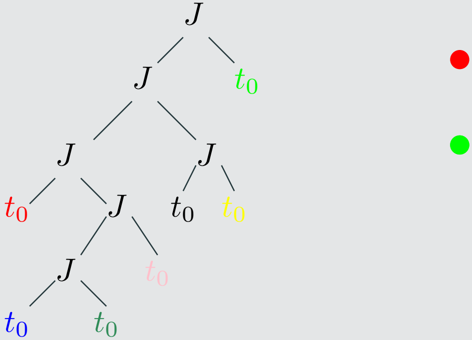
$J(J(J(t_0, J(J(t_0, t_0), t_0)), J(t_0, t_0)), t_0)$



Encoding rooted trees in ordered binary trees - example

Example

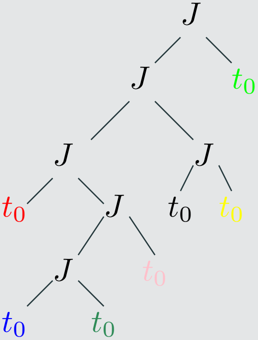
$J(J(J(t_0, J(J(t_0, t_0), t_0)), J(t_0, t_0)), t_0)$

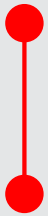


Encoding rooted trees in ordered binary trees - example

Example

$J(J(J(t_0, J(J(t_0, t_0), t_0)), J(t_0, t_0)), t_0)$

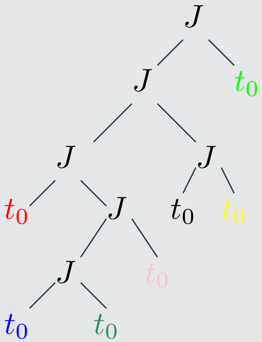


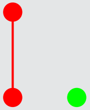


Encoding rooted trees in ordered binary trees - example

Example

$J(J(J(t_0, J(J(t_0, t_0), t_0)), J(t_0, t_0)), t_0)$

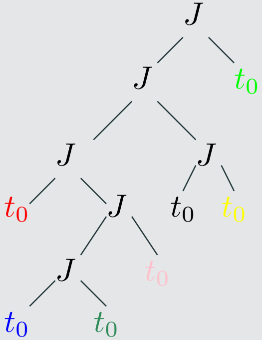


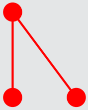


Encoding rooted trees in ordered binary trees - example

Example

$J(J(J(t_0, J(J(t_0, t_0), t_0)), J(t_0, t_0)), t_0)$

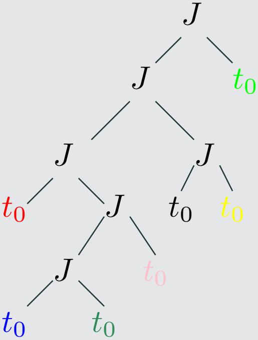




Encoding rooted trees in ordered binary trees - example

Example

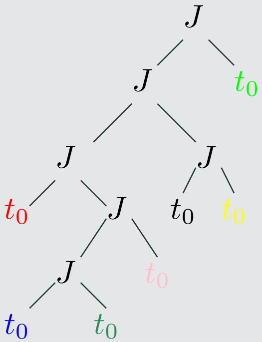
$J(J(J(t_0, J(J(t_0, t_0), t_0)), J(t_0, t_0)), t_0)$

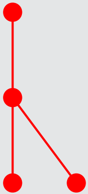


Encoding rooted trees in ordered binary trees - example

Example

$J(J(J(t_0, J(J(t_0, t_0), t_0)), J(t_0, t_0)), t_0)$

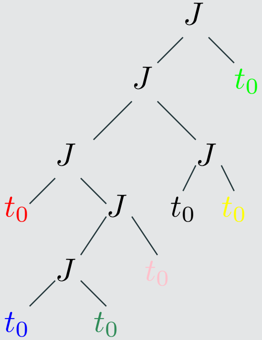


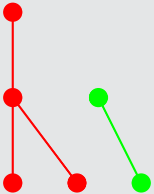


Encoding rooted trees in ordered binary trees - example

Example

$J(J(J(t_0, J(J(t_0, t_0), t_0)), J(t_0, t_0)), t_0)$

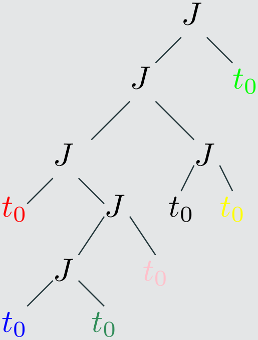


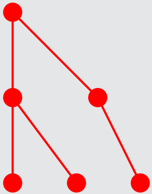


Encoding rooted trees in ordered binary trees - example

Example

$J(J(J(t_0, J(J(t_0, t_0), t_0)), J(t_0, t_0)), t_0)$

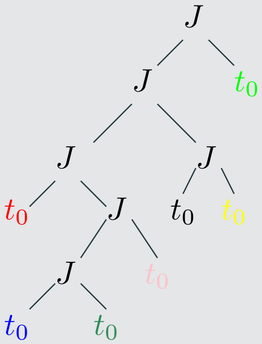


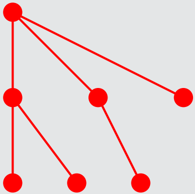


Encoding rooted trees in ordered binary trees - example

Example

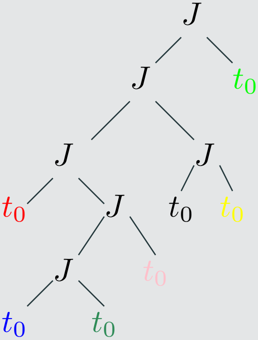
$J(J(J(t_0, J(J(t_0, t_0), t_0)), J(t_0, t_0)), t_0)$

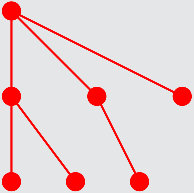




Example

$J(J(J(t_0, J(J(t_0, t_0), t_0)), J(t_0, t_0)), t_0)$

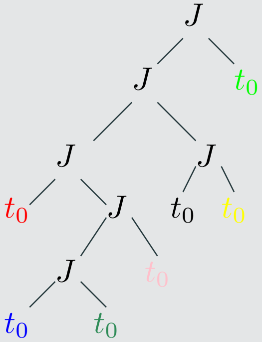


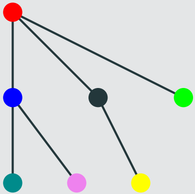


Encoding rooted trees in ordered binary trees - example

Example

$J(J(J(t_0, J(J(t_0, t_0), t_0)), J(t_0, t_0)), t_0)$

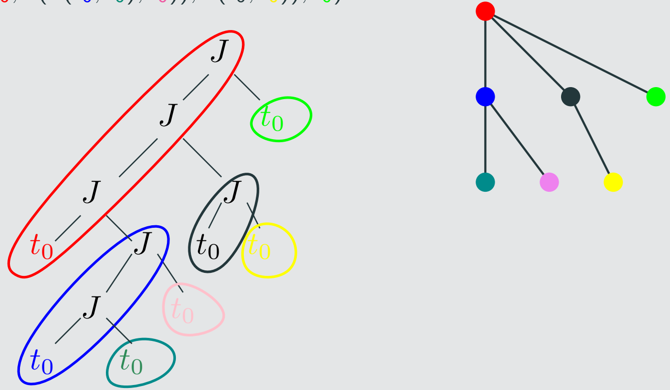




Encoding rooted trees in ordered binary trees - example

Example

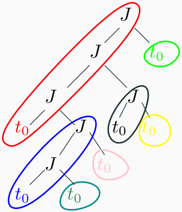
$J(J(J(t_0, J(J(t_0, t_0), t_0)), J(t_0, t_0)), t_0)$



Encoding the logical formula

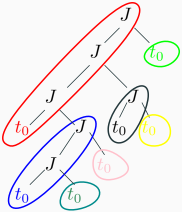
We have a way to "encode" trees into binary trees.

How to translate $\underbrace{\text{MSO}}_{\text{graph}}$ properties of trees into $\underbrace{\text{MSO}}_{\text{LOB tree}}$ properties of LOB trees ?



Encoding the logical formula

We have a way to "encode" trees into binary trees.



How to translate $\underbrace{\text{MSO}}_{\text{graph}}$ properties of trees into $\underbrace{\text{MSO}}_{\text{LOB tree}}$ properties of LOB trees ?

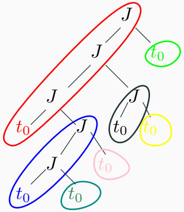
Let v, u be vertices and v', u' the corresponding leaves in the LOB tree.

$$Adj(u, v) \iff \underbrace{v' \text{ and } u' \text{ have two adjacents right ancestors } v'' \text{ and } u''}$$

Encoding the logical formula

We have a way to "encode" trees into binary trees.

How to translate $\underbrace{\text{MSO}}_{\text{graph}}$ properties of trees into $\underbrace{\text{MSO}}_{\text{LOB tree}}$ properties of LOB trees ?



Let v, u be vertices and v', u' the corresponding leaves in the LOB tree.

$$Adj(u, v) \iff \underbrace{v' \text{ and } u' \text{ have two adjacents right ancestors } v'' \text{ and } u''}_{\text{In MSO over the LOB tree ?}}$$

Adjacency

$Adj(u, v) \iff v'$ and u' have two adjacents right ancestors v'' and u''

We define:

Adjacency

$$Adj(u,v) \iff v' \text{ and } u' \text{ have two adjacents right ancestors } v'' \text{ and } u''$$

We define:

$$LCC(X) := \underbrace{\forall x, y : (x \in X \land y = x.l) \implies y \in X}$$

Adjacency

$Adj(u, v) \iff v'$ and u' have two adjacents right ancestors v'' and u''

We define:

$$LCC(X) := \underbrace{\forall x, y : (x \in X \wedge y = x.l) \implies y \in X}_{X \text{ is left child closed}}$$

Adjacency

$$Adj(u,v) \iff v' \text{ and } u' \text{ have two adjacents right ancestors } v'' \text{ and } u''$$

We define:

$$LCC(X) := \underbrace{\forall x, y : (x \in X \wedge y = x.l) \implies y \in X}_{X \text{ is left child closed}}$$

$$RA(x,y) := \underbrace{\forall A : (LCC(A) \wedge y \in A) \implies x \in A}$$

Adjacency

$$Adj(u,v) \iff v' \text{ and } u' \text{ have two adjacents right ancestors } v'' \text{ and } u''$$

We define:

$$LCC(X) := \underbrace{\forall x, y : (x \in X \land y = x.l) \implies y \in X}_{X \text{ is left child closed}}$$

$$RA(x,y) := \underbrace{\forall A : (LCC(A) \land y \in A) \implies x \in A}_{y \text{ is a right ancestor of } x}$$

Adjacency

$Adj(u, v) \iff$ v' and u' have two adjacents right ancestors v'' and u''

We define:

$$LCC(X) := \underbrace{\forall x, y : (x \in X \land y = x.l) \implies y \in X}_{X \text{ is left child closed}}$$

$$RA(x, y) := \underbrace{\forall A : (LCC(A) \land y \in A) \implies x \in A}_{y \text{ is a right ancestor of } x}$$

$$\begin{aligned} Edge(u', v') := \underbrace{\exists u'', v'' : RA(u', u'') \land RA(v', v'') \\ \land (u'' = v''.r \lor v'' = u''.r)} \end{aligned}$$

Adjacency

$$Adj(u, v) \iff v' \text{ and } u' \text{ have two adjacents right ancestors } v'' \text{ and } u''$$

We define:

$$LCC(X) := \underbrace{\forall x, y : (x \in X \land y = x.l) \implies y \in X}_{X \text{ is left child closed}}$$

$$RA(x, y) := \underbrace{\forall A : (LCC(A) \land y \in A) \implies x \in A}_{y \text{ is a right ancestor of } x}$$

$$\begin{aligned} Edge(u', v') := \exists u'', v'' : \; & RA(u', u'') \land RA(v', v'') \\ & \underbrace{\land (u'' = v''.r \lor v'' = u''.r)}_{v' \text{ and } u' \text{ have two adjacents right ancestors } v'' \text{ and } u''} \end{aligned}$$

Adjacency

$$Adj(u,v) \iff v' \text{ and } u' \text{ have two adjacents right ancestors } v'' \text{ and } u''$$

We define:

$$LCC(X) := \underbrace{\forall x, y : (x \in X \wedge y = x.l) \implies y \in X}_{X \text{ is left child closed}}$$

$$RA(x,y) := \underbrace{\forall A : (LCC(A) \wedge y \in A) \implies x \in A}_{y \text{ is a right ancestor of } x}$$

$$Edge(u',v') := \underbrace{\exists u'', v'' : RA(u',u'') \wedge RA(v',v'') \wedge (u'' = v''.r \vee v'' = u''.r)}_{v' \text{ and } u' \text{ have two adjacents right ancestors } v'' \text{ and } u''}$$

$$Adj(u,v) \iff Edge(u',v')$$

Quantification

A vertex from a tree correspond to a leaf from the correponding LOB tree

Quantification

A vertex from a tree correspond to a leaf from the correponding LOB tree

If $\phi(x)$ is an MSO formula over graphs and $\phi'(x)$ is the equivalent MSO formula over LOB trees

$$\overbrace{\exists x, \phi(x)}^{\text{graph MSO}} \iff \overbrace{\exists x, (\phi'(x) \wedge (x \text{ is a leaf}))}^{\text{LOB tree MSO}}$$

Quantification

A vertex from a tree correspond to a leaf from the correponding LOB tree

If $\phi(x)$ is an MSO formula over graphs and $\phi'(x)$ is the equivalent MSO formula over LOB trees

$$\overbrace{\exists x, \phi(x)}^{\text{graph MSO}} \iff \overbrace{\exists x, (\phi'(x) \wedge (x \text{ is a leaf}))}^{\text{LOB tree MSO}}$$

$$\mathit{Leaf}(x) := \forall y : \neg(y = x.l) \wedge \neg(y = x.r)$$

Quantification

A vertex from a tree correspond to a leaf from the correponding LOB tree

If $\phi(x)$ is an MSO formula over graphs and $\phi'(x)$ is the equivalent MSO formula over LOB trees

$$\overbrace{\exists x, \phi(x)}^{\text{graph MSO}} \iff \overbrace{\exists x, (\phi'(x) \land (x \text{ is a leaf}))}^{\text{LOB tree MSO}}$$

$$\mathit{Leaf}(x) := \forall y : \neg(y = x.l) \land \neg(y = x.r)$$

$$\mathit{Leaves}(X) := \forall x : x \in X \implies \mathit{leaf}(x)$$

The interpretation

We have:

- a map $\mathcal{D}$ from trees to LOB trees,
- a map $\mathcal{I}$ from MSO formula on graphs to MSO on LOB trees

such that

The interpretation

We have:

- a map $\mathcal{D}$ from trees to LOB trees,
- a map $\mathcal{I}$ from MSO formula on graphs to MSO on LOB trees

such that

for any tree $\mathcal{T}$ and MSO formula ψ:

$$\mathcal{T} \models \psi \qquad \iff \qquad \mathcal{D}(\mathcal{T}) \models \mathcal{I}(\psi)$$

$\mathcal{D}$ and $\mathcal{I}$ can be computed in linear time.

Corollary

Given any tree $\mathcal{T}$ and MSO formula ψ, one can decide $\mathcal{T} \models \psi$ in time $f(|\psi|) \cdot |\mathcal{T}|$.

Cographs

A cograph is either:

$$\begin{aligned} s_0 &\;\; \text{a single vertex graph,} \\ G \cup H &: \text{the disjoint union of two cographs,} \\ J(G, H) &: \text{the join of two cographs.} \end{aligned}$$

Cographs

A cograph is either:

s_0 a single vertex graph,

$G \cup H$: the disjoint union of two cographs,

$J(G, H)$: the join of two cographs.

a b c d e f g

a b c d e f g

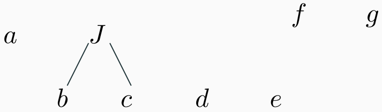
Cographs

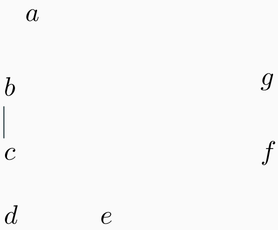
A cograph is either:

s_0 a single vertex graph,

$G \cup H$: the disjoint union of two cographs,

$J(G, H)$: the join of two cographs.





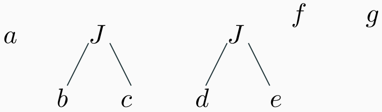
Cographs

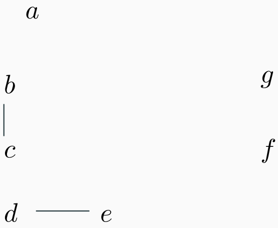
A cograph is either:

s_0 a single vertex graph,

$G \cup H$: the disjoint union of two cographs,

$J(G, H)$: the join of two cographs.





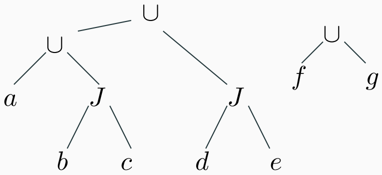
Cographs

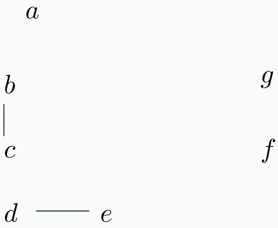
A cograph is either:

s_0 a single vertex graph,

$G \cup H$: the disjoint union of two cographs,

$J(G, H)$: the join of two cographs.





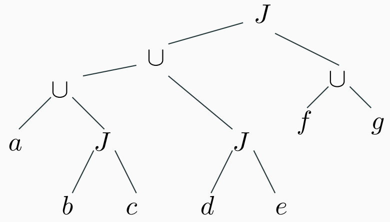
Cographs

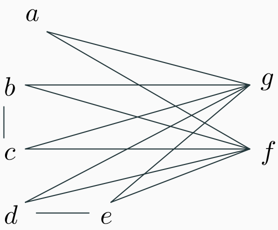
A cograph is either:

s_0 a single vertex graph,

$G \cup H$: the disjoint union of two cographs,

$J(G, H)$: the join of two cographs.





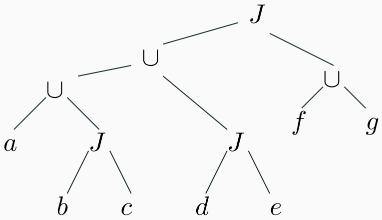
Cographs

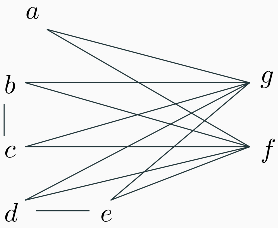
A cograph is either:

s_0 a single vertex graph,

$G \cup H$: the disjoint union of two cographs,

$J(G, H)$: the join of two cographs.





Two vertices are adjacent iff their lowest common ancestor is J

MSO : is LCC in J

$$\text{ancestor}(v, a) :=$$

$$\forall C, (a \in C \ \land \underbrace{(\forall x, x \in C \implies (x.l \in C \land x.r \in C)))}_{C \text{ is children closed}} \implies v \in C$$

MSO : is LCC in J

$$\text{ancestor}(v, a) :=$$
$$\forall C, (a \in C \ \wedge \underbrace{(\forall x, x \in C \implies (x.l \in C \wedge x.r \in C)))}_{C \text{ is children closed}} \implies v \in C$$

$$\begin{aligned} LCA(l_1, l_2, a) := &\ \text{ancestor}(l_1, a) \wedge \text{ancestor}(l_2, a) \\ &\wedge \forall x : (\text{ancestor}(l_1, x) \wedge \text{ancestor}(l_2, x)) \implies \text{ancestor}(a, x) \end{aligned}$$

MSO : is LCC in J

$$\text{ancestor}(v, a) :=$$
$$\forall C, (a \in C \ \wedge \underbrace{(\forall x, x \in C \implies (x.l \in C \wedge x.r \in C)))}_{C \text{ is children closed}} \implies v \in C$$

$$\begin{aligned} LCA(l_1, l_2, a) := &\ \text{ancestor}(l_1, a) \wedge \text{ancestor}(l_2, a) \\ &\wedge \forall x : (\text{ancestor}(l_1, x) \wedge \text{ancestor}(l_2, x)) \implies \text{ancestor}(a, x) \end{aligned}$$

$$Edge(l_1, l_2) := \exists a, LCA(l_1, l_2, a) \wedge Q_J(a)$$

The interpretation: cographs

We have:

- a map $\mathcal{D}$ from cographs to LOB trees,
- a map $\mathcal{I}$ from MSO formula on graphs to MSO on LOB trees

such that

The interpretation: cographs

We have:

- a map $\mathcal{D}$ from cographs to LOB trees,
- a map $\mathcal{I}$ from MSO formula on graphs to MSO on LOB trees

such that

for any cograph G and MSO formula ψ:

$$G \models \psi \qquad \Longleftrightarrow \qquad \mathcal{D}(G) \models \mathcal{I}(\psi)$$

$\mathcal{D}$ and $\mathcal{I}$ can be computed in linear time (non-trivial for $\mathcal{D}$, cf. *modular tree decomposition*).

The interpretation: cographs

We have:

- a map $\mathcal{D}$ from cographs to LOB trees,
- a map $\mathcal{I}$ from MSO formula on graphs to MSO on LOB trees

such that

for any cograph G and MSO formula ψ:

$$G \models \psi \qquad \iff \qquad \mathcal{D}(G) \models \mathcal{I}(\psi)$$

$\mathcal{D}$ and $\mathcal{I}$ can be computed in linear time (non-trivial for $\mathcal{D}$, cf. *modular tree decomposition*).

Corollary

Given any cograph G and MSO formula ψ, one can decide $G \models \psi$ in time linear in $|G|$ (but non elementary in $|\psi|$).

Bounded clique-width

clique-width $\leq 2 \iff$ **cograph**

Bounded clique-width

clique-width $\leq 2 \iff$ **cograph**

A graph with k labels has clique-width at most k if it is:

- a single vertex with label $i \in \{1, \dots, k\}$,
- $G \cup H$, where G and H are two labeled graphs of clique-width $\leq k$
- obtained by adding all the possible edges between two label classes in a labeled graph G of clique-width at most k
- obtained by renaming a label class with another label in a labeled graph G of clique-width at most k.

Bounded clique-width

clique-width $\leq 2 \iff$ **cograph**

A graph with k labels has clique-width at most k if it is:

- a single vertex with label $i \in \{1, \ldots, k\}$,
- $G \cup H$, where G and H are two labeled graphs of clique-width $\leq k$
- obtained by adding all the possible edges between two label classes in a labeled graph G of clique-width at most k
- obtained by renaming a label class with another label in a labeled graph G of clique-width at most k.

The decomposition is (almost) given by the definition (and can be computed in linear time).

Bounded clique-width

clique-width $\leq 2 \iff$ **cograph**

A graph with k labels has clique-width at most k if it is:

- a single vertex with label $i \in \{1, \ldots, k\}$,
- $G \cup H$, where G and H are two labeled graphs of clique-width $\leq k$
- obtained by adding all the possible edges between two label classes in a labeled graph G of clique-width at most k
- obtained by renaming a label class with another label in a labeled graph G of clique-width at most k.

The decomposition is (almost) given by the definition (and can be computed in linear time).

The translation of the MSO formula uses the same idea as for cographs.

Corollary

Given any graph G of clique-width at most k and MSO formula ψ, one can decide $G \models \psi$ in time in $f(|\psi|, |k|) \cdot O(|G|)$.

Tree-automata: beyond model checking

Other use of tree-automaton

Tree automaton can be used for other purposes than model checking.

Other use of tree-automaton

Tree automaton can be used for other purposes than model checking.

Other use of tree-automaton

Tree automaton can be used for other purposes than model checking.

For any MSO formula $\Psi(S)$, we let λ_Ψ be the smallest value such that

Theorem template - 1

For any tree T, the number of sets S satisfying Ψ is in $O((\lambda_\Psi)^{|T|})$.

Meta-Theorem, Rosenfeld, 2021

There exists an algorithm for the following problem

Input: An MSO formula $\Psi(S)$ and a real $\varepsilon > 0$

Output: $\lambda \in \mathbb{Q}$ such that $|\lambda - \lambda_\Psi| < \varepsilon$

Other results (Rosenfeld, SODA 2021)

Familly	λ_Ψ	Comments
Independent dominating sets	$\sqrt{2}$	
Total perfect dominating	$(2^{27} \times 7)^{\frac{1}{85}} \approx 1.2751$	
Induced matchings	≈ 1.46557	root of $x^3 - x^2 - 1$
Perfect codes	$3^{\frac{1}{7}} \approx 1.16993$	
Minimal perfect dominating	≈ 1.32472	root of $x^3 - x - 1$
Maximal matchings	$\left(\frac{11+\sqrt{85}}{2}\right)^{\frac{1}{7}} \approx 1.3917$	
3-matchings	≈ 1.3802	root of $x^4 - x^3 - 1$
4-matchings	$13^{\frac{1}{9}} \approx 1.329754$	
5-matchings	$1.2932 \leq ? \leq 1.2941$	
Maximal induced matchings	≈ 1.331576868	imprecision of 10^{-40}
Maximal irredundant sets	$1.537 \leq ? \leq 1.556$	

Other results (Rosenfeld, SODA 2021)

Familly	λ_Ψ	Comments
Independent dominating sets	$\sqrt{2}$	
Total perfect dominating	$(2^{27} \times 7)^{\frac{1}{85}} \approx 1.2751$	
Induced matchings	≈ 1.46557	root of $x^3 - x^2 - 1$
Perfect codes	$3^{\frac{1}{7}} \approx 1.16993$	
Minimal perfect dominating	≈ 1.32472	root of $x^3 - x - 1$
Maximal matchings	$\left(\frac{11+\sqrt{85}}{2}\right)^{\frac{1}{7}} \approx 1.3917$	
3-matchings	≈ 1.3802	root of $x^4 - x^3 - 1$
4-matchings	$13^{\frac{1}{9}} \approx 1.329754$	
5-matchings	$1.2932 \leq ? \leq 1.2941$	
Maximal induced matchings	≈ 1.331576868	imprecision of 10^{-40}
Maximal irredundant sets	$1.537 \leq ? \leq 1.556$	

Thanks !