Lazy Automata Techniques for WS1S

(TACAS'17)

Tomáš Fiedor[1,2] Lukáš Holík[2] Petr Janků[2]

[1]Red Hat, Czech Republic

Ondřej Lengál[2,3] Tomáš Vojnar[2]

[2]Brno University of Technology, Czech Republic

[3]Academia Sinica, Taiwan

MOSCA'19

WS1S

- weak monadic second-order logic of one successor
  - second-order ⇒ quantification over relations;
  - monadic ⇒ relations are unary (i.e. sets);
  - weak ⇒ sets are finite;
  - of one successor ⇒ reasoning about linear structures.

WS1S

- weak monadic second-order logic of one successor
  - second-order ⇒ quantification over relations;
  - monadic ⇒ relations are unary (i.e. sets);
  - weak ⇒ sets are finite;
  - of one successor ⇒ reasoning about linear structures.

- corresponds to finite automata [Büchi'60]

WS1S

- weak monadic second-order logic of one successor
  - second-order ⇒ quantification over relations;
  - monadic ⇒ relations are unary (i.e. sets);
  - weak ⇒ sets are finite;
  - of one successor ⇒ reasoning about linear structures.

- corresponds to finite automata [Büchi'60]

- decidable — but **NONELEMENTARY**
  - constructive proof via translation to finite automata

Application of WS1S

- allows one to define rich invariants

Application of WS1S

- allows one to define rich invariants
- used in tools for checking structural invariants
  - Pointer Assertion Logic Engine (`PALE`)
  - STRucture ANd Data (`STRAND`)
  - Unbounded Arrays Bounded Elements (`UABE`)

Application of WS1S

- allows one to define rich invariants
- used in tools for checking structural invariants
  - Pointer Assertion Logic Engine (`PALE`)
  - STRucture ANd Data (`STRAND`)
  - Unbounded Arrays Bounded Elements (`UABE`)
- many other applications
  - program and protocol verifications, linguistics, theorem provers …

Application of WS1S

- allows one to define rich invariants
- used in tools for checking structural invariants
  - Pointer Assertion Logic Engine (`PALE`)
  - STRucture ANd Data (`STRAND`)
  - Unbounded Arrays Bounded Elements (`UABE`)
- many other applications
  - program and protocol verifications, linguistics, theorem provers . . .
- decision procedure: the well-known MONA tool
  - sometimes efficient in practice
  - other times the complexity strikes back (unavoidable in general)
  - we try to push the usability border further

WS1S

- Syntax:
  - term ψ ::= $X \subseteq Y \quad | \quad \mathrm{Sing}(X) \quad | \quad X = \{0\} \quad | \quad X = \sigma(Y)$

WS1S

- Syntax:
  - term ψ ::= $X \subseteq Y \quad | \quad \mathrm{Sing}(X) \quad | \quad X = \{0\} \quad | \quad X = \sigma(Y)$
  - formula φ ::= $\psi \quad | \quad \varphi \wedge \varphi \quad | \quad \varphi \vee \varphi \quad | \quad \neg\varphi \quad | \quad \exists X.\varphi$

WS1S

- Syntax:
  - term ψ ::= $X \subseteq Y \quad | \quad \mathrm{Sing}(X) \quad | \quad X = \{0\} \quad | \quad X = \sigma(Y)$
  - formula φ ::= $\psi \quad | \quad \varphi \land \varphi \quad | \quad \varphi \lor \varphi \quad | \quad \neg\varphi \quad | \quad \exists X.\varphi$
- Interpretation: over finite subsets of $\mathbb{N}$
  - models of formulae = assignments of finite sets to variables
- sets can be encoded as finite binary strings:
  - $$\{1,4,5\} \to \begin{array}{llcl|lcl|ll} \text{Index:} & 012345 & & 012345 & 6 & & 012345 & 67 & \\ \text{Membership:} & \mathrm{x}\checkmark\mathrm{xx}\checkmark\checkmark & , & \mathrm{x}\checkmark\mathrm{xx}\checkmark\checkmark & \mathrm{x} & \text{or} & \mathrm{x}\checkmark\mathrm{xx}\checkmark\checkmark & \mathrm{xx} & \ldots \\ \text{Encoding:} & 010011 & & 010011 & 0 & & 010011 & 00 & \end{array}$$

WS1S

- Syntax:
  - term ψ ::= $X \subseteq Y \quad | \quad \mathrm{Sing}(X) \quad | \quad X = \{0\} \quad | \quad X = \sigma(Y)$
  - formula φ ::= $\psi \quad | \quad \varphi \wedge \varphi \quad | \quad \varphi \vee \varphi \quad | \quad \neg\varphi \quad | \quad \exists X.\varphi$
- Interpretation: over finite subsets of $\mathbb{N}$
  - models of formulae = assignments of finite sets to variables
- sets can be encoded as finite binary strings:
  - $\{1,4,5\} \rightarrow \begin{array}{lccccccc} \text{Index:} & 012345 & & 012345|6 & & 012345|67 & \\ \text{Membership:} & \text{x✓xx✓✓} & , & \text{x✓xx✓✓}|\text{x} & \text{or} & \text{x✓xx✓✓}|\text{xx} & \ldots \\ \text{Encoding:} & 010011 & & 010011|0 & & 010011|00 & \end{array}$
- Language interpretation $L(\varphi)$:
  - **Alphabet**: for each variable, we have one track in the alphabet
    - e.g. $\begin{array}{r} X: \\ Y: \end{array}\begin{bmatrix} 0 \\ 0 \end{bmatrix}$ is a symbol

WS1S

- Syntax:
  - term ψ ::= $X \subseteq Y \;\mid\; \mathrm{Sing}(X) \;\mid\; X = \{0\} \;\mid\; X = \sigma(Y)$
  - formula φ ::= $\psi \;\mid\; \varphi \wedge \varphi \;\mid\; \varphi \vee \varphi \;\mid\; \neg\varphi \;\mid\; \exists X.\varphi$
- Interpretation: over finite subsets of $\mathbb{N}$
  - models of formulae = assignments of finite sets to variables
- sets can be encoded as finite binary strings:
  - $\{1, 4, 5\} \rightarrow$

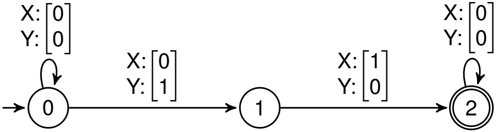
Index:	012345		012345 | 6		012345 | 67	
Membership:	x✓xx✓✓	,	x✓xx✓✓ | x	or	x✓xx✓✓ | xx	. . .
Encoding:	010011		010011 | 0		010011 | 00	

- Language interpretation $L(\varphi)$:
  - **Alphabet**: for each variable, we have one track in the alphabet
    - e.g. $\begin{array}{r}X{:}\\Y{:}\end{array}\begin{bmatrix}0\\0\end{bmatrix}$ is a symbol
  - **Models** are represented as a stack of (0-padded) binary strings
  - Example:

$$\{X \mapsto \emptyset, Y \mapsto \{2,4\}\} \models \varphi \quad \text{iff} \quad \begin{array}{r}X{:}\\Y{:}\end{array}\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} \in L(\varphi)$$

Deciding WS1S using automata

- example of base automaton for $X = \sigma(Y)$ (successor)

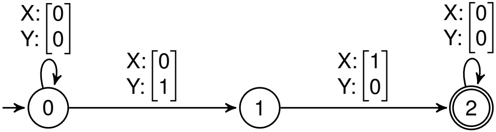


- Example:

$$\neg(X \subseteq Y) \wedge \Big(\mathrm{Sing}(Z) \vee \exists W . W = \sigma(Z)\Big)$$

Deciding WS1S using automata

- example of base automaton for $X = \sigma(Y)$ (successor)

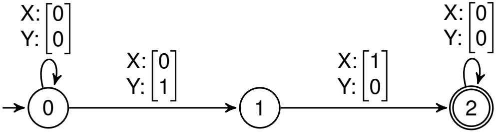


- Example:

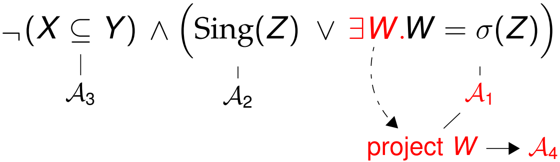
$$\neg\underset{\mathcal{A}_3}{(X \subseteq Y)} \wedge \Big(\underset{\mathcal{A}_2}{\mathrm{Sing}(Z)} \vee \exists W.\underset{\mathcal{A}_1}{W = \sigma(Z)}\Big)$$

Deciding WS1S using automata

- example of base automaton for $X = \sigma(Y)$ (successor)



- Example:

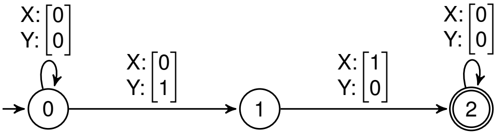


$$\neg(X \subseteq Y) \land \Big(\mathrm{Sing}(Z) \lor \exists W. W = \sigma(Z)\Big)$$

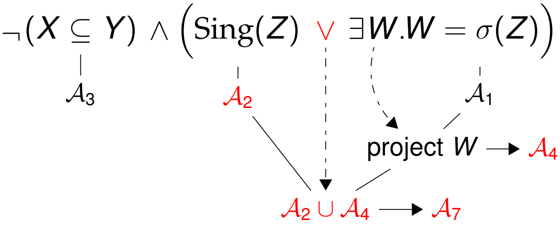
project W: $\begin{array}{r}\cancel{W}:\\ Z:\end{array}\begin{bmatrix}\cancel{0}\\ 1\end{bmatrix} \mapsto Z\colon [1]$

Deciding WS1S using automata

- example of base automaton for $X = \sigma(Y)$ (successor)



- Example:



$$\neg(X \subseteq Y) \wedge \Big(\mathrm{Sing}(Z) \vee \exists W. W = \sigma(Z)\Big)$$

$\mathcal{A}_3$

$\mathcal{A}_2$

$\mathcal{A}_1$

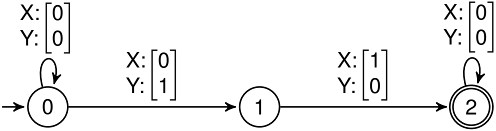
project $W \longrightarrow \mathcal{A}_4$

$\mathcal{A}_2 \cup \mathcal{A}_4 \longrightarrow \mathcal{A}_7$

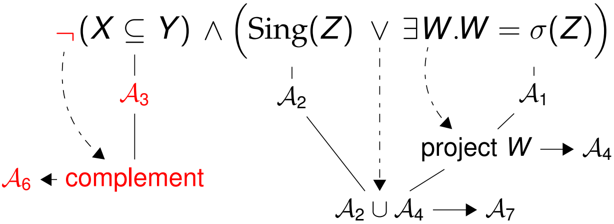
project W: $\begin{array}{l} \cancel{W:} \\ Z: \end{array} \begin{bmatrix} \cancel{0} \\ 1 \end{bmatrix} \mapsto Z: [1]$

Deciding WS1S using automata

- example of base automaton for $X = \sigma(Y)$ (successor)



- Example:



$$\neg(X \subseteq Y) \land \left(\mathrm{Sing}(Z) \lor \exists W. W = \sigma(Z)\right)$$

$\mathcal{A}_3$ $\mathcal{A}_2$ $\mathcal{A}_1$

project $W \longrightarrow \mathcal{A}_4$

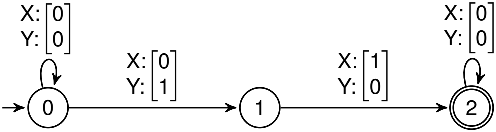
$\mathcal{A}_6 \longleftarrow$ complement

$\mathcal{A}_2 \cup \mathcal{A}_4 \longrightarrow \mathcal{A}_7$

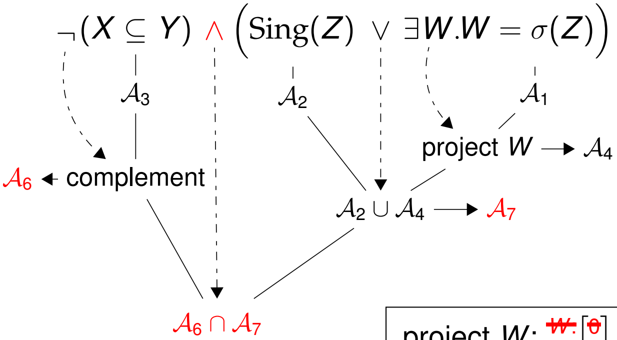
project W: $\begin{array}{l}\cancel{W}:\\ Z:\end{array}\begin{bmatrix}\cancel{0}\\ 1\end{bmatrix} \mapsto Z: [1]$

Deciding WS1S using automata

- example of base automaton for $X = \sigma(Y)$ (successor)



- Example:



$$\neg(X \subseteq Y) \wedge \Big(\mathrm{Sing}(Z) \vee \exists W.W = \sigma(Z)\Big)$$

$\mathcal{A}_3$

$\mathcal{A}_2$

$\mathcal{A}_1$

project $W \longrightarrow \mathcal{A}_4$

$\mathcal{A}_6 \leftarrow$ complement

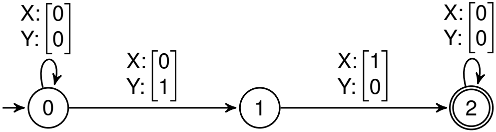
$\mathcal{A}_2 \cup \mathcal{A}_4 \longrightarrow \mathcal{A}_7$

$\mathcal{A}_6 \cap \mathcal{A}_7$

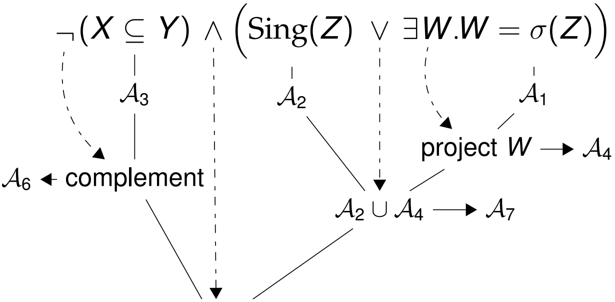
project W: $\begin{array}{l}\cancel{W}: \\ Z:\end{array}\begin{bmatrix}\cancel{0}\\1\end{bmatrix} \mapsto Z: [1]$

Deciding WS1S using automata

- example of base automaton for $X = \sigma(Y)$ (successor)



- Example:



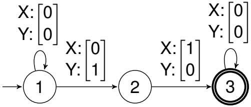
$$\neg(X \subseteq Y) \wedge \Big(\mathrm{Sing}(Z) \vee \exists W.W = \sigma(Z)\Big)$$

$L(\mathcal{A}_6 \cap \mathcal{A}_7) \overset{?}{=} \emptyset$

project W: $\begin{matrix}\cancel{W}: \\ Z:\end{matrix}\begin{bmatrix}\cancel{0} \\ 1\end{bmatrix} \mapsto Z: [1]$

How to handle quantification

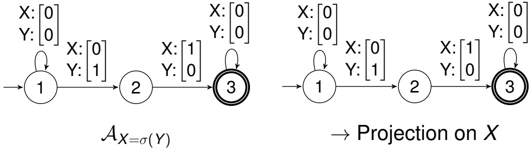
- issue with projection (existential quantification)
  - after removing of the tracks not all models would be accepted (problem with 0-padding)
    - needed for **soundness**!
    - for every assignment, it is necesssary to accept all or none encodings
  - so after projection we need to adjust the final states by saturation
    - pump the final states with all states backward reachable with 0
- consider $\exists X. X = \sigma(Y)$



$\mathcal{A}_{X=\sigma(Y)}$

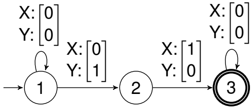
How to handle quantification

- issue with projection (existential quantification)
  - after removing of the tracks not all models would be accepted (problem with 0-padding)
    - needed for **soundness**!
    - for every assignment, it is necesssary to accept all or none encodings
  - so after projection we need to adjust the final states by saturation
    - pump the final states with all states backward reachable with 0
- consider $\exists X . X = \sigma(Y)$

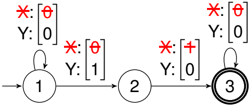


How to handle quantification

- issue with projection (existential quantification)
  - after removing of the tracks not all models would be accepted (problem with 0-padding)
    - needed for **soundness**!
    - for every assignment, it is necesssary to accept all or none encodings
  - so after projection we need to adjust the final states by saturation
    - pump the final states with all states backward reachable with 0
- consider $\exists X. X = \sigma(Y)$



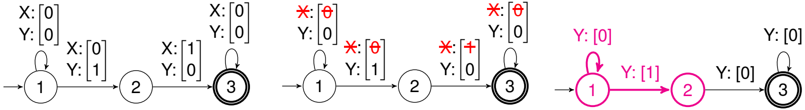
$\mathcal{A}_{X=\sigma(Y)}$



$\rightarrow$ Projection on X

How to handle quantification

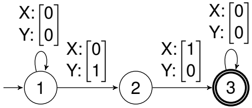
- issue with projection (existential quantification)
  - after removing of the tracks not all models would be accepted (problem with 0-padding)
    - needed for **soundness**!
    - for every assignment, it is necesssary to accept all or none encodings
  - so after projection we need to adjust the final states by saturation
    - pump the final states with all states backward reachable with 0
- consider $\exists X. X = \sigma(Y)$



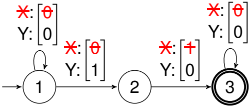
$\mathcal{A}_{X=\sigma(Y)}$ → Projection on X → Accepting 0*100*

How to handle quantification

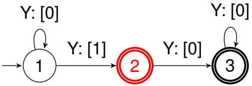
- issue with projection (existential quantification)
  - after removing of the tracks not all models would be accepted (problem with 0-padding)
    - needed for **soundness**!
    - for every assignment, it is necesssary to accept all or none encodings
  - so after projection we need to adjust the final states by saturation
    - pump the final states with all states backward reachable with 0
- consider $\exists X . X = \sigma(Y)$



$\mathcal{A}_{X=\sigma(Y)}$



→ Projection on X



→ Accepting 0*100*
→ make to accept 0*10*

Ground Formulae

We focus on validity of ground formulae (all variables are quantified)

- satisfiability/validity of other formulae: prefixing with $\exists/\forall$

Key observation for ground formulae

$$\models \varphi \quad \text{iff} \quad \varepsilon \in L(\varphi)$$

Ground Formulae

We focus on validity of ground formulae (all variables are quantified)

- satisfiability/validity of other formulae: prefixing with $\exists/\forall$

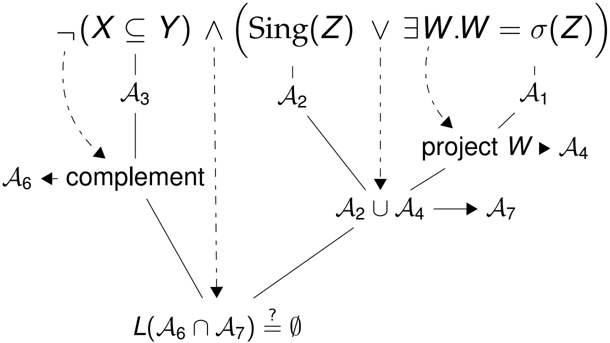
Key observation for ground formulae

$$\models \varphi \quad \text{iff} \quad \varepsilon \in L(\varphi)$$

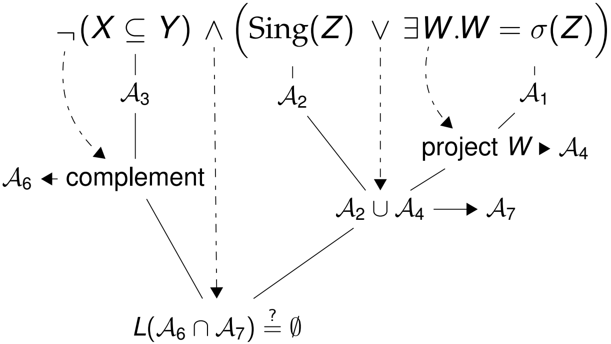
Why?

- Formula φ is valid if it accepts everything ($L(\varphi) = \Sigma^*$)
- Formula φ is unsatisfiable if it accepts nothing ($L(\varphi) = \emptyset$)
  - so it is sufficient to just test membership of ε

Issues with constructing automata



Issues with constructing automata



$$\neg(X \subseteq Y) \land \Big(\mathrm{Sing}(Z) \lor \exists W.W = \sigma(Z)\Big)$$

$\mathcal{A}_3$

$\mathcal{A}_2$

$\mathcal{A}_1$

project W ▸ $\mathcal{A}_4$

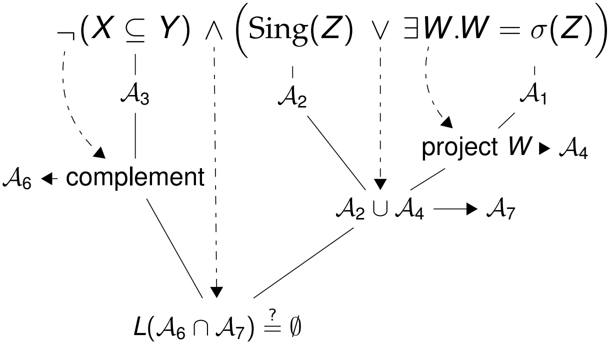
$\mathcal{A}_6$ ◂ complement

$\mathcal{A}_2 \cup \mathcal{A}_4 \longrightarrow \mathcal{A}_7$

$L(\mathcal{A}_6 \cap \mathcal{A}_7) \overset{?}{=} \emptyset$

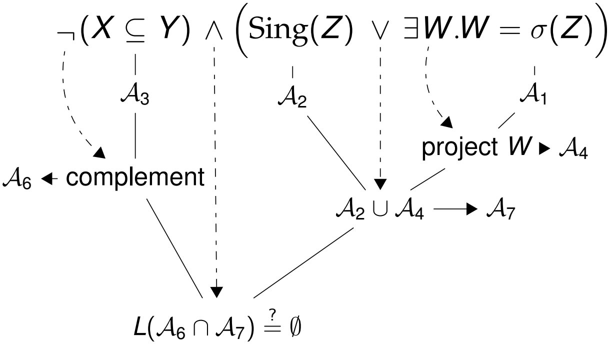
1. Constructing the whole automaton, checking $\varepsilon \in L(\mathcal{A})$ later!

Issues with constructing automata



1. Constructing the whole automaton, checking $\varepsilon \in L(\mathcal{A})$ later!
2. Quantifier alternations ($\forall\exists \leadsto \neg\exists\neg\exists$)
   $\leadsto$ exponential blow-up after subset construction.

Issues with constructing automata



$\neg(X \subseteq Y) \wedge \Big(\mathrm{Sing}(Z) \vee \exists W.W = \sigma(Z)\Big)$

$\mathcal{A}_3$

$\mathcal{A}_2$

$\mathcal{A}_1$

project W ▸ $\mathcal{A}_4$

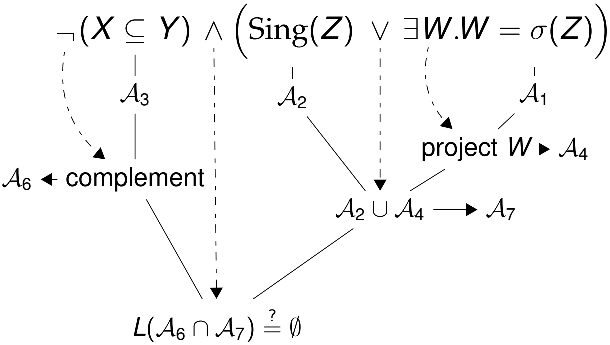
$\mathcal{A}_6$ ◂ complement

$\mathcal{A}_2 \cup \mathcal{A}_4 \longrightarrow \mathcal{A}_7$

$L(\mathcal{A}_6 \cap \mathcal{A}_7) \overset{?}{=} \emptyset$

1. Constructing the whole automaton, checking $\varepsilon \in L(\mathcal{A})$ later!
2. Quantifier alternations ($\forall\exists \rightsquigarrow \neg\exists\neg\exists$)
   $\rightsquigarrow$ exponential blow-up after subset construction.
3. For $\mathcal{A}_6 \cap \mathcal{A}_7$, what if $L(\mathcal{A}_6) = \emptyset$?
   - No need to construct $\mathcal{A}_7$ and $\mathcal{A}_6 \cap \mathcal{A}_7$!

Towards Language Terms



$$\neg(X \subseteq Y) \wedge \Big(\mathrm{Sing}(Z) \vee \exists W. W = \sigma(Z)\Big)$$

$\mathcal{A}_3$

$\mathcal{A}_2$

$\mathcal{A}_1$

project $W \blacktriangleright \mathcal{A}_4$

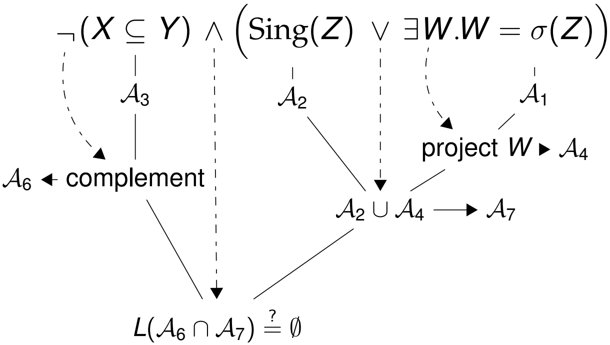
$\mathcal{A}_6 \blacktriangleleft$ complement

$\mathcal{A}_2 \cup \mathcal{A}_4 \longrightarrow \mathcal{A}_7$

$L(\mathcal{A}_6 \cap \mathcal{A}_7) \stackrel{?}{=} \emptyset$

- Instead, we:

Towards Language Terms



$$\neg(X \subseteq Y) \wedge \left(\mathrm{Sing}(Z) \vee \exists W.W = \sigma(Z)\right)$$

$\mathcal{A}_3$

$\mathcal{A}_2$

$\mathcal{A}_1$

project W ▸ $\mathcal{A}_4$

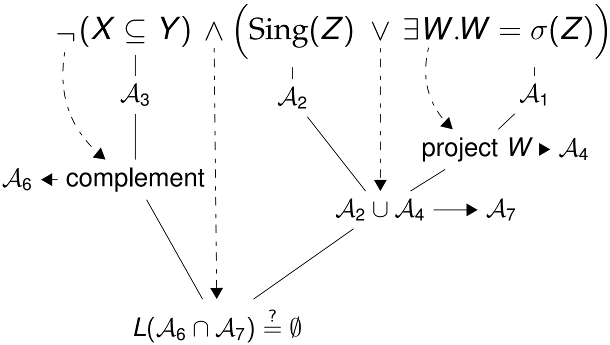
$\mathcal{A}_6$ ◂ complement

$\mathcal{A}_2 \cup \mathcal{A}_4 \longrightarrow \mathcal{A}_7$

$L(\mathcal{A}_6 \cap \mathcal{A}_7) \overset{?}{=} \emptyset$

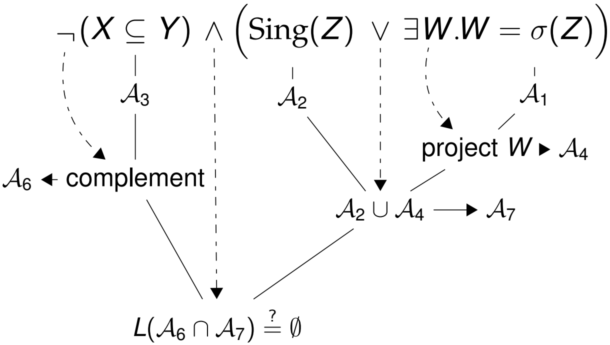
- Instead, we:
  - Represent (sub)formulae as so-called language terms

Towards Language Terms



- Instead, we:
  - Represent (sub)formulae as so-called language terms
  - Evaluate the $\varepsilon \in L(\mathcal{A})$ query lazily → on-the-fly

Towards Language Terms



$$\neg(X \subseteq Y) \land \Big(\mathrm{Sing}(Z) \lor \exists W.W = \sigma(Z)\Big)$$

$\mathcal{A}_3$

$\mathcal{A}_2$

$\mathcal{A}_1$

project W ▸ $\mathcal{A}_4$

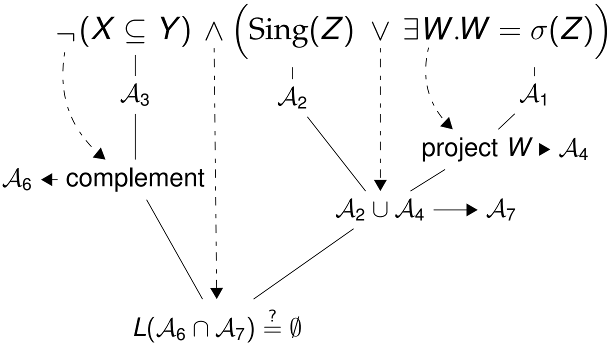
$\mathcal{A}_6$ ◂ complement

$\mathcal{A}_2 \cup \mathcal{A}_4 \longrightarrow \mathcal{A}_7$

$L(\mathcal{A}_6 \cap \mathcal{A}_7) \overset{?}{=} \emptyset$

- Instead, we:
  - Represent (sub)formulae as so-called language terms
  - Evaluate the $\varepsilon \in L(\mathcal{A})$ query lazily → on-the-fly
  - Compute the saturation fixpoints lazily

Towards Language Terms



$$\neg(X \subseteq Y) \land \Big(\mathrm{Sing}(Z) \lor \exists W.W = \sigma(Z)\Big)$$

$\mathcal{A}_3$

$\mathcal{A}_2$

$\mathcal{A}_1$

project W ▸ $\mathcal{A}_4$

$\mathcal{A}_6$ ◂ complement

$\mathcal{A}_2 \cup \mathcal{A}_4 \longrightarrow \mathcal{A}_7$

$$L(\mathcal{A}_6 \cap \mathcal{A}_7) \stackrel{?}{=} \emptyset$$

- Instead, we:
  - Represent (sub)formulae as so-called language terms
  - Evaluate the $\varepsilon \in L(\mathcal{A})$ query lazily → on-the-fly
  - Compute the saturation fixpoints lazily
  - Use subsumption to prune state space

Overview of our method

1. Reasoning over language terms
    - Structure of the terms $t_\varphi \sim$ structure of φ
        - but terms can be partially evaluated, unfolded, DAGified, etc.

Overview of our method

1. Reasoning over language terms
    - Structure of the terms $t_\varphi \sim$ structure of φ
        - but terms can be partially evaluated, unfolded, DAGified, etc.
    - Leaves of the terms correspond to states of Finite Automata

Overview of our method

1. Reasoning over language terms
    - Structure of the terms $t_\varphi \sim$ structure of φ
        - but terms can be partially evaluated, unfolded, DAGified, etc.
    - Leaves of the terms correspond to states of Finite Automata
    - Inner nodes:
        - $\varphi \wedge \psi \rightsquigarrow t_\varphi \cap t_\psi$
        - $\varphi \vee \psi \rightsquigarrow t_\varphi \cup t_\psi$

Overview of our method

1. Reasoning over language terms
    - Structure of the terms $t_\varphi \sim$ structure of φ
        - but terms can be partially evaluated, unfolded, DAGified, etc.
    - Leaves of the terms correspond to states of Finite Automata
    - Inner nodes:
        - $\varphi \land \psi \rightsquigarrow t_\varphi \cap t_\psi$
        - $\varphi \lor \psi \rightsquigarrow t_\varphi \cup t_\psi$
        - $\neg\varphi \rightsquigarrow \overline{t_\varphi}$

Overview of our method

1. Reasoning over language terms
    - Structure of the terms $t_\varphi \sim$ structure of φ
        - but terms can be partially evaluated, unfolded, DAGified, etc.
    - Leaves of the terms correspond to states of Finite Automata
    - Inner nodes:
        - $\varphi \wedge \psi \rightsquigarrow t_\varphi \cap t_\psi$
        - $\varphi \vee \psi \rightsquigarrow t_\varphi \cup t_\psi$
        - $\neg\varphi \rightsquigarrow \overline{t_\varphi}$
        - $\exists X.\varphi \rightsquigarrow \pi_x(t_\varphi) - \overline{0}^*$
            - π_X corresponds to the projection of the variable X in $L(\varphi)$
            - -0^* corresonds to the left quotient of $L(\varphi)$

Overview of our method

1. Reasoning over language terms
    - Structure of the terms $t_\varphi \sim$ structure of φ
        - but terms can be partially evaluated, unfolded, DAGified, etc.
    - Leaves of the terms correspond to states of Finite Automata
    - Inner nodes:
        - $\varphi \land \psi \rightsquigarrow t_\varphi \cap t_\psi$
        - $\varphi \lor \psi \rightsquigarrow t_\varphi \cup t_\psi$
        - $\neg\varphi \rightsquigarrow \overline{t_\varphi}$
        - $\exists X.\varphi \rightsquigarrow \pi_x(t_\varphi) - \overline{0}^*$
            - π_X corresponds to the projection of the variable X in $L(\varphi)$
            - -0^* corresonds to the left quotient of $L(\varphi)$

2. Validity checking of ground formula φ is reduced to the ε-membership test on t_φ

Overview of our method

1. Reasoning over language terms
   - Structure of the terms $t_\varphi \sim$ structure of φ
     - but terms can be partially evaluated, unfolded, DAGified, etc.
   - Leaves of the terms correspond to states of Finite Automata
   - Inner nodes:
     - $\varphi \land \psi \rightsquigarrow t_\varphi \cap t_\psi$
     - $\varphi \lor \psi \rightsquigarrow t_\varphi \cup t_\psi$
     - $\neg\varphi \rightsquigarrow \overline{t_\varphi}$
     - $\exists X.\varphi \rightsquigarrow \pi_x(t_\varphi) - \overline{0}^*$
       - π_X corresponds to the projection of the variable X in $L(\varphi)$
       - -0^* corresonds to the left quotient of $L(\varphi)$

2. Validity checking of ground formula φ is reduced to the ε-membership test on t_φ
   - Intuition: Automaton either accepts Σ^* or nothing, so ε test suffices
   - $\models \varphi \iff \varepsilon \in t_\varphi$

Overview of our method

3. Lazy evaluation of ε-membership on term t

Overview of our method

3. Lazy evaluation of ε-membership on term t
   - $\varepsilon \in \mathcal{A} \Leftrightarrow I_{\mathcal{A}} \cap F_{\mathcal{A}} \neq \emptyset$

Overview of our method

3. Lazy evaluation of ε-membership on term t
   - $\varepsilon \in \mathcal{A} \Leftrightarrow I_{\mathcal{A}} \cap F_{\mathcal{A}} \neq \emptyset$
   - $\varepsilon \in t_{\varphi} \cap t_{\psi} \Leftrightarrow \varepsilon \in t_{\varphi} \wedge \varepsilon \in t_{\psi}$
     - if $\varepsilon \notin t_{\varphi}$ no need to check if $\varepsilon \in t_{\psi}$

Overview of our method

3. Lazy evaluation of ε-membership on term t
   - $\varepsilon \in \mathcal{A} \Leftrightarrow I_{\mathcal{A}} \cap F_{\mathcal{A}} \neq \emptyset$
   - $\varepsilon \in t_{\varphi} \cap t_{\psi} \Leftrightarrow \varepsilon \in t_{\varphi} \wedge \varepsilon \in t_{\psi}$
     - if $\varepsilon \notin t_{\varphi}$ no need to check if $\varepsilon \in t_{\psi}$
   - $\varepsilon \in t_{\varphi} \cup t_{\psi} \Leftrightarrow \varepsilon \in t_{\varphi} \vee \varepsilon \in t_{\psi}$
     - if $\varepsilon \in t_{\varphi}$ no need to check if $\varepsilon \in t_{\psi}$

Overview of our method

3. Lazy evaluation of ε-membership on term t
   - $\varepsilon \in \mathcal{A} \Leftrightarrow I_{\mathcal{A}} \cap F_{\mathcal{A}} \neq \emptyset$
   - $\varepsilon \in t_{\varphi} \cap t_{\psi} \Leftrightarrow \varepsilon \in t_{\varphi} \land \varepsilon \in t_{\psi}$
     - if $\varepsilon \notin t_{\varphi}$ no need to check if $\varepsilon \in t_{\psi}$
   - $\varepsilon \in t_{\varphi} \cup t_{\psi} \Leftrightarrow \varepsilon \in t_{\varphi} \lor \varepsilon \in t_{\psi}$
     - if $\varepsilon \in t_{\varphi}$ no need to check if $\varepsilon \in t_{\psi}$
   - $\varepsilon \in \overline{t_{\varphi}} \Leftrightarrow \varepsilon \notin t_{\varphi}$

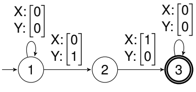
Overview of our method

3. Lazy evaluation of ε-membership on term t
   - $\varepsilon \in \mathcal{A} \Leftrightarrow I_{\mathcal{A}} \cap F_{\mathcal{A}} \neq \emptyset$
   - $\varepsilon \in t_{\varphi} \cap t_{\psi} \Leftrightarrow \varepsilon \in t_{\varphi} \wedge \varepsilon \in t_{\psi}$
     - if $\varepsilon \notin t_{\varphi}$ no need to check if $\varepsilon \in t_{\psi}$
   - $\varepsilon \in t_{\varphi} \cup t_{\psi} \Leftrightarrow \varepsilon \in t_{\varphi} \vee \varepsilon \in t_{\psi}$
     - if $\varepsilon \in t_{\varphi}$ no need to check if $\varepsilon \in t_{\psi}$
   - $\varepsilon \in \overline{t_{\varphi}} \Leftrightarrow \varepsilon \notin t_{\varphi}$
   - $\varepsilon \in \pi_X(t_{\varphi}) \Leftrightarrow \varepsilon \in t_{\varphi}$

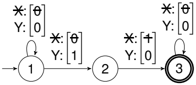
Overview of our method

3 Lazy evaluation of ε-membership on term t

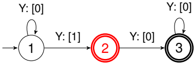
- $\varepsilon \in t - \overline{0}^* \Leftrightarrow \varepsilon \in t \vee \varepsilon \in t - \overline{0} \vee \varepsilon \in t - \overline{0}\,\overline{0} \vee \dots$
    - evaluation of the quotients leads to fixpoint computations
    - lazy evaluation $\rightsquigarrow$ iteratively test $\varepsilon \in t, \varepsilon \in t - \overline{0}, \dots$
    - …until fixpoint reached or satisfying member found



$\mathcal{A}_{X=\sigma(Y)}$



→ Projection on X



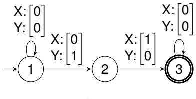
→ Accepting 0*100*
→ make to accept 0*10*

Overview of our method

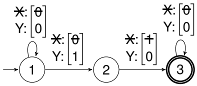
3 Lazy evaluation of ε-membership on term t

- $\varepsilon \in t - \overline{0}^* \Leftrightarrow \varepsilon \in t \vee \varepsilon \in t - \overline{0} \vee \varepsilon \in t - \overline{0}\,\overline{0} \vee \ldots$
  - evaluation of the quotients leads to fixpoint computations
  - lazy evaluation $\rightsquigarrow$ iteratively test $\varepsilon \in t, \varepsilon \in t - \overline{0}, \ldots$
  - …until fixpoint reached or satisfying member found

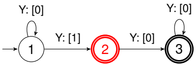
- $\varepsilon \in t - \overline{0}$
  - $-\overline{0}$ on inner nodes: push through to leaves
  - $-\overline{0}$ on leaves: compute 0-predecessors of final states



$\mathcal{A}_{X=\sigma(Y)}$



→ Projection on X

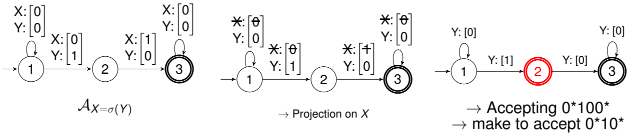


→ Accepting 0*100*
→ make to accept 0*10*

Overview of our method

3 Lazy evaluation of ε-membership on term t

- $\varepsilon \in t - \overline{0}^* \Leftrightarrow \varepsilon \in t \vee \varepsilon \in t - \overline{0} \vee \varepsilon \in t - \overline{0}\,\overline{0} \vee \dots$
  - evaluation of the quotients leads to fixpoint computations
  - lazy evaluation $\rightsquigarrow$ iteratively test $\varepsilon \in t, \varepsilon \in t - \overline{0}, \dots$
  - …until fixpoint reached or satisfying member found
- $\varepsilon \in t - \overline{0}$
  - $-\overline{0}$ on inner nodes: push through to leaves
  - $-\overline{0}$ on leaves: compute 0-predecessors of final states

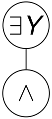


4 Further optimizations

- e.g. subsumption, continuations, formula preprocessing, etc.

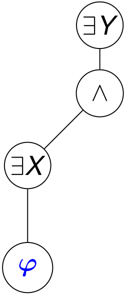
Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$

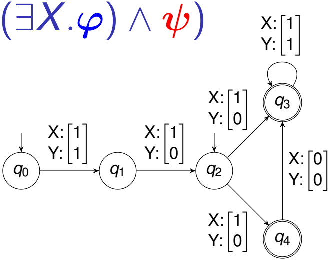
Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$



- We represent the formula symbolically as a language term $t_{\exists Y.(\exists X.\varphi)\land\psi}$ and test the emptiness.
- $\varepsilon \in t_{\exists Y.(\exists X.\varphi)\land\psi} \iff \varepsilon \in (t_{\exists X.\varphi} \cap t_{\psi}) - \bar{0}^*$
  $\iff \varepsilon \in t_{\exists X.\varphi} \cap t_{\psi} \quad \lor \quad \varepsilon \in (t_{\exists X.\varphi} \cap t_{\psi}) - \bar{0} \quad \lor \quad \varepsilon \in (t_{\exists X.\varphi} \cap t_{\psi}) - \bar{0}^2 \ldots$
- We will demonstrate our method just on testing if $\varepsilon \in t_{\exists X.\varphi} \cap t_{\psi}$
  - (some details will be omitted)

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$

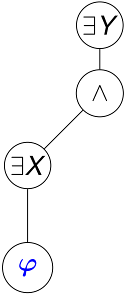


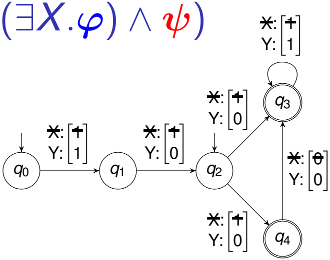


(a) Automaton for φ

- The term $t_{\exists X.\varphi}$ corresponds to the left subformula $\exists X.\varphi$

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$

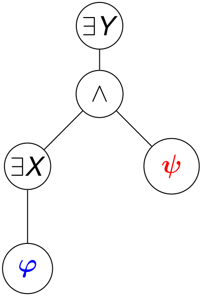


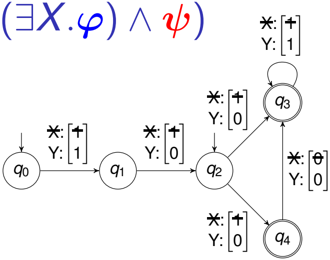


(a) Automaton for $\exists X.\varphi$

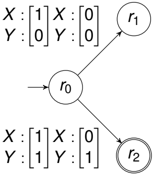
- The term $t_{\exists X.\varphi}$ corresponds to the left subformula $\exists X.\varphi$

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$





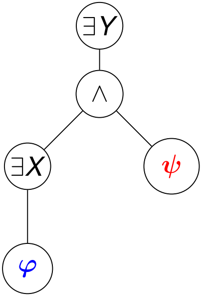
(a) Automaton for $\exists X.\varphi$

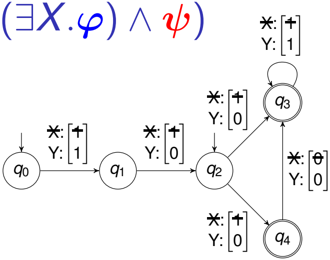


(b) Automaton for ψ

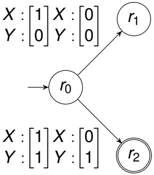
- The term $t_{\exists X.\varphi}$ corresponds to the left subformula $\exists X.\varphi$
- The term t_{ψ} corresponds to the right subformula ψ

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$





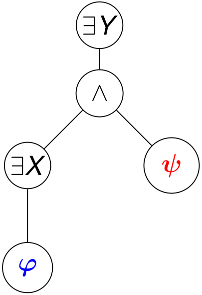
(a) Automaton for $\exists X.\varphi$

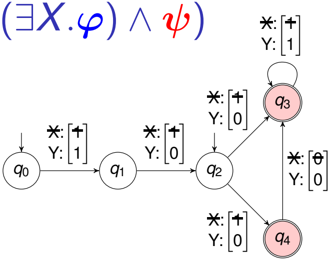


(b) Automaton for ψ

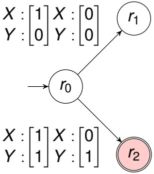
- We start the emptiness check from final states of leaf automata.
- (After projection new final states are backward reachable from current final states)

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$





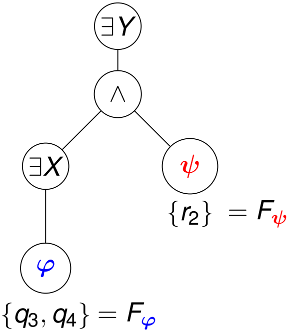
(a) Automaton for $\exists X.\varphi$

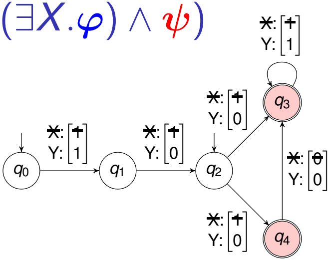


(b) Automaton for ψ

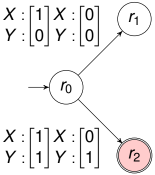
- We start the emptiness check from final states of leaf automata.
- (After projection new final states are backward reachable from current final states)

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$





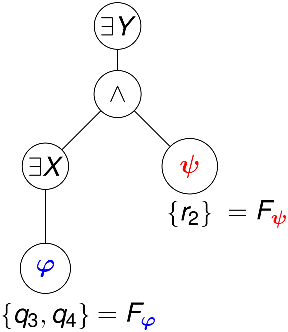
(a) Automaton for $\exists X.\varphi$

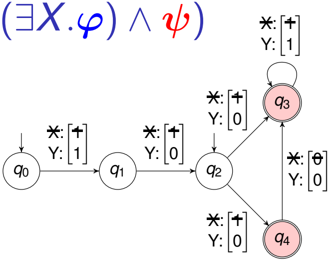


(b) Automaton for ψ

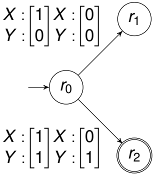
- We start the emptiness check from final states of leaf automata.
- (After projection new final states are backward reachable from current final states)

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$





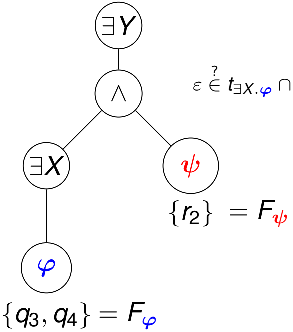
(a) Automaton for $\exists X.\varphi$



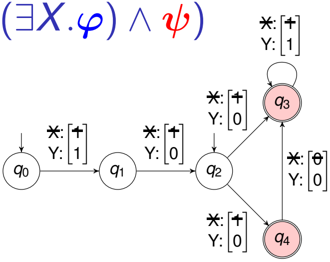
(b) Automaton for ψ

- $\varepsilon \in t_{\exists X.\varphi} \cap t_{\psi} \iff$

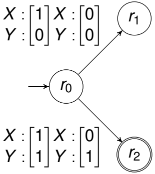
Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$



$\varepsilon \overset{?}{\in} t_{\exists X.\varphi} \cap t_{\psi}$



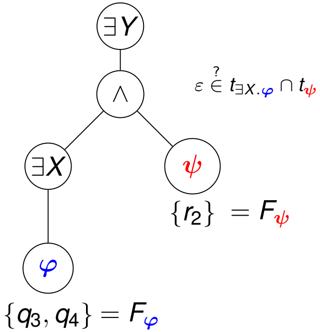
(a) Automaton for $\exists X.\varphi$

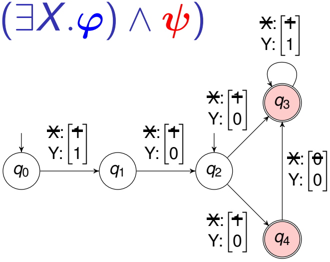


(b) Automaton for ψ

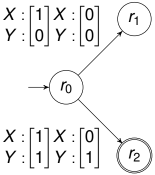
- $\varepsilon \in t_{\exists X.\varphi} \cap t_{\psi} \iff$

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$





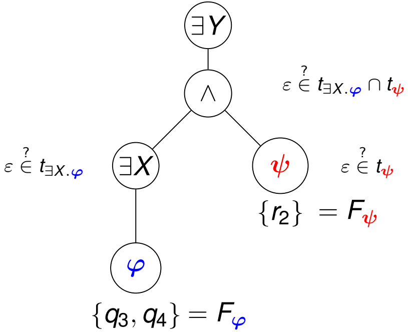
(a) Automaton for $\exists X.\varphi$

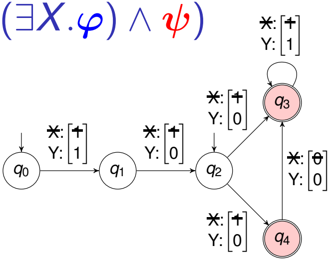


(b) Automaton for ψ

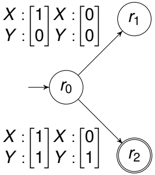
- $\varepsilon \in t_{\exists X.\varphi} \cap t_\psi \iff$
  $\iff \varepsilon \in t_{\exists X.\varphi} \land \varepsilon \in t_\psi$

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$





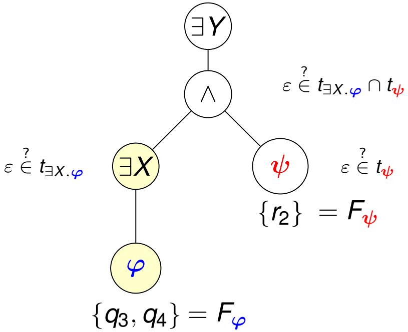
(a) Automaton for $\exists X.\varphi$

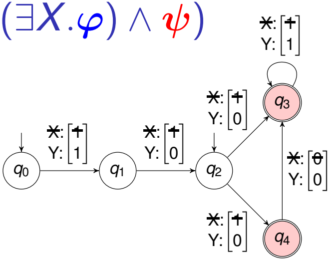


(b) Automaton for ψ

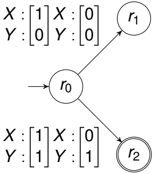
- $\varepsilon \in t_{\exists X.\varphi} \cap t_{\psi} \iff$
  $\iff \varepsilon \in t_{\exists X.\varphi} \land \varepsilon \in t_{\psi}$

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$





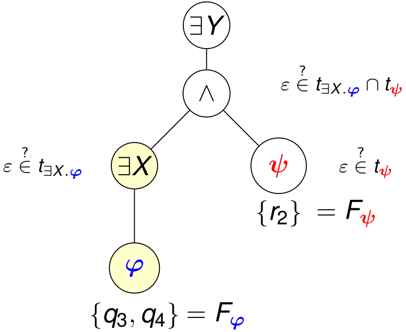
(a) Automaton for $\exists X.\varphi$

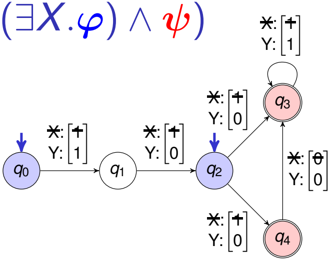


(b) Automaton for ψ

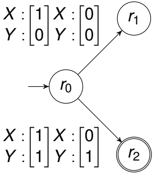
- $\varepsilon \in t_{\exists X.\varphi} \iff \varepsilon \in t_{\varphi} - \bar{0}^*$
  $\iff \varepsilon \in t_{\varphi} \lor \varepsilon \in t_{\varphi} - \bar{0} \lor \varepsilon \in t_{\varphi} - \bar{0}^2 \ldots$
- $\varepsilon \in t_{\varphi} \iff I_{\varphi} \cap F_{\varphi} \neq \emptyset$.

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$





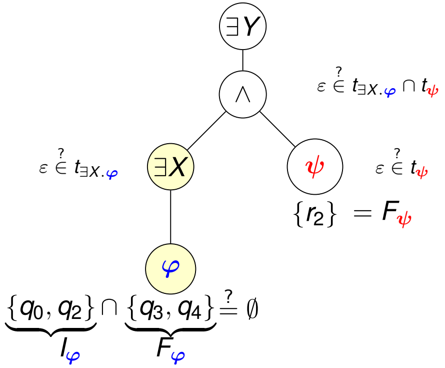
(a) Automaton for $\exists X.\varphi$

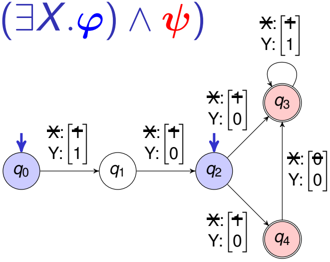


(b) Automaton for ψ

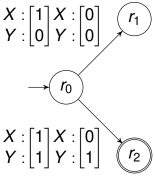
- $\varepsilon \in t_{\exists X.\varphi} \iff \varepsilon \in t_{\varphi} - \bar{0}^{*}$
  $\iff \varepsilon \in t_{\varphi} \lor \varepsilon \in t_{\varphi} - \bar{0} \lor \varepsilon \in t_{\varphi} - \bar{0}^{2} \ldots$
- $\varepsilon \in t_{\varphi} \iff I_{\varphi} \cap F_{\varphi} \neq \emptyset.$

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$





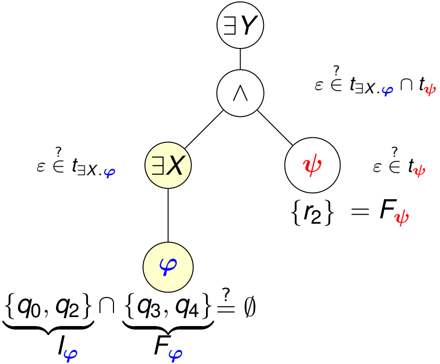
(a) Automaton for $\exists X.\varphi$

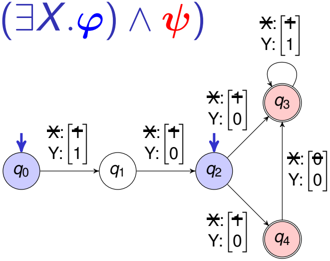


(b) Automaton for ψ

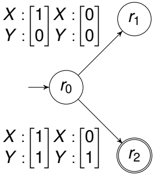
- $\varepsilon \in t_{\exists X.\varphi} \iff \varepsilon \in t_\varphi - \bar{0}^*$
  $\iff \varepsilon \in t_\varphi \lor \varepsilon \in t_\varphi - \bar{0} \lor \varepsilon \in t_\varphi - \bar{0}^2 \ldots$
- $\varepsilon \in t_\varphi \iff I_\varphi \cap F_\varphi \neq \emptyset.$

Validity checking of $\exists Y.((\exists X.\varphi) \wedge \psi)$





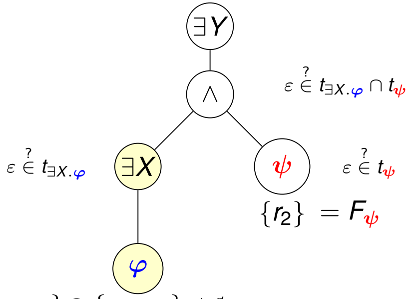
(a) Automaton for $\exists X.\varphi$



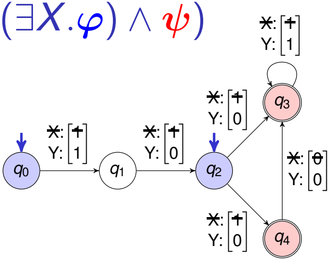
(b) Automaton for ψ

- $\{q_0, q_2\} \cap \{q_3, q_4\} = \emptyset, \ldots$
- …but we cannot conclude that $\varepsilon \notin t_{\exists X.\varphi}, \ldots$

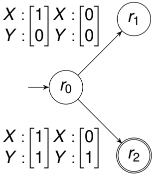
Validity checking of $\exists Y.((\exists X.\varphi) \wedge \psi)$



$$\underbrace{\{q_0, q_2\}}_{I_\varphi} \cap \underbrace{\{q_3, q_4\}}_{F_\varphi} \neq \emptyset$$



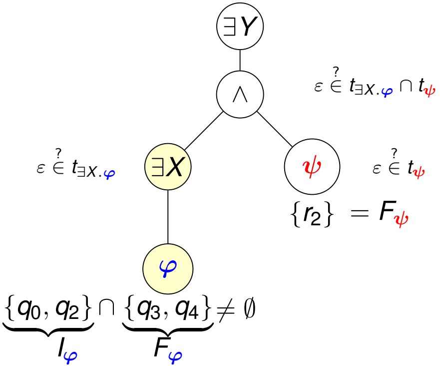
(a) Automaton for $\exists X.\varphi$

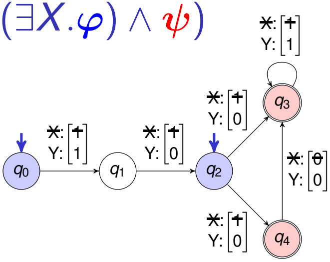


(b) Automaton for ψ

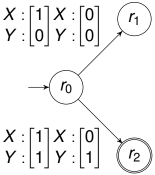
- $\{q_0, q_2\} \cap \{q_3, q_4\} = \emptyset$,...
- ...but we cannot conclude that $\varepsilon \notin t_{\exists X.\varphi}$, ...

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$





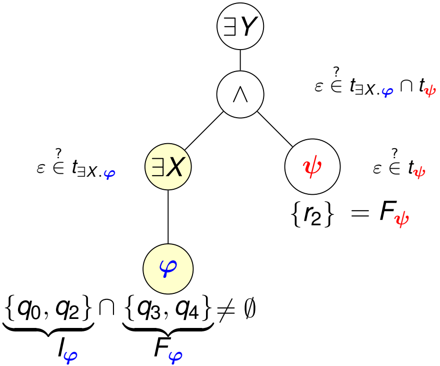
(a) Automaton for $\exists X.\varphi$

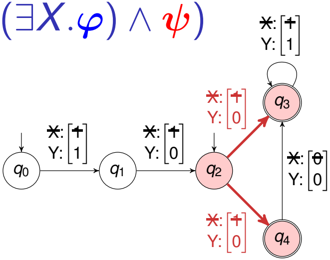


(b) Automaton for ψ

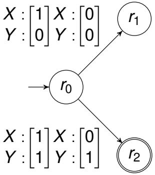
- We have to saturate the final states (because of projection)
- One step of saturation yields the set of states $F_\varphi - \overline{0}$.

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$





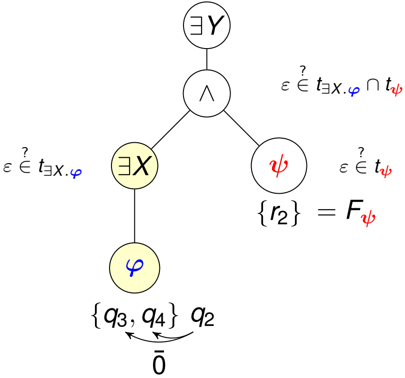
(a) Automaton for $\exists X.\varphi$

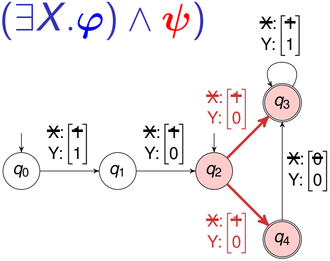


(b) Automaton for ψ

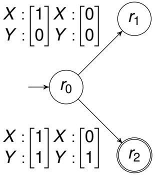
- We have to saturate the final states (because of projection)
- One step of saturation yields the set of states $F_\varphi - \overline{0}$.

Validity checking of $\exists Y.((\exists X.\varphi) \wedge \psi)$





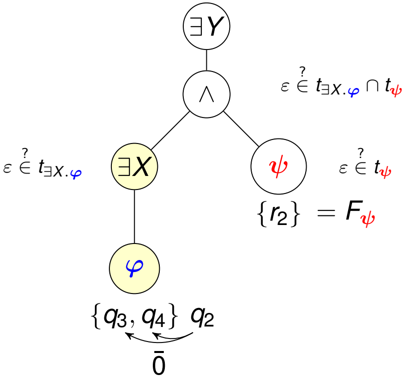
(a) Automaton for $\exists X.\varphi$

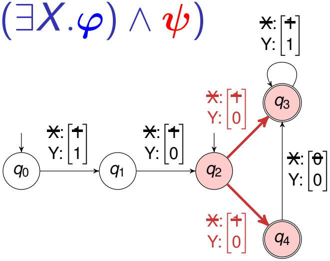


(b) Automaton for ψ

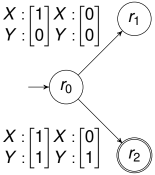
- We have to saturate the final states (because of projection)
- One step of saturation yields the set of states $F_{\varphi} - \bar{0}$.

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$





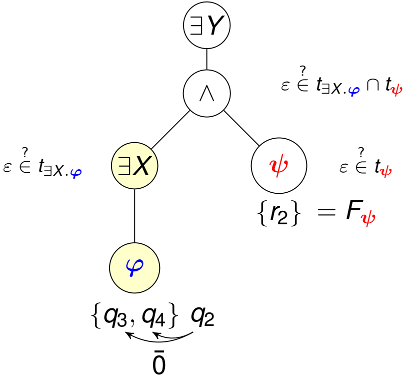
(a) Automaton for $\exists X.\varphi$

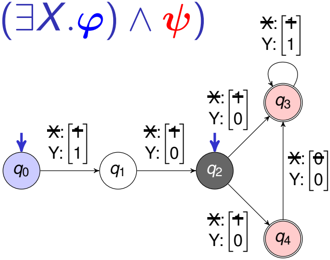


(b) Automaton for ψ

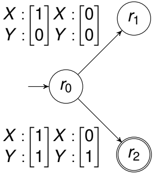
- We repeat the check: $\varepsilon \in t_\varphi - \overline{0} \iff$
  $\iff I_\varphi \cap F_\varphi - \overline{0} \neq \emptyset$

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$





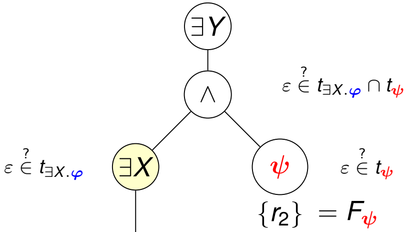
(a) Automaton for $\exists X.\varphi$



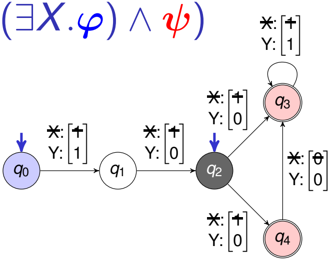
(b) Automaton for ψ

- We repeat the check: $\varepsilon \in t_{\varphi} - \overline{0} \iff$
  $\iff I_{\varphi} \cap F_{\varphi} - \overline{0} \neq \emptyset$

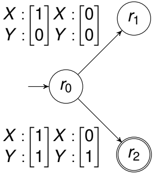
Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$



$$\underbrace{\{q_0, q_2\}}_{I_\varphi} \cap \underbrace{\{q_3, q_4, q_2\}}_{F_\varphi - \overline{0}} \overset{?}{=} \emptyset$$



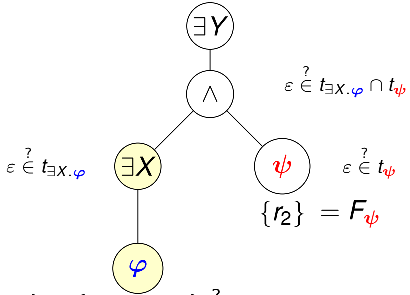
(a) Automaton for $\exists X.\varphi$



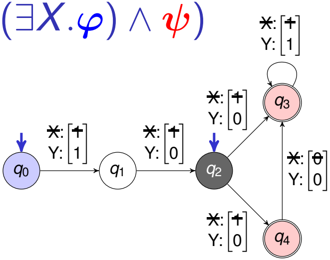
(b) Automaton for ψ

- We repeat the check: $\varepsilon \in t_\varphi - \overline{0} \iff$
  $\iff I_\varphi \cap F_\varphi - \overline{0} \neq \emptyset$

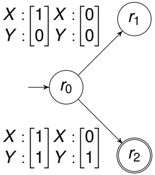
Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$



$$\underbrace{\{q_0, q_2\}}_{I_\varphi} \cap \underbrace{\{q_3, q_4, q_2\}}_{F_\varphi - \bar{0}} \overset{?}{=} \emptyset$$



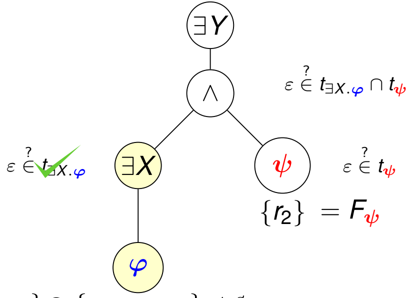
(a) Automaton for $\exists X.\varphi$



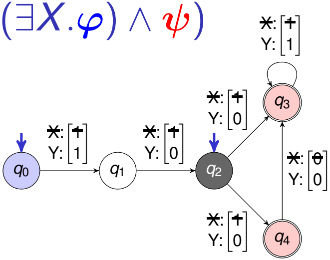
(b) Automaton for ψ

- Since $\{q_0, q_2\} \cap \{q_3, q_4, q_2\} \neq \emptyset$, ...
- ... we conclude that $\varepsilon \in t_{\varphi} - \bar{0}$ and hence $\varepsilon \in t_{\exists X.\varphi}$.

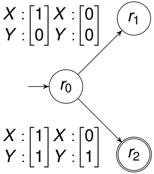
Validity checking of $\exists Y.((\exists X.\varphi) \wedge \psi)$



$$\underbrace{\{q_0, q_2\}}_{I_{\varphi}} \cap \underbrace{\{q_3, q_4, q_2\}}_{F_{\varphi} - \bar{0}} \neq \emptyset$$



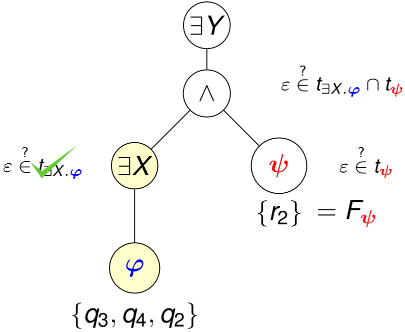
(a) Automaton for $\exists X.\varphi$

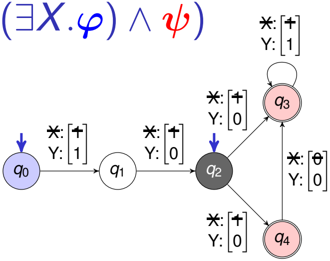


(b) Automaton for ψ

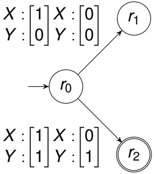
- Since $\{q_0, q_2\} \cap \{q_3, q_4, q_2\} \neq \emptyset$, . . .
- . . . we conclude that $\varepsilon \in t_{\varphi} - \bar{0}$ and hence $\varepsilon \in t_{\exists X.\varphi}$.

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$





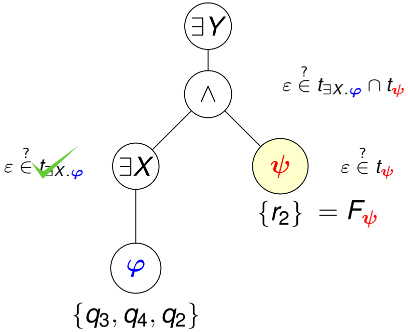
(a) Automaton for $\exists X.\varphi$

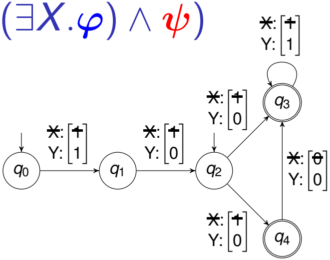


(b) Automaton for ψ

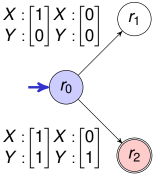
- However, we cannot short-circuit the test.
- So we have to compute $\varepsilon \in t_{\psi}$

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$





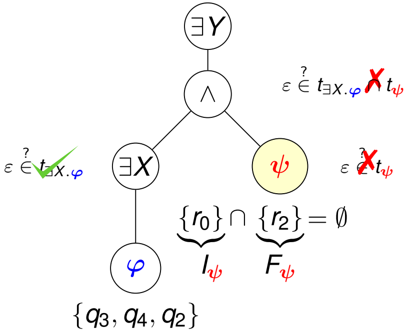
(a) Automaton for $\exists X.\varphi$

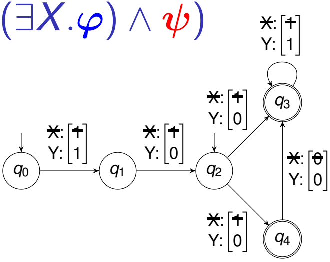


(b) Automaton for ψ

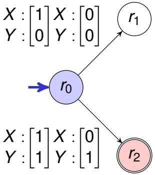
- However, we cannot short-circuit the test.
- So we have to compute $\varepsilon \in t_{\psi}$

Validity checking of $\exists Y.((\exists X.\varphi) \wedge \psi)$





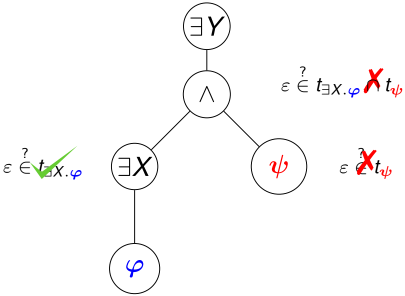
(a) Automaton for $\exists X.\varphi$

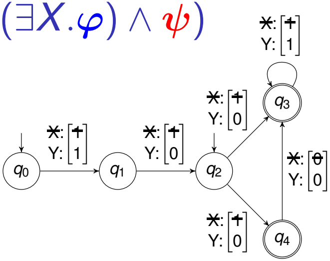


(b) Automaton for ψ

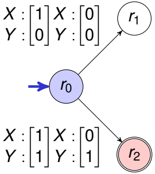
- However, we cannot short-circuit the test.
- So we have to compute $\varepsilon \in t_{\psi}$

Validity checking of $\exists Y.((\exists X.\varphi) \land \psi)$





(a) Automaton for $\exists X.\varphi$



(b) Automaton for ψ

- Until we find satisfying member or all of the fixpoints are computed...

Essential Optimizations

- lazy evaluation
  - if one branch of a binary operator suffices: short-circuit!

Essential Optimizations

- lazy evaluation
  - if one branch of a binary operator suffices: short-circuit!
  - if we find a satisfying guy in a fixpoint computation: short-circuit!

Essential Optimizations

- lazy evaluation
  - if one branch of a binary operator suffices: short-circuit!
  - if we find a satisfying guy in a fixpoint computation: short-circuit!
  - but with a caveat!

Essential Optimizations

- lazy evaluation
  - if one branch of a binary operator suffices: short-circuit!
  - if we find a satisfying guy in a fixpoint computation: short-circuit!
  - but with a caveat!
  - the algorithm has 2 interleaved phases:
    1. testing ε-membership
    2. computing left quotients

Essential Optimizations

- lazy evaluation
  - if one branch of a binary operator suffices: short-circuit!
  - if we find a satisfying guy in a fixpoint computation: short-circuit!
  - but with a caveat!
  - the algorithm has 2 interleaved phases:
    1. testing ε-membership
    2. computing left quotients
  - when computing quotients, we may need the result of a previously short-circuited operation
    - one needs to continue unfolding the fixpoint → continuations

Essential Optimizations

- lazy evaluation
  - if one branch of a binary operator suffices: short-circuit!
  - if we find a satisfying guy in a fixpoint computation: short-circuit!
  - but with a caveat!
  - the algorithm has 2 interleaved phases:
    1. testing ε-membership
    2. computing left quotients
  - when computing quotients, we may need the result of a previously short-circuited operation
    - one needs to continue unfolding the fixpoint → continuations

- combination with the explicit automata procedure (MONA)
  - we can prepare a minimal automaton for a subformula
  - reduces the underlying state space
  - various heuristics
    - we explicitly construct quantifier-free subformulae

Essential Optimizations

- Subsumption
  - when computing fixpoints, some elements can subsume other
  - keep fixpoint states minimal (cf. antichains)
  - subsumption even on partially computed elements

Essential Optimizations

- Subsumption
    - when computing fixpoints, some elements can subsume other
    - keep fixpoint states minimal (cf. antichains)
    - subsumption even on partially computed elements
- Formula pre-processing
    - pre-processing of the formula can greatly affect performance
    - anti-prenexing — pushing quantifiers down can reduce the explored state space (even exponentially!)

Experimental Evaluation of our tool Gaston

- Results on formulae generated by the UABE tool
  - Array Theory of Bounded Elements [ZhouHWGS'14]
  - formulae encode various array invariants
- ∞ represents that the tool timeouted in 2 minutes

Benchmark	Mona		Gaston	
	Time [s]	**Space**	**Time [s]**	**Space**
`a-a`	**1.51**	30 253	∞	∞
`ex10`	**6.92**	131 835	11.82	82 236
`ex11`	4.04	2 393	**0.10**	4 156
`ex12`	**0.11**	2 591	5.40	68 159
`ex13`	**0.01**	2 601	0.87	16 883
`ex16`	**0.01**	3 384	0.18	3 960
`ex17`	3.15	165 173	**0.09**	3 952
`ex18`	**0.18**	19 463	∞	∞
`ex2`	0.10	26 565	**0.01**	1 841
`ex20`	1.26	1 077	**0.21**	12 266
`ex21`	**1.51**	30 253	∞	∞
`ex4`	**0.03**	6 797	0.33	22 442
`ex6`	**3.69**	27 903	21.44	132 848
`ex7`	0.75	857	**0.01**	594
`ex8`	6.83	106 555	**0.01**	1 624
`ex9`	6.37	586 447	8.31	412 417
`fib`	**0.04**	8 128	22.15	126 688

Experimental Evaluation of our tool GASTON

- Results on set of parametrized benchmarks up to $k = 20$
- `oom`(k) represents that the tool run out of memory on formula k
- ∞(k) represents that the tool timeouted in 2 minutes on formula k

Benchmark	MONA	DWINA	TOSS	COALG	SFA	GASTON
`HornLeq`	`oom`(18)	**0.03**	0.08	∞(08)	**0.03**	**0.01**
`HornLeq` (+3)	`oom`(18)	∞(11)	0.16	∞(07)	∞(11)	**0.01**
`HornLeq` (+4)	`oom`(18)	∞(13)	**0.04**	∞(06)	∞(11)	**0.01**
`HornIn`	`oom`(15)	∞(11)	0.07	∞(08)	∞(08)	**0.01**
`HornTrans`	86.43	∞(14)	`N/A`	`N/A`	38.56	**1.06**
`SetClosed`	`oom`(05)	∞(14)	∞(03)	∞(01)	∞(04)	∞(06)
`SetSingle`	`oom`(04)	∞(08)	0.10	`N/A`	∞(03)	**0.01**
`Ex8`	`oom`(08)	`N/A`	`N/A`	`N/A`	`N/A`	**0.15**
`Ex11(10)`	`oom`(14)	`N/A`	`N/A`	`N/A`	`N/A`	**1.62**

- DWINA: Fiedor et al.: Nested antichains for WS1S
- TOSS: Ganzow and Kaizer: New algorithm for weak monadic second-order login on inductive structures
- COALG: Traytel: A coalgebraic decision procedure for WS1S
- SFA: D'Antoni and Veanes: Minimization of symbolic automata

Future Work

- extension to WS*k*S
  - weak monadic second-order logic of k successors
  - opens whole new world of tree structures

Future Work

- extension to WS*k*S
  - weak monadic second-order logic of k successors
  - opens whole new world of tree structures
- extension to infinite words/trees

Future Work

- extension to WS*k*S
    - weak monadic second-order logic of k successors
    - opens whole new world of tree structures
- extension to infinite words/trees
- application of the ideas in other automata-handling algorithms