Deciding WS1S Using an Automata-based Approach

Tomáš Fiedor[1,2] Lukáš Holík[2]

[1]Red Hat, Czech Republic

Ondřej Lengál[2] **Tomáš Vojnar**[2]

[2]Brno University of Technology, Czech Republic

Vienna UT, 2015

WS1S

- Weak monadic second-order logic of one successor:
  - second-order ⇒ quantification over relations;
  - monadic ⇒ the relations are unary – i.e. sets;
  - weak ⇒ the 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 ⇒ the relations are unary – i.e. sets;
  - weak ⇒ the sets are finite;
  - of one successor ⇒ reasoning about linear structures.
- Extensions of WS1S:
  - WS*k*S – with *k* successors;
  - S*k*S – allows quantification over infinite sets;
  - M2L(str) – allows quantification over infinite (but bounded) sets.

WS1S

- Weak monadic second-order logic of one successor:
  - second-order ⇒ quantification over relations;
  - monadic ⇒ the relations are unary – i.e. sets;
  - weak ⇒ the sets are finite;
  - of one successor ⇒ reasoning about linear structures.
- Extensions of WS1S:
  - WS*k*S – with *k* successors;
  - S*k*S – allows quantification over infinite sets;
  - M2L(str) – allows quantification over infinite (but bounded) sets.
- Corresponds to finite automata [Büchi'60].

WS1S

- Weak monadic second-order logic of one successor:
  - second-order ⇒ quantification over relations;
  - monadic ⇒ the relations are unary – i.e. sets;
  - weak ⇒ the sets are finite;
  - of one successor ⇒ reasoning about linear structures.
- Extensions of WS1S:
  - WS*k*S – with *k* successors;
  - S*k*S – allows quantification over infinite sets;
  - M2L(str) – allows quantification over infinite (but bounded) sets.
- Corresponds to finite automata [Büchi'60].
- Decidable, but **NONELEMENTARY**:
  - tower of exponentials of height given by quantifier alternations.
    - Deciding WS1S via DFAs: determinization, complementation, ...

The MONA Tool

- MONA – an automata-based WS1S/WSkS decision procedure:
  - semi-symbolic DFAs/DTAs: MTBDDs used to encode transitions,
  - efficient on many formulae obtained in various applications.
- Used in tools for checking complex shape invariants:
  - Pointer Assertion Logic Engine (`PALE`),
  - STRucture ANd Data (`STRAND`).
- Various other applications:
  - other kinds of program and protocol verification, parsing, synthesis, linguistics, multimedia, . . .

The MONA Tool

- MONA – an automata-based WS1S/WSkS decision procedure:
  - semi-symbolic DFAs/DTAs: MTBDDs used to encode transitions,
  - efficient on many formulae obtained in various applications.
- Used in tools for checking complex shape invariants:
  - Pointer Assertion Logic Engine (`PALE`),
  - STRucture ANd Data (`STRAND`).
- Various other applications:
  - other kinds of program and protocol verification, parsing, synthesis, linguistics, multimedia, . . .
- However, sometimes the complexity strikes back:
  - unavoidable in general,
  - one can try to push the usability border further:
    - using the recent advancements in non-deterministic automata.

Syntax and Semantics of WS1S

- Minimal syntax:
  - Let $X, Y, ...$ be 2nd-order variables.
  - Terms: ψ ::= $X \subseteq Y \mid \mathrm{Sing}(X) \mid X = \{0\} \mid X = \sigma(Y)$
  - Formulae: φ ::= $\psi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists X.\varphi$

Syntax and Semantics of WS1S

- Minimal syntax:
  - Let $X, Y, \ldots$ be 2nd-order variables.
  - Terms: ψ ::= $X \subseteq Y \mid \mathrm{Sing}(X) \mid X = \{0\} \mid X = \sigma(Y)$
  - Formulae: φ ::= $\psi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists X.\varphi$

- A note on semantics:
  - Variables interpreted as finite subsets of $\mathbb{N}$.
  - Singleton – $\mathrm{Sing}(X)$: $X = \{n\}$ for some $n \in \mathbb{N}$.
  - Successor – $X = \sigma(Y)$: $Y = \{n\}$ and $X = \{n+1\}$ for some $n \in \mathbb{N}$.

WS1S Syntactical Sugar

$$\varphi \Rightarrow \psi \quad \overset{\textit{def}}{\Leftrightarrow} \quad \neg\varphi \vee \psi$$

WS1S Syntactical Sugar

$$\varphi \Rightarrow \psi \overset{def}{\Leftrightarrow} \neg\varphi \lor \psi$$

$$\forall X.\varphi \overset{def}{\Leftrightarrow} \neg\exists X.\neg\varphi$$

WS1S Syntactical Sugar

$$\varphi \Rightarrow \psi \quad \stackrel{def}{\Leftrightarrow} \quad \neg\varphi \vee \psi$$

$$\forall X.\varphi \quad \stackrel{def}{\Leftrightarrow} \quad \neg\exists X.\neg\varphi$$

$$\exists x.\varphi \quad \stackrel{def}{\Leftrightarrow} \quad \exists X.\mathrm{Sing}(X) \wedge \varphi$$

WS1S Syntactical Sugar

$$\varphi \Rightarrow \psi \overset{def}{\Leftrightarrow} \neg\varphi \vee \psi$$

$$\forall X.\varphi \overset{def}{\Leftrightarrow} \neg\exists X.\neg\varphi$$

$$\exists x.\varphi \overset{def}{\Leftrightarrow} \exists X.\mathrm{Sing}(X) \wedge \varphi$$

$$\forall x.\varphi \overset{def}{\Leftrightarrow} \forall X.\mathrm{Sing}(X) \Rightarrow \varphi$$

WS1S Syntactical Sugar

$$\varphi \Rightarrow \psi \quad \overset{def}{\Leftrightarrow} \quad \neg\varphi \vee \psi$$

$$\forall X.\varphi \quad \overset{def}{\Leftrightarrow} \quad \neg\exists X.\neg\varphi$$

$$\exists x.\varphi \quad \overset{def}{\Leftrightarrow} \quad \exists X.\mathrm{Sing}(X) \wedge \varphi$$

$$\forall x.\varphi \quad \overset{def}{\Leftrightarrow} \quad \forall X.\mathrm{Sing}(X) \Rightarrow \varphi$$

$$X = Y \quad \overset{def}{\Leftrightarrow} \quad X \subseteq Y \wedge Y \subseteq X$$

WS1S Syntactical Sugar

$$\begin{aligned}
\varphi \Rightarrow \psi &\overset{\mathit{def}}{\Leftrightarrow} \neg\varphi \vee \psi \\
\forall X.\varphi &\overset{\mathit{def}}{\Leftrightarrow} \neg\exists X.\neg\varphi \\
\exists x.\varphi &\overset{\mathit{def}}{\Leftrightarrow} \exists X.\mathrm{Sing}(X) \wedge \varphi \\
\forall x.\varphi &\overset{\mathit{def}}{\Leftrightarrow} \forall X.\mathrm{Sing}(X) \Rightarrow \varphi \\
X = Y &\overset{\mathit{def}}{\Leftrightarrow} X \subseteq Y \wedge Y \subseteq X \\
X = \emptyset &\overset{\mathit{def}}{\Leftrightarrow} \forall Z.X \subseteq Z
\end{aligned}$$

WS1S Syntactical Sugar

$$
\begin{aligned}
\varphi \Rightarrow \psi &\overset{def}{\Leftrightarrow} \neg\varphi \vee \psi \\
\forall X.\varphi &\overset{def}{\Leftrightarrow} \neg\exists X.\neg\varphi \\
\exists x.\varphi &\overset{def}{\Leftrightarrow} \exists X.\mathrm{Sing}(X) \wedge \varphi \\
\forall x.\varphi &\overset{def}{\Leftrightarrow} \forall X.\mathrm{Sing}(X) \Rightarrow \varphi \\
X = Y &\overset{def}{\Leftrightarrow} X \subseteq Y \wedge Y \subseteq X \\
X = \emptyset &\overset{def}{\Leftrightarrow} \forall Z.X \subseteq Z \\
x \in Y &\overset{def}{\Leftrightarrow} \mathrm{Sing}(X) \wedge X \subseteq Y
\end{aligned}
$$

WS1S Syntactical Sugar

$$
\begin{aligned}
\varphi \Rightarrow \psi &\overset{def}{\Leftrightarrow} \neg\varphi \vee \psi \\
\forall X.\varphi &\overset{def}{\Leftrightarrow} \neg\exists X.\neg\varphi \\
\exists x.\varphi &\overset{def}{\Leftrightarrow} \exists X.\mathrm{Sing}(X) \wedge \varphi \\
\forall x.\varphi &\overset{def}{\Leftrightarrow} \forall X.\mathrm{Sing}(X) \Rightarrow \varphi \\
X = Y &\overset{def}{\Leftrightarrow} X \subseteq Y \wedge Y \subseteq X \\
X = \emptyset &\overset{def}{\Leftrightarrow} \forall Z.X \subseteq Z \\
x \in Y &\overset{def}{\Leftrightarrow} \mathrm{Sing}(X) \wedge X \subseteq Y \\
x \leq y &\overset{def}{\Leftrightarrow} \forall X.\,(y \in X \wedge (\forall z.\sigma(z) \in X \Rightarrow z \in X)) \Rightarrow x \in X
\end{aligned}
$$

Models of WS1S Formulae as Words

- Models of WS1S formulae:
  - maps of the variables to finite subsets of $\mathbb{N}$.

Models of WS1S Formulae as Words

- Models of WS1S formulae:
  - maps of the variables to finite subsets of $\mathbb{N}$.

- Such sets can be encoded as binary strings:
  - $\{1, 4, 5\} \rightarrow$

Index:	012345		0123456		01234567	
Membership:	x✓xx✓✓	,	x✓xx✓✓x	or	x✓xx✓✓xx	...
Encoding:	010011		0100110		01001100	

Models of WS1S Formulae as Words

- Models of WS1S formulae:
  - maps of the variables to finite subsets of $\mathbb{N}$.
- Such sets can be encoded as binary strings:
  - $\{1, 4, 5\} \to$

Index:	012345	0123456		01234567	
Membership:	x✓xx✓✓ ,	x✓xx✓✓x	or	x✓xx✓✓xx	…
Encoding:	010011	0100110		01001100	

  - Note that any number of zeros can always be added to the right!

Models of WS1S Formulae as Words

- Models of WS1S formulae:
  - maps of the variables to finite subsets of $\mathbb{N}$.
- Such sets can be encoded as binary strings:
  - $\{1, 4, 5\} \rightarrow$

Index:	012345	0123456		01234567
Membership:	x✓xx✓✓ ,	x✓xx✓✓x	or	x✓xx✓✓xx …
Encoding:	010011	0100110		01001100

  - Note that any number of zeros can always be added to the right!
- For each variable, we have one track in the alphabet:
  - e.g., $\begin{bmatrix}0\\0\end{bmatrix}$ is a symbol.

Models of WS1S Formulae as Words

- Models of WS1S formulae:
  - maps of the variables to finite subsets of $\mathbb{N}$.

- Such sets can be encoded as binary strings:
  - $\{1, 4, 5\} \to$

Index:	012345		0123456		01234567
Membership:	x✓xx✓✓	,	x✓xx✓✓x	or	x✓xx✓✓xx ...
Encoding:	010011		0100110		01001100

  - Note that any number of zeros can always be added to the right!

- For each variable, we have one track in the alphabet:
  - e.g., $\begin{bmatrix}0\\0\end{bmatrix}$ is a symbol.
  - A natural encoding for sets of transitions over such symbols:
    - multi-terminal BDDs.

Models of WS1S Formulae as Words

- Models of WS1S formulae:
  - maps of the variables to finite subsets of $\mathbb{N}$.

- Such sets can be encoded as binary strings:
  - $\{1, 4, 5\} \rightarrow$

Index:	012345	0123456		01234567
Membership:	x✓xx✓✓ ,	x✓xx✓✓x	or	x✓xx✓✓xx …
Encoding:	010011	0100110		01001100

  - Note that any number of zeros can always be added to the right!

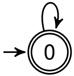
- For each variable, we have one track in the alphabet:
  - e.g., $\begin{bmatrix}0\\0\end{bmatrix}$ is a symbol.
  - A natural encoding for sets of transitions over such symbols:
    - multi-terminal BDDs.

- Example: $\{X_1 \mapsto \emptyset, X_2 \mapsto \{4, 2\}\} \rightsquigarrow \begin{matrix}X_1:\\X_2:\end{matrix}\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}.$

Atomic Automata for WS1S

$X \subseteq Y$ (*X* is a subset of *Y*)

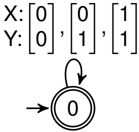
$$\begin{matrix}X:\\Y:\end{matrix}\begin{bmatrix}0\\0\end{bmatrix},\begin{bmatrix}0\\1\end{bmatrix},\begin{bmatrix}1\\1\end{bmatrix}$$



- $\begin{matrix}X \mapsto \{\ \ 2,\ \ 4\}\\ Y \mapsto \{1,2,3,4\}\end{matrix} \models X \subseteq Y \quad \rightsquigarrow \quad \begin{matrix}X:\\Y:\end{matrix}\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix}^* \subseteq L(\mathcal{A}_{X\subseteq Y})$

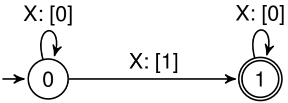
Atomic Automata for WS1S

$X \subseteq Y$ (X is a subset of Y)



- $\begin{matrix} X \mapsto \{\ \ 2,\ \ 4\} \\ Y \mapsto \{1,2,3,4\} \end{matrix} \models X \subseteq Y \quad \rightsquigarrow \quad \begin{matrix} X: \\ Y: \end{matrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix}^* \subseteq L(\mathcal{A}_{X \subseteq Y})$

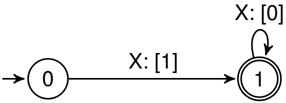
$\mathrm{Sing}(X)$ (X is singleton)



- $X \mapsto \{2\} \models Sing(X) \quad \rightsquigarrow \quad X : [0][0][1][0]^* \subseteq L(\mathcal{A}_{\mathrm{Sing}(X)})$

Atomic Automata for WS1S

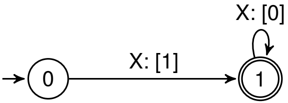
$X = \{0\}$ (X is the set $\{0\}$)



- $X \mapsto \{0\} \models X = \{0\} \quad \rightsquigarrow \quad X : [1][0]^* \subseteq L(\mathcal{A}_{X=\{0\}})$

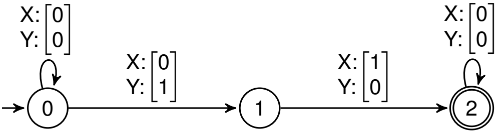
Atomic Automata for WS1S

$X = \{0\}$ (X is the set $\{0\}$)



- $X \mapsto \{0\} \models X = \{0\} \quad \leadsto \quad X : [1][0]^* \subseteq L(\mathcal{A}_{X=\{0\}})$

$X = \sigma(Y)$ (X is the successor of Y)



- $\begin{matrix} X \mapsto \{2\} \\ Y \mapsto \{1\} \end{matrix} \models X = \sigma(Y) \quad \leadsto \quad \begin{matrix} X: \\ Y: \end{matrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix}^* \subseteq L(\mathcal{A}_{X=\sigma(Y)})$

Construction of More Complex Automata

- Logical connectives mapped to automata operations.
  - More complex formulae handled by composition of automata.

Construction of More Complex Automata

- Logical connectives mapped to automata operations.
  - More complex formulae handled by composition of automata.

- Example:

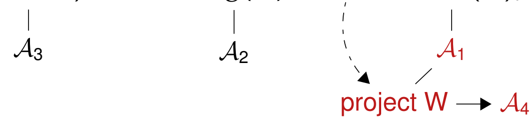
$$\neg \underbrace{(X \subseteq Y)}_{\mathcal{A}_3} \wedge \exists Z.(\underbrace{\mathrm{Sing}(Z)}_{\mathcal{A}_2} \vee \exists W. \underbrace{W = \sigma(Z)}_{\mathcal{A}_1})$$

Construction of More Complex Automata

- Logical connectives mapped to automata operations.
  - More complex formulae handled by composition of automata.

- Example:

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

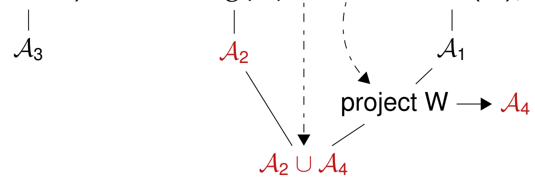


Construction of More Complex Automata

- Logical connectives mapped to automata operations.
  - More complex formulae handled by composition of automata.

- Example:

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

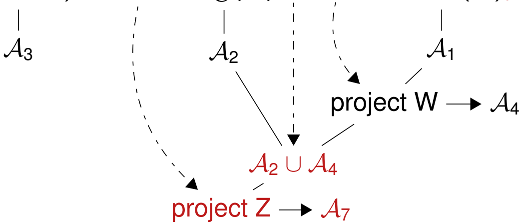


Construction of More Complex Automata

- Logical connectives mapped to automata operations.
  - More complex formulae handled by composition of automata.

- Example:

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

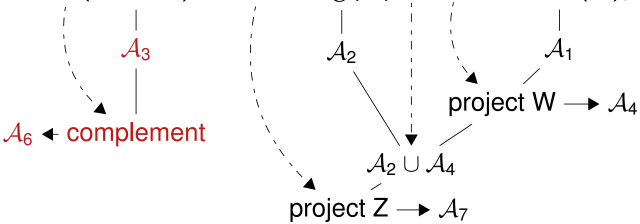


Construction of More Complex Automata

- Logical connectives mapped to automata operations.
  - More complex formulae handled by composition of automata.

- Example:

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

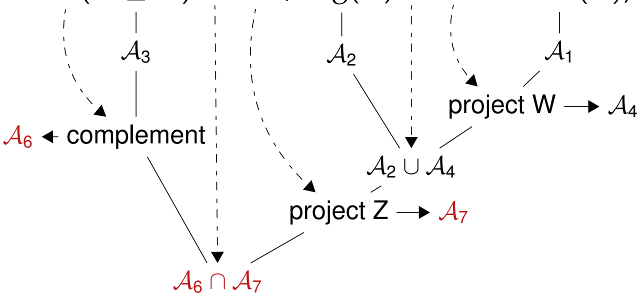


Construction of More Complex Automata

- Logical connectives mapped to automata operations.
  - More complex formulae handled by composition of automata.

- Example:

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



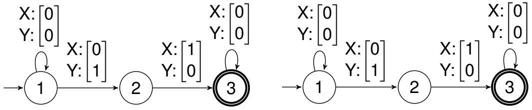
Handling Quantification

- Quantification handled by projection of a certain track.
  - Introduces non-determinism to automata.
  - Alternations require negation, hence determinization $\rightsquigarrow$ high cost.

Handling Quantification

- Quantification handled by projection of a certain track.
  - Introduces non-determinism to automata.
  - Alternations require negation, hence determinization $\rightsquigarrow$ high cost.

- An additional issue with projection of variables in an automaton:
  - after removing some tracks, some models need not be accepted:
    - some zero suffixes need not be present.
  - One needs to adjust the final states.

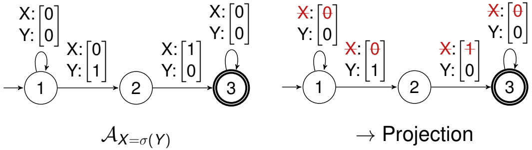


$\mathcal{A}_{X=\sigma(Y)}$ $\rightarrow$ Projection

Handling Quantification

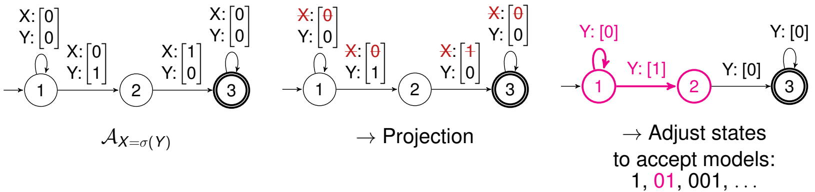
- Quantification handled by projection of a certain track.
  - Introduces non-determinism to automata.
  - Alternations require negation, hence determinization $\rightsquigarrow$ high cost.

- An additional issue with projection of variables in an automaton:
  - after removing some tracks, some models need not be accepted:
    - some zero suffixes need not be present.
  - One needs to adjust the final states.



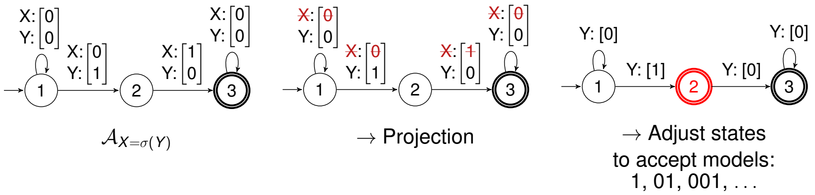
Handling Quantification

- Quantification handled by projection of a certain track.
  - Introduces non-determinism to automata.
  - Alternations require negation, hence determinization $\rightsquigarrow$ high cost.
- An additional issue with projection of variables in an automaton:
  - after removing some tracks, some models need not be accepted:
    - some zero suffixes need not be present.
  - One needs to adjust the final states.



Handling Quantification

- Quantification handled by projection of a certain track.
  - Introduces non-determinism to automata.
  - Alternations require negation, hence determinization $\rightsquigarrow$ high cost.
- An additional issue with projection of variables in an automaton:
  - after removing some tracks, some models need not be accepted:
    - some zero suffixes need not be present.
  - One needs to adjust the final states.



Deciding WS1S Using Non-deterministic Automata

- We consider only formulae in Prenex Normal Form (∃PNF).
- We focus on dealing with the prefix of alternating quantifiers.

Deciding WS1S Using Non-deterministic Automata

- We consider only formulae in Prenex Normal Form (∃PNF).
- We focus on dealing with the prefix of alternating quantifiers.
- For a formula φ with m quantifier alternations over *sets* of variables $\mathcal{X}_i \subseteq \mathbb{X}$,

$$\varphi = \underbrace{\neg \exists \mathcal{X}_m \neg \ldots \neg \exists \mathcal{X}_2 \underbrace{\neg \exists \mathcal{X}_1 : \varphi_0(\mathbb{X})}_{\varphi_1}}_{\varphi_m}$$

Deciding WS1S Using Non-deterministic Automata

- We consider only formulae in Prenex Normal Form (∃PNF).
- We focus on dealing with the prefix of alternating quantifiers.
- For a formula φ with m quantifier alternations over *sets* of variables $\mathcal{X}_i \subseteq \mathbb{X}$,

$$\varphi = \underbrace{\neg \exists \mathcal{X}_m \neg \ldots \neg \exists \mathcal{X}_2 \underbrace{\neg \exists \mathcal{X}_1 : \varphi_0(\mathbb{X})}_{\varphi_1}}_{\varphi_m}$$

→ construct a hierarchical family of automata defined as follows:

- $\mathcal{A}_{\varphi_0}$: a composition of atomic automata described before,
- $\mathcal{A}_{\varphi_m} = (\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2^{Q_0}}}}}}}_{m}, \Delta_m, I_m, F_m)$: described next.

Deciding WS1S Using Non-deterministic Automata

- We consider only formulae in Prenex Normal Form (∃PNF).
- We focus on dealing with the prefix of alternating quantifiers.
- For a formula φ with m quantifier alternations over *sets* of variables $\mathcal{X}_i \subseteq \mathbb{X}$,

$$\varphi = \underbrace{\neg \exists \mathcal{X}_m \neg \ldots \neg \exists \mathcal{X}_2 \underbrace{\neg \exists \mathcal{X}_1 : \varphi_0(\mathbb{X})}_{\varphi_1}}_{\varphi_m}$$

→ construct a hierarchical family of automata defined as follows:

- $\mathcal{A}_{\varphi_0}$: a composition of atomic automata described before,
- $\mathcal{A}_{\varphi_m} = (\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2^{Q_0}}}}}}}_{m}, \Delta_m, I_m, F_m)$: described next.
  - Intuition: on-the-fly projection and subset construction for all m levels (instead of doing it one-by-one), with antichain pruning.

More Intuition behind the Procedure

Correspondence between formulae and automata:

- A formula φ is valid iff $L(\mathcal{A}_\varphi) = \Sigma^*$.

More Intuition behind the Procedure

Correspondence between formulae and automata:

- A formula φ is valid iff $L(\mathcal{A}_\varphi) = \Sigma^*$.
- A formula φ is satisfiable iff $L(\mathcal{A}_\varphi) \neq \emptyset$.

More Intuition behind the Procedure

Correspondence between formulae and automata:

- A formula φ is valid iff $L(\mathcal{A}_\varphi) = \Sigma^*$.
- A formula φ is satisfiable iff $L(\mathcal{A}_\varphi) \neq \emptyset$.
- A formula φ is unsatisfiable iff $L(\mathcal{A}_\varphi) = \emptyset$.

More Intuition behind the Procedure

Correspondence between formulae and automata:

- A formula φ is valid iff $L(\mathcal{A}_\varphi) = \Sigma^*$.
- A formula φ is satisfiable iff $L(\mathcal{A}_\varphi) \neq \emptyset$.
- A formula φ is unsatisfiable iff $L(\mathcal{A}_\varphi) = \emptyset$.

A key observation for ground formulae:

- The symbols degenerate to the empty one: $\Sigma = \{[]\}$.

More Intuition behind the Procedure

Correspondence between formulae and automata:

- A formula φ is valid iff $L(\mathcal{A}_\varphi) = \Sigma^*$.
- A formula φ is satisfiable iff $L(\mathcal{A}_\varphi) \neq \emptyset$.
- A formula φ is unsatisfiable iff $L(\mathcal{A}_\varphi) = \emptyset$.

A key observation for ground formulae:

- The symbols degenerate to the empty one: $\Sigma = \{[]\}$.
- A ground formula φ is valid iff it is satisfiable iff $L(\mathcal{A}_\varphi) = []^*$.

More Intuition behind the Procedure

Correspondence between formulae and automata:

- A formula φ is valid iff $L(\mathcal{A}_\varphi) = \Sigma^*$.
- A formula φ is satisfiable iff $L(\mathcal{A}_\varphi) \neq \emptyset$.
- A formula φ is unsatisfiable iff $L(\mathcal{A}_\varphi) = \emptyset$.

A key observation for ground formulae:

- The symbols degenerate to the empty one: $\Sigma = \{[]\}$.
- A ground formula φ is valid iff it is satisfiable iff $L(\mathcal{A}_\varphi) = []^*$.
- A ground formula φ is satisfiable iff $L(\mathcal{A}_\varphi) = []^*$ iff $I_m \cap F_m \neq \emptyset$.

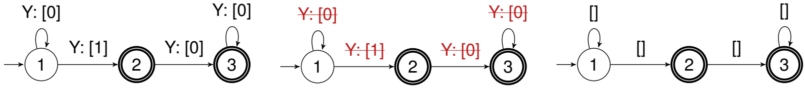
More Intuition behind the Procedure

Correspondence between formulae and automata:

- A formula φ is valid iff $L(\mathcal{A}_\varphi) = \Sigma^*$.
- A formula φ is satisfiable iff $L(\mathcal{A}_\varphi) \neq \emptyset$.
- A formula φ is unsatisfiable iff $L(\mathcal{A}_\varphi) = \emptyset$.

A key observation for ground formulae:

- The symbols degenerate to the empty one: $\Sigma = \{[]\}$.
- A ground formula φ is valid iff it is satisfiable iff $L(\mathcal{A}_\varphi) = []^*$.
- A ground formula φ is satisfiable iff $L(\mathcal{A}_\varphi) = []^*$ iff $I_m \cap F_m \neq \emptyset$.



$\mathcal{A}_{\exists X.X=\sigma(Y)}$ $\rightarrow$ Projection $\mathcal{A}_{\exists Y,X.X=\sigma(Y)}$

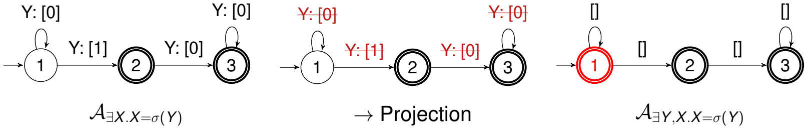
More Intuition behind the Procedure

Correspondence between formulae and automata:

- A formula φ is valid iff $L(\mathcal{A}_\varphi) = \Sigma^*$.
- A formula φ is satisfiable iff $L(\mathcal{A}_\varphi) \neq \emptyset$.
- A formula φ is unsatisfiable iff $L(\mathcal{A}_\varphi) = \emptyset$.

A key observation for ground formulae:

- The symbols degenerate to the empty one: $\Sigma = \{[]\}$.
- A ground formula φ is valid iff it is satisfiable iff $L(\mathcal{A}_\varphi) = []^*$.
- A ground formula φ is satisfiable iff $L(\mathcal{A}_\varphi) = []^*$ iff $I_m \cap F_m \neq \emptyset$.



Construction the set of Initial States I_m

- Constructing the whole automaton for φ_m is not necessary!
  - We construct the sets of initial/final states only,
  - and test whether they intersect.

Construction the set of Initial States I_m

- Constructing the whole automaton for φ_m is not necessary!
  - We construct the sets of initial/final states only,
  - and test whether they intersect.
- Construction of initial states is straightforward; starting from I_0:

Construction the set of Initial States I_m

- Constructing the whole automaton for φ_m is not necessary!
  - We construct the sets of initial/final states only,
  - and test whether they intersect.
- Construction of initial states is straightforward; starting from I_0:
  - $I_1 = \{I_0\}$

Construction the set of Initial States I_m

- Constructing the whole automaton for φ_m is not necessary!
  - We construct the sets of initial/final states only,
  - and test whether they intersect.

- Construction of initial states is straightforward; starting from I_0:
  - $I_1 = \{I_0\}$
  - $I_2 = \{I_1\} = \{\{I_0\}\}$

Construction the set of Initial States I_m

- Constructing the whole automaton for φ_m is not necessary!
  - We construct the sets of initial/final states only,
  - and test whether they intersect.

- Construction of initial states is straightforward; starting from I_0:
  - $I_1 = \{I_0\}$
  - $I_2 = \{I_1\} = \{\{I_0\}\}$

  $\vdots$
  - $I_m = \{I_{m-1}\} = \underbrace{\{\{\ldots\{}_{m} I_0\}\ldots\}\}$
    - No space for the subset construction to construct anything else!

Construction the set of Initial States I_m

- Constructing the whole automaton for φ_m is not necessary!
  - We construct the sets of initial/final states only,
  - and test whether they intersect.

- Construction of initial states is straightforward; starting from I_0:
  - $I_1 = \{I_0\}$
  - $I_2 = \{I_1\} = \{\{I_0\}\}$

    $\vdots$
  - $I_m = \{I_{m-1}\} = \underbrace{\{\{\ldots\{}_{m} I_0\}\ldots\}\}$
    - No space for the subset construction to construct anything else!

- Final states are more tricky:
  - a need to saturate after projection as described previously,
  - a lot of space for constructing different sets of sets of ... of states,
  - a need of switching the acceptance mode.

Introduction to the Computation of Final States (1/3)

- Given a formula $\varphi = \neg \exists \mathcal{X}_m \neg \ldots \neg \exists \mathcal{X}_2 \neg \exists \mathcal{X}_1 : \varphi_0(\mathbb{X})$ in $\exists$PNF,
  - start by constructing the base automaton for φ_0.

Introduction to the Computation of Final States (1/3)

- Given a formula $\varphi = \neg\,\exists\mathcal{X}_m\,\neg\ldots\neg\,\exists\mathcal{X}_2\,\neg\,\exists\mathcal{X}_1 : \varphi_0(\mathbb{X})$ in $\exists$PNF,
  - start by constructing the base automaton for φ_0.

- Given final states F_i of level i,
  - compute the set $F_i^{\exists}$ of their predecessors over 0 (pre_0) after projecting $\mathcal{X}_{i+1}$,

Introduction to the Computation of Final States (1/3)

- Given a formula $\varphi = \neg\exists\mathcal{X}_m \neg \ldots \neg\exists\mathcal{X}_2 \neg\exists\mathcal{X}_1 : \varphi_0(\mathbb{X})$ in $\exists$PNF,
  - start by constructing the base automaton for φ_0.

- Given final states F_i of level i,
  - compute the set $F_i^{\exists}$ of their predecessors over 0 (pre_0) after projecting $\mathcal{X}_{i+1}$,
  - after subset construction, any set of states of level i containing a state from $F_i^{\exists}$ is final,

Introduction to the Computation of Final States (1/3)

- Given a formula $\varphi = \neg \exists \mathcal{X}_m \neg \ldots \neg \exists \mathcal{X}_2 \neg \exists \mathcal{X}_1 : \varphi_0(\mathbb{X})$ in $\exists$PNF,
  - start by constructing the base automaton for φ_0.

- Given final states F_i of level i,
  - compute the set $F_i^{\exists}$ of their predecessors over 0 (pre_0) after projecting $\mathcal{X}_{i+1}$,
  - after subset construction, any set of states of level i containing a state from $F_i^{\exists}$ is final,
  - after negation, such sets become non-final.

Introduction to the Computation of Final States (1/3)

- Given a formula $\varphi = \neg \exists \mathcal{X}_m \neg \ldots \neg \exists \mathcal{X}_2 \neg \exists \mathcal{X}_1 : \varphi_0(\mathbb{X})$ in $\exists$PNF,
  - start by constructing the base automaton for φ_0.

- Given final states F_i of level i,
  - compute the set $F_i^{\exists}$ of their predecessors over 0 (pre_0) after projecting $\mathcal{X}_{i+1}$,
  - after subset construction, any set of states of level i containing a state from $F_i^{\exists}$ is final,
  - after negation, such sets become non-final.
    - We stay with non-final states.

Introduction to the Computation of Final States (1/3)

- Given a formula $\varphi = \neg \exists \mathcal{X}_m \neg \ldots \neg \exists \mathcal{X}_2 \neg \exists \mathcal{X}_1 : \varphi_0(\mathbb{X})$ in $\exists$PNF,
  - start by constructing the base automaton for φ_0.

- Given final states F_i of level i,
  - compute the set $F_i^{\exists}$ of their predecessors over 0 (*pre*$_0$) after projecting $\mathcal{X}_{i+1}$,
  - after subset construction, any set of states of level i containing a state from $F_i^{\exists}$ is final,
  - after negation, such sets become non-final.
    - We stay with non-final states.
  - Hence, non-final states $N_{i+1} = \uparrow \{\{q\} \mid q \in F_i^{\exists}\}$,
    - the upward closed set with the set of generators (minimal elements) $\{\{q\} \mid q \in F_i^{\exists}\}$.

Introduction to the Computation of Final States (1/3)

- Given a formula $\varphi = \neg\exists\mathcal{X}_m\neg\ldots\neg\exists\mathcal{X}_2\neg\exists\mathcal{X}_1 : \varphi_0(\mathbb{X})$ in $\exists$PNF,
  - start by constructing the base automaton for φ_0.

- Given final states F_i of level i,
  - compute the set $F_i^{\exists}$ of their predecessors over 0 (pre_0) after projecting $\mathcal{X}_{i+1}$,
  - after subset construction, any set of states of level i containing a state from $F_i^{\exists}$ is final,
  - after negation, such sets become non-final.
    - We stay with non-final states.
  - Hence, non-final states $N_{i+1} = \uparrow\{\{q\} \mid q \in F_i^{\exists}\}$,
    - the upward closed set with the set of generators (minimal elements) $\{\{q\} \mid q \in F_i^{\exists}\}$.
  - i.e., $N_{i+1} = \uparrow\coprod\{F_i^{\exists}\}$ – choice (unordered Cartesian product).

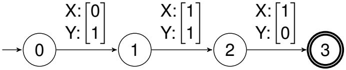
Introduction to the Computation of Final States (1/3)

- Given a formula $\varphi = \neg \exists \mathcal{X}_m \neg \ldots \neg \exists \mathcal{X}_2 \neg \exists \mathcal{X}_1 : \varphi_0(\mathbb{X})$ in $\exists$PNF,
  - start by constructing the base automaton for φ_0.

- Given final states F_i of level i,
  - compute the set $F_i^{\exists}$ of their predecessors over 0 (pre_0) after projecting $\mathcal{X}_{i+1}$,
  - after subset construction, any set of states of level i containing a state from $F_i^{\exists}$ is final,
  - after negation, such sets become non-final.
    - We stay with non-final states.
  - Hence, non-final states $N_{i+1} = \uparrow \{\{q\} \mid q \in F_i^{\exists}\}$,
    - the upward closed set with the set of generators (minimal elements) $\{\{q\} \mid q \in F_i^{\exists}\}$.
  - i.e., $N_{i+1} = \uparrow \coprod \{F_i^{\exists}\}$ – choice (unordered Cartesian product).
    - Let $\mathcal{Q} = \{Q_1, ..., Q_n\}$, $\coprod \mathcal{Q} = \{\{q_1, ..., q_n\} \mid (q_1, ..., q_n) \in \prod_{i=1}^{n} Q_i\}$.

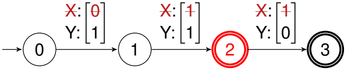
Introduction to the Computation of Final States (2/3)

- Example: projection of X and negation on the FA:



Introduction to the Computation of Final States (2/3)

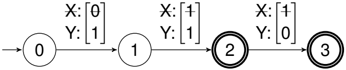
- Example: projection of X and negation on the FA:



- After projection:
  - $F_0^{\exists} = \{2, 3\}$.

Introduction to the Computation of Final States (2/3)

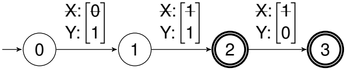
- Example: projection of X and negation on the FA:



- After projection:
  - $F_0^{\exists} = \{2, 3\}$.
- After negation:
  - $N_1 = \{\{2\}, \{3\}, \{2, 0\}, \{3, 0\}, \dots \{2, 3, 0\}, \{2, 3, 1\}, \dots \{0, 1, 2, 3\}\}$,

Introduction to the Computation of Final States (2/3)

- Example: projection of X and negation on the FA:



- After projection:
  - $F_0^{\exists} = \{2, 3\}$.
- After negation:
  - $N_1 = \{\{2\}, \{3\}, \{2, 0\}, \{3, 0\}, \ldots \{2, 3, 0\}, \{2, 3, 1\}, \ldots \{0, 1, 2, 3\}\}$,
  - $N_1 = \ \uparrow \{\{2\}, \{3\}\} = \ \uparrow \coprod \{\{2, 3\}\}$.

Introduction to the Computation of Final States (3/3)

- Given non-final states N_i of level i,
  - compute the set $N_i^{\exists}$ of their controllable predecessors over 0 ($cpre_0$) after projecting $\mathcal{X}_{i+1}$,
    - only states that cannot get to a final state stay non-final,

Introduction to the Computation of Final States (3/3)

- Given non-final states N_i of level i,
  - compute the set $N_i^{\exists}$ of their controllable predecessors over 0 ($cpre_0$) after projecting $\mathcal{X}_{i+1}$,
    - only states that cannot get to a final state stay non-final,
  - after subset construction, any set of states of level i consisting of non-final states of $N_i^{\exists}$ is non-final,

Introduction to the Computation of Final States (3/3)

- Given non-final states N_i of level i,
  - compute the set $N_i^{\exists}$ of their controllable predecessors over 0 ($cpre_0$) after projecting $\mathcal{X}_{i+1}$,
    - only states that cannot get to a final state stay non-final,
  - after subset construction, any set of states of level i consisting of non-final states of $N_i^{\exists}$ is non-final,
  - after negation, any such set becomes final.

Introduction to the Computation of Final States (3/3)

- Given non-final states N_i of level i,
  - compute the set $N_i^{\exists}$ of their controllable predecessors over 0 ($cpre_0$) after projecting $\mathcal{X}_{i+1}$,
    - only states that cannot get to a final state stay non-final,
  - after subset construction, any set of states of level i consisting of non-final states of $N_i^{\exists}$ is non-final,
  - after negation, any such set becomes final.
  - Hence, final states $F_{i+1} = \downarrow\{N_i^{\exists}\}$ (downward closure).

Introduction to the Computation of Final States (3/3)

- Given non-final states N_i of level i,
  - compute the set $N_i^{\exists}$ of their controllable predecessors over 0 ($cpre_0$) after projecting $\mathcal{X}_{i+1}$,
    - only states that cannot get to a final state stay non-final,
  - after subset construction, any set of states of level i consisting of non-final states of $N_i^{\exists}$ is non-final,
  - after negation, any such set becomes final.
  - Hence, final states $F_{i+1} = \downarrow\{N_i^{\exists}\}$ (downward closure).

- Continue with iterating the computation of non-final states from final, final from non-final, ...

Introduction to the Computation of Final States (3/3)

- Given non-final states N_i of level i,
  - compute the set $N_i^{\exists}$ of their controllable predecessors over 0 ($cpre_0$) after projecting $\mathcal{X}_{i+1}$,
    - only states that cannot get to a final state stay non-final,
  - after subset construction, any set of states of level i consisting of non-final states of $N_i^{\exists}$ is non-final,
  - after negation, any such set becomes final.
  - Hence, final states $F_{i+1} = \downarrow\{N_i^{\exists}\}$ (downward closure).

- Continue with iterating the computation of non-final states from final, final from non-final, ...

- Do not enumerate the sets F_i / N_i:
  - use symbolic encoding via expressions with the $\uparrow \coprod$ / $\downarrow$ operators.
  - A form of antichain reduction: keeping minimal/maximal elements.

Computing Final States F_m of Formula φ_m

- Given $\varphi = \neg \exists \mathcal{X}_m \neg \ldots \neg \exists \mathcal{X}_2 \neg \exists \mathcal{X}_1 : \varphi_0(\mathbb{X})$.

Computing Final States F_m of Formula φ_m

- Given $\varphi = \neg \exists \mathcal{X}_m \neg \ldots \neg \exists \mathcal{X}_2 \neg \exists \mathcal{X}_1 : \varphi_0(\mathbb{X})$.

1. Add final states after $\exists$: $F_0^{\exists} = \{\mu Z. F \cup \mathtt{pre}_0(Z)\}$.

Computing Final States F_m of Formula φ_m

- Given $\varphi = \neg \exists \mathcal{X}_m \neg \ldots \neg \exists \mathcal{X}_2 \neg \exists \mathcal{X}_1 : \varphi_0(\mathbb{X})$.

1. Add final states after $\exists$: $F_0^{\exists} = \{\mu Z . F \cup \mathtt{pre}_0(Z)\}$.
2. Negate the final states: $N_1 = \uparrow \coprod \{F_0^{\exists}\}$.

Computing Final States F_m of Formula φ_m

- Given $\varphi = \neg \exists \mathcal{X}_m \neg \ldots \neg \exists \mathcal{X}_2 \neg \exists \mathcal{X}_1 : \varphi_0(\mathbb{X})$.

1. Add final states after $\exists$: $F_0^{\exists} = \{\mu Z.F \cup \mathtt{pre}_0(Z)\}$.
2. Negate the final states: $N_1 = \uparrow \coprod \{F_0^{\exists}\}$.
3. Prune non-final states after $\exists$: $N_1^{\exists} = \{\nu Z.N_1 \cap \mathtt{cpre}_0(Z)\}$.

Computing Final States F_m of Formula φ_m

- Given $\varphi = \neg\exists\mathcal{X}_m\neg\ldots\neg\exists\mathcal{X}_2\neg\exists\mathcal{X}_1 : \varphi_0(\mathbb{X})$.

1. Add final states after $\exists$: $F_0^{\exists} = \{\mu Z.F \cup \mathtt{pre}_0(Z)\}$.
2. Negate the final states: $N_1 = \uparrow \coprod\{F_0^{\exists}\}$.
3. Prune non-final states after $\exists$: $N_1^{\exists} = \{\nu Z.N_1 \cap \mathtt{cpre}_0(Z)\}$.
4. Negate the non-final states: $F_2 = \downarrow\{N_1^{\exists}\}$.

Computing Final States F_m of Formula φ_m

- Given $\varphi = \neg\exists\mathcal{X}_m \neg \ldots \neg\exists\mathcal{X}_2 \neg\exists\mathcal{X}_1 : \varphi_0(\mathbb{X})$.

1. Add final states after $\exists$: $F_0^{\exists} = \{\mu Z.F \cup \mathtt{pre}_0(Z)\}$.
2. Negate the final states: $N_1 = \uparrow \coprod\{F_0^{\exists}\}$.
3. Prune non-final states after $\exists$: $N_1^{\exists} = \{\nu Z.N_1 \cap \mathtt{cpre}_0(Z)\}$.
4. Negate the non-final states: $F_2 = \downarrow\{N_1^{\exists}\}$.

⋮

5. Keep alternating between computing final and non-final states until F_m as follows:
   - $F_{i+1} = \downarrow\{\nu Z.N_i \cap \mathtt{cpre}_0(Z)\}$,
   - $N_{i+1} = \uparrow \coprod\{\mu Z.F_i \cup \mathtt{pre}_0(Z)\}$.

Computing Predecessors of Symbolic States

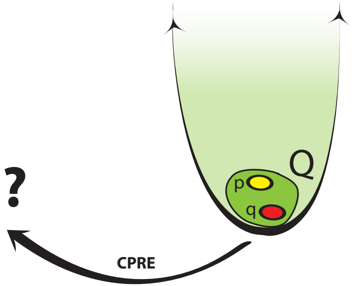
- (Controllable) predecessors of symbolic states can be computed without enumerating their elements.

Computing Predecessors of Symbolic States

- (Controllable) predecessors of symbolic states can be computed without enumerating their elements.
- E.g., note that $\mathtt{cpre}_{\mathbf{0}}(\uparrow\{Q\}) = \uparrow \coprod \{\mathtt{pre}_{\mathbf{0}}(q) \mid q \in Q\}$:

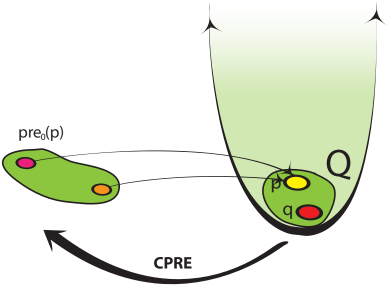
Computing Predecessors of Symbolic States

- (Controllable) predecessors of symbolic states can be computed without enumerating their elements.
- E.g., note that $\mathtt{cpre}_{\mathbf{0}}(\uparrow\{Q\}) = \uparrow \coprod \{\mathtt{pre}_{\mathbf{0}}(q) \mid q \in Q\}$:



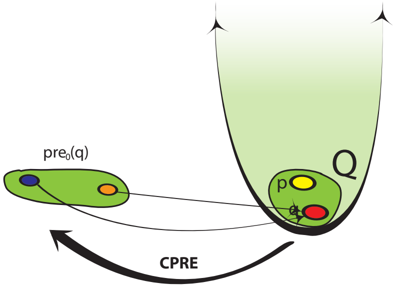
Computing Predecessors of Symbolic States

- (Controllable) predecessors of symbolic states can be computed without enumerating their elements.
- E.g., note that $\mathtt{cpre}_0(\uparrow\{Q\}) = \uparrow \coprod \{\mathtt{pre}_0(q) \mid q \in Q\}$:



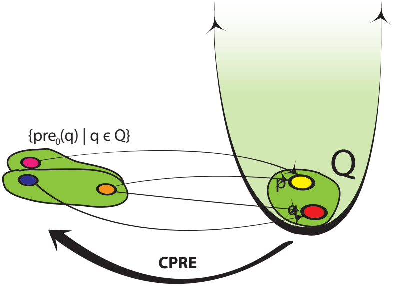
Computing Predecessors of Symbolic States

- (Controllable) predecessors of symbolic states can be computed without enumerating their elements.
- E.g., note that $\mathtt{cpre}_0(\uparrow\{Q\}) = \uparrow\coprod\{\mathtt{pre}_0(q) \mid q \in Q\}$:



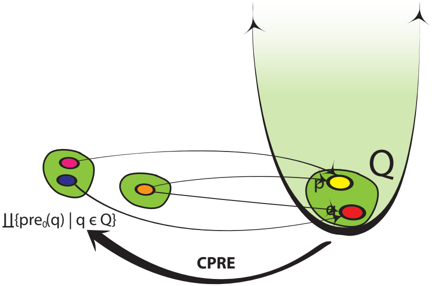
Computing Predecessors of Symbolic States

- (Controllable) predecessors of symbolic states can be computed without enumerating their elements.
- E.g., note that $\mathtt{cpre}_0(\uparrow\{Q\}) = \uparrow \coprod \{\mathtt{pre}_0(q) \mid q \in Q\}$:



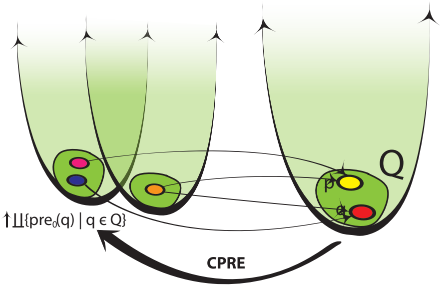
Computing Predecessors of Symbolic States

- (Controllable) predecessors of symbolic states can be computed without enumerating their elements.
- E.g., note that $\mathtt{cpre}_0(\uparrow \{Q\}) = \uparrow \coprod \{\mathtt{pre}_0(q) \mid q \in Q\}$:



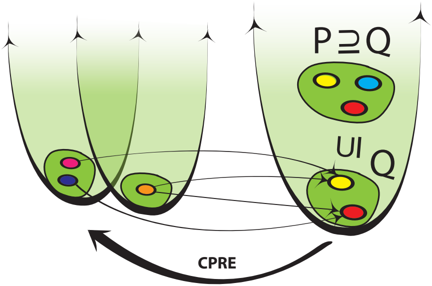
Computing Predecessors of Symbolic States

- (Controllable) predecessors of symbolic states can be computed without enumerating their elements.
- E.g., note that $\mathtt{cpre}_0(\uparrow\{Q\}) = \uparrow\coprod\{\mathtt{pre}_0(q) \mid q \in Q\}$:



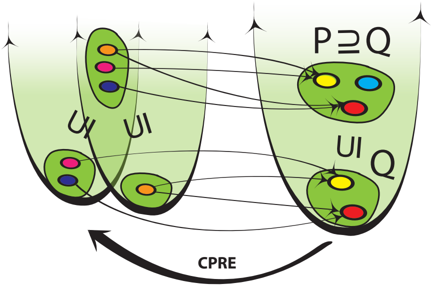
Computing Predecessors of Symbolic States

- (Controllable) predecessors of symbolic states can be computed without enumerating their elements.
- E.g., note that $\mathtt{cpre}_0(\uparrow\{Q\}) = \uparrow \coprod \{\mathtt{pre}_0(q) \mid q \in Q\}$:



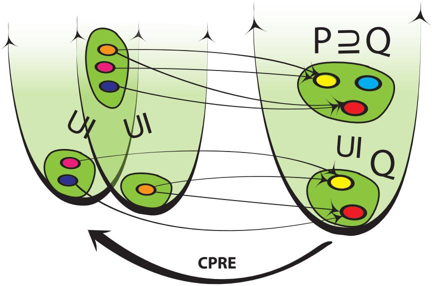
Computing Predecessors of Symbolic States

- (Controllable) predecessors of symbolic states can be computed without enumerating their elements.
- E.g., note that $\mathtt{cpre}_0(\uparrow\{Q\}) = \uparrow\coprod\{\mathtt{pre}_0(q) \mid q \in Q\}$:



Computing Predecessors of Symbolic States

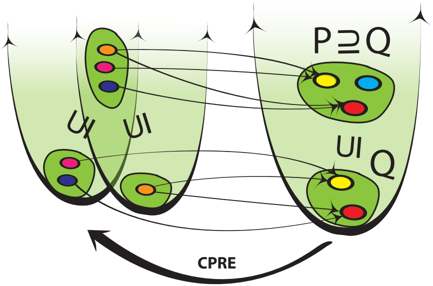
- (Controllable) predecessors of symbolic states can be computed without enumerating their elements.
- E.g., note that $\mathtt{cpre}_0(\uparrow\{Q\}) = \uparrow \coprod \{\mathtt{pre}_0(q) \mid q \in Q\}$:



- Likewise for the predecessors of a downward closed set.

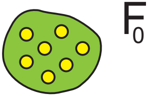
Computing Predecessors of Symbolic States

- (Controllable) predecessors of symbolic states can be computed without enumerating their elements.
- E.g., note that $\mathtt{cpre}_0(\uparrow\{Q\}) = \uparrow \coprod \{\mathtt{pre}_0(q) \mid q \in Q\}$:

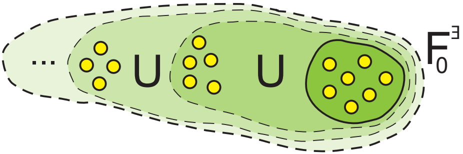


- Likewise for the predecessors of a downward closed set.
- Can be adapted for symbolic states with the needed structure.

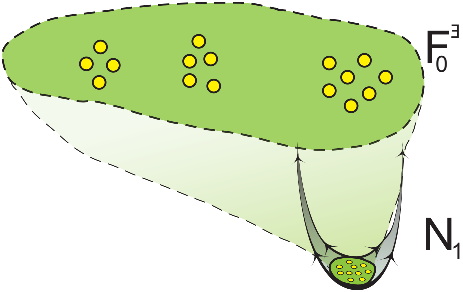
A Summary of the Inner Structure of F_m



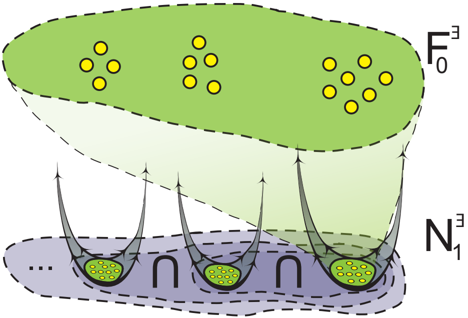
A Summary of the Inner Structure of F_m



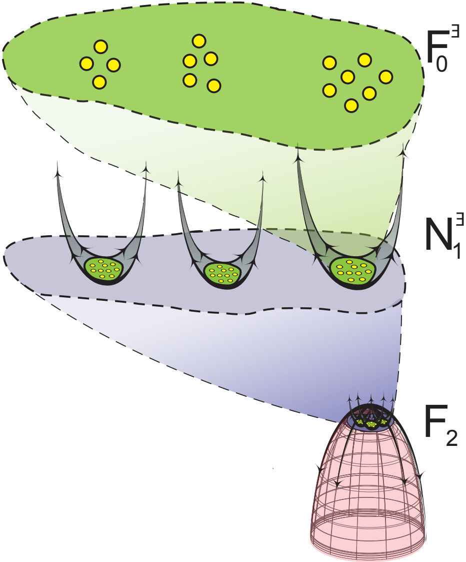
A Summary of the Inner Structure of F_m



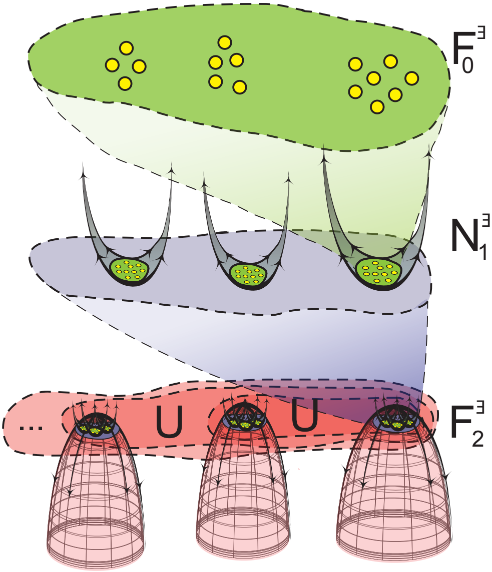
A Summary of the Inner Structure of F_m



A Summary of the Inner Structure of F_m



A Summary of the Inner Structure of F_m



State Space Pruning

- Sets of states on the various levels of the subset construction encoded as up(down)ward closed sets given by their generators.
  - First source of reduction.
  - Can be viewed as having an antichain flavour: generators and the bigger/smaller states are comparable.

State Space Pruning

- Sets of states on the various levels of the subset construction encoded as up(down)ward closed sets given by their generators.
  - First source of reduction.
  - Can be viewed as having an antichain flavour: generators and the bigger/smaller states are comparable.

- Further, we prune the generators subsumed by other generators:
  - the subsumption relation is computed on nested structure of symbolic representation of lower levels as follows.

$$\downarrow\mathbb{X} \subseteq \downarrow\mathbb{Y} \iff \forall X \in \mathbb{X} \,.\, \exists Y \in \mathbb{Y} \,.\, X \subseteq Y$$

$$\uparrow\coprod\mathbb{X} \subseteq \uparrow\coprod\mathbb{Y} \iff \forall Y \in \mathbb{Y} \,.\, \exists X \in \mathbb{X} \,.\, X \subseteq Y$$

Intersection of Initial and Final States

- Can also be done on the symbolic representation.

Intersection of Initial and Final States

- Can also be done on the symbolic representation.
- Depending on whether the number of alternations is even or odd, test:

Intersection of Initial and Final States

- Can also be done on the symbolic representation.
- Depending on whether the number of alternations is even or odd, test:
  - whether initial states intersect the final ones,
    - $I_m \cap F_m \neq \emptyset \;\Leftrightarrow\; I_{m-1} \in F_m$, or

Intersection of Initial and Final States

- Can also be done on the symbolic representation.
- Depending on whether the number of alternations is even or odd, test:
  - whether initial states intersect the final ones,
    - $I_m \cap F_m \neq \emptyset \;\Leftrightarrow\; I_{m-1} \in F_m$, or
  - or that initial states are not among the non-final ones,
    - reduces to an “and/or” search:

$$\{x\} \in \downarrow\mathbb{Y} \iff \exists Y \in \mathbb{Y} : x \in Y$$
$$\{x\} \in \uparrow\coprod\mathbb{Y} \iff \forall Y \in \mathbb{Y} : x \in Y$$

Implementations

- dWiNA (deciding WS1S using Non-deterministic Automata):
  - our prototype implementation,
  - antichain-based approach, with non-deterministic automata,
  - uses library VATA for manipulation with the automata:
    - uses degenerated tree automata.
- MONA:
  - (old but) state-of-the-art tool,
  - classic approach, with deterministic automata,
  - implemented range of optimizations like:
    - automata minimization,
    - automata caching,
    - using a DAG representation for formulae,
    - and many others.

Experiments on Formulae from Verification

- Compared with MONA:
  - on formulae from verification benchmarks,
    - taken from the `STRAND` tool (STRucture ANd Data),
    - encoding loop invariants of heap-manipulating programs,
  - in the general and ∃PNF form.

	MONA				dWiNA	
	Time [s]		Space [states]		Time [s]	Space [states]
benchmark	general	∃PNF	general	∃PNF	Prefix	Prefix
list-insert-after-loop	0.01	0.01	167	686	0.01	28
list-insert-before-head	0.01	0.01	43	152	0.01	38
list-insert-before-loop	0.01	0.01	103	1021	0.01	38
list-insert-in-loop	0.01	0.01	463	5015	0.01	59
list-reverse-after-loop	0.01	0.01	179	1 326	0.01	100
list-reverse-in-loop	0.02	0.47	1 311	70 278	0.02	260
bubblesort-else	0.01	0.45	1 285	12 071	0.01	14
bubblesort-if-else	0.02	2.17	4 260	116 760	0.23	234
bubblesort-if-if	0.12	5.29	8 390	233 372	1.14	28

Experiments with Generated Formulae

- Compared with MONA:
  - on generated formulae,
    - parametric, various lengths of prefix, number of alternations,
    - base formulae encode various set problems (transitivity, etc.),
  - in the ∃PNF form.
- An example of a generated formula:

$$\exists Y : \neg\exists X_1 \neg \ldots \neg\exists X_k, \ldots, X_n : \bigwedge_{1 \leq i < n} \left(X_i \subseteq Y \wedge X_i \subset X_{i+1}\right) \Rightarrow X_{i+1} \subseteq Y.$$

benchmark	MONA Time [s] general	MONA Time [s] ∃PNF	MONA Space [states] general	MONA Space [states] ∃PNF	dWiNA Time [s] Prefix	dWiNA Space [states] Prefix
1 alternation	-	0.11	-	10 718	0.01	39
2 alternations	-	0.20	-	25 517	0.01	44
3 alternations	-	0.57	-	60 924	0.01	50
4 alternations	-	1.79	-	145 765	0.02	58
5 alternations	-	4.98	-	349 314	0.02	70
6 alternations	-	**TO**	-	**TO**	0.47	90

Future Work

- Extension to WS2S.
- Generalization of the symbolic tree representation:
  - to process logical connectives,
  - to handle general (non-∃PNF) formulae.
- Syntactical optimizations:
  - using Direct Acyclic Graph (DAG) for representation of formulae,
  - anti-prenexing,
  - smarter conversion to ∃PNF, ...