New Algorithm for Weak Monadic Second-Order Logic on Inductive Structures

Łukasz Kaiser and Tobias Ganzow

Mathematische Grundlagen der Informatik
RWTH Aachen

Games

Oxford, 2010

Algorithm for What?

Input:

(1) **Weak monadic second-order logic formula φ**

- **WMSO** is first-order logic + quantifiers over **finite subsets**
- **Example:** $\forall X(x \in X \wedge (\forall z, v(z \in X \wedge E(z,v) \rightarrow v \in X)) \rightarrow y \in X)$

Algorithm for What?

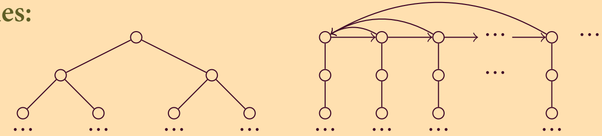
Input:

(1) **Weak monadic second-order logic formula φ**

- **WMSO** is first-order logic + quantifiers over **finite subsets**
- **Example:** $\forall X(x \in X \wedge (\forall z, v(z \in X \wedge E(z,v) \to v \in X)) \to y \in X)$

(2) **Inductive structure $\mathfrak{A}$**

- **Represented by equations**, e.g. **Tree** = **Tree** + **single node** + **Tree**
- **Examples:**



Algorithm for What?

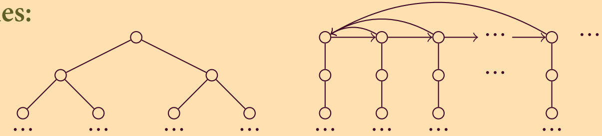
Input:

(1) **Weak monadic second-order logic formula φ**

- **WMSO** is first-order logic + quantifiers over **finite subsets**
- **Example:** $\forall X(x \in X \land (\forall z, v(z \in X \land E(z,v) \to v \in X)) \to y \in X)$

(2) **Inductive structure $\mathfrak{A}$**

- Represented by equations, e.g. **Tree** = **Tree** + **single node** + **Tree**
- **Examples:**



Output: $\mathfrak{A} \vDash \varphi$?

Algorithm for What?

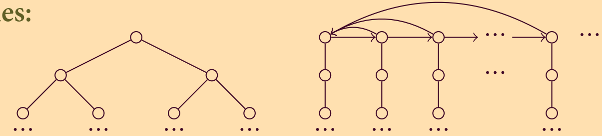
Input:

(1) **Weak monadic second-order logic formula φ**

- **WMSO** is first-order logic + quantifiers over **finite subsets**
- **Example:** $\forall X(x \in X \wedge (\forall z, v(z \in X \wedge E(z,v) \to v \in X)) \to y \in X)$

(2) **Inductive structure $\mathfrak{A}$**

- **Represented by equations**, e.g. **Tree** = **Tree** + **single node** + **Tree**
- **Examples:**



Output: $\mathfrak{A} \models \varphi$?

But this has been done a long time ago! Main method: **automata**

Automata

Automata are **beautiful** and **everywhere**

Running Automaton

Automata

Automata are beautiful and everywhere

Running Automaton

Algorithm for WMSO using automata

- for each formula $\varphi(X, Y, \ldots)$ automaton reads $\chi(X), \chi(Y), \ldots$
- use closure properties to build automaton bottom-up
- test for emptiness or universality

Motivation

Problems with automata

- In **verification**: need to communicate with **other solvers**
- For **data structures**: logical interpretation can be **costly**
- Technical limitations: **bottom-up**

Motivation

Problems with automata

- In **verification**: need to communicate with **other solvers**
- For **data structures**: logical interpretation can be **costly**
- Technical limitations: **bottom-up**

Our approach

- Based on **pure formula manipulation**
- **Avoids costly interpretations** (structure defined by equations)
- Very **simple**, costly computations on **propositional** level

Motivation

Problems with automata

- In **verification**: need to communicate with **other solvers**
- For **data structures**: logical interpretation can be **costly**
- Technical limitations: **bottom-up**

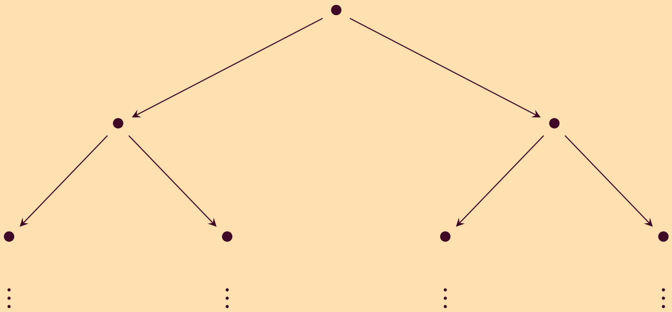
Our approach

- Based on **pure formula manipulation**
- **Avoids costly interpretations** (structure defined by equations)
- Very **simple**, costly computations on **propositional** level

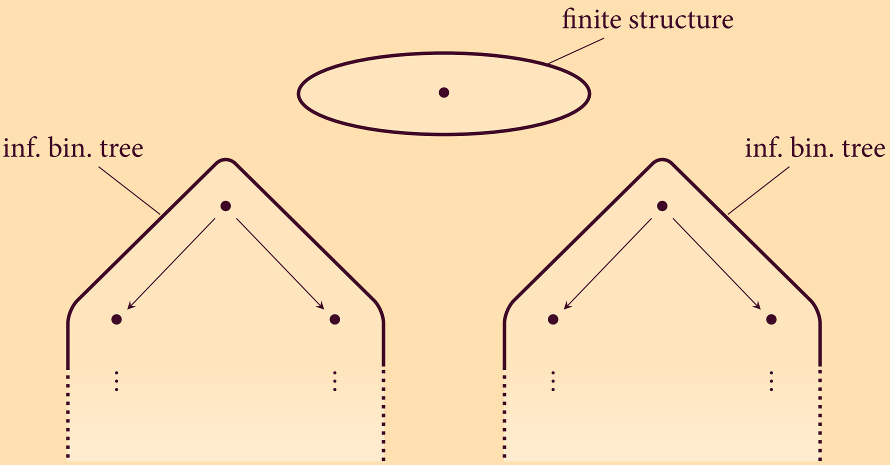
Compared to Automata

- In general a **similar method**
- But **explicit formulas** and **on the fly**
- Some problems **easier in this presentation** (**unbounding quantifier**)

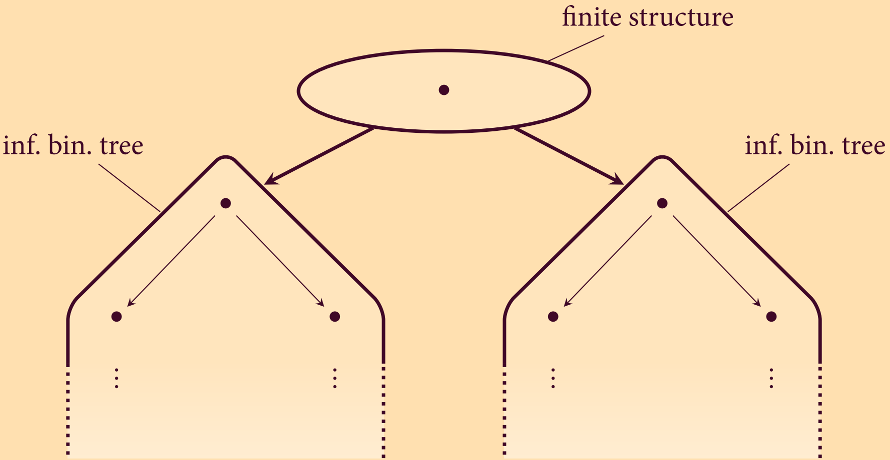
Inductive Structures — Intuition



Inductive Structures — Intuition



Inductive Structures — Intuition



Structure Equations

System of structure equations $\mathcal{D}$ over $\tau = \{R_1, \ldots, R_\ell\}$:

$$\mathcal{D} = \begin{cases} \Lambda^1 & = & \mathfrak{A}^1_1 & \oplus & \mathfrak{A}^1_2 & \oplus & \ldots & \oplus & \mathfrak{A}^1_{k_1} & \text{with } f^1_1, \ldots, f^1_\ell \\ \vdots & & & & & \vdots & & & & \vdots \\ \Lambda^n & = & \mathfrak{A}^n_1 & \oplus & \mathfrak{A}^n_2 & \oplus & \ldots & \oplus & \mathfrak{A}^n_{k_n} & \text{with } f^n_1, \ldots, f^n_\ell \end{cases}$$

- each $\mathfrak{A}^i_j$ is a **finite structure** or a **formal variable** in $\Lambda^1, \ldots, \Lambda^n$
- each f^i_j is a function $\{1, \ldots, k_i\}^{r_j} \to \{\bot, \top\}$

Solution of $\mathcal{D}$: $\mathcal{S}(\mathcal{D}) = (\mathfrak{A}_1, \ldots, \mathfrak{A}_n)$

Example: Binary Tree with Predicates

$$\mathcal{D} = \begin{cases} \Lambda^T = \Lambda^L \oplus R \bullet \oplus \Lambda^R \\ \Lambda^L = \Lambda^L \oplus S_L \bullet \oplus \Lambda^R \\ \Lambda^R = \Lambda^L \oplus S_R \bullet \oplus \Lambda^R \end{cases} \qquad f_{<}(i,j) = \begin{cases} \top & \text{if } i = 2 \\ & j \in \{1,3\} \\ \bot & \text{otherwise} \end{cases}$$

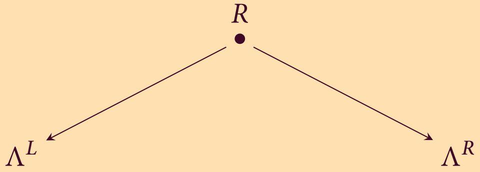
$\mathcal{S}_1(\mathcal{D})$:

Example: Binary Tree with Predicates

$$\mathcal{D} = \begin{cases} \Lambda^T = \Lambda^L \oplus R\,\bullet \oplus \Lambda^R \\ \Lambda^L = \Lambda^L \oplus S_L\,\bullet \oplus \Lambda^R \\ \Lambda^R = \Lambda^L \oplus S_R\,\bullet \oplus \Lambda^R \end{cases}$$

$$f_{<}(i, j) = \begin{cases} \top & \text{if } i = 2 \\ & j \in \{1, 3\} \\ \bot & \text{otherwise} \end{cases}$$

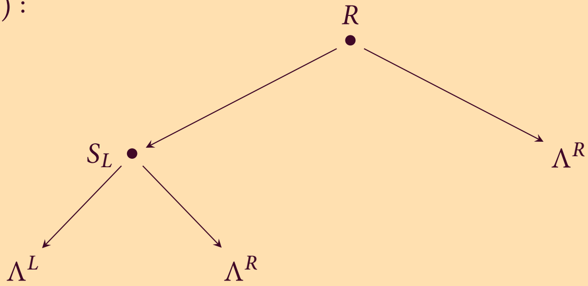
$\mathcal{S}_1(\mathcal{D})$:



Example: Binary Tree with Predicates

$$\mathcal{D} = \begin{cases} \Lambda^T = \Lambda^L \oplus R\bullet \oplus \Lambda^R \\ \Lambda^L = \Lambda^L \oplus S_L\bullet \oplus \Lambda^R \\ \Lambda^R = \Lambda^L \oplus S_R\bullet \oplus \Lambda^R \end{cases} \qquad f_{\prec}(i,j) = \begin{cases} \top & \text{if } i = 2 \\ & j \in \{1,3\} \\ \bot & \text{otherwise} \end{cases}$$

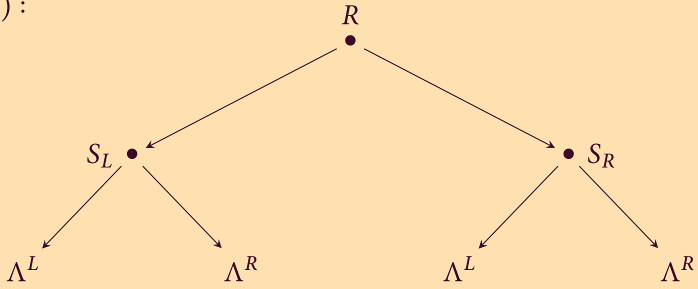
$\mathcal{S}_1(\mathcal{D})$:



Example: Binary Tree with Predicates

$$\mathcal{D} = \begin{cases} \Lambda^T = \Lambda^L \oplus R\bullet \oplus \Lambda^R \\ \Lambda^L = \Lambda^L \oplus S_L\bullet \oplus \Lambda^R \\ \Lambda^R = \Lambda^L \oplus S_R\bullet \oplus \Lambda^R \end{cases} \qquad f_{\prec}(i, j) = \begin{cases} \top & \text{if } i = 2 \\ & j \in \{1, 3\} \\ \bot & \text{otherwise} \end{cases}$$

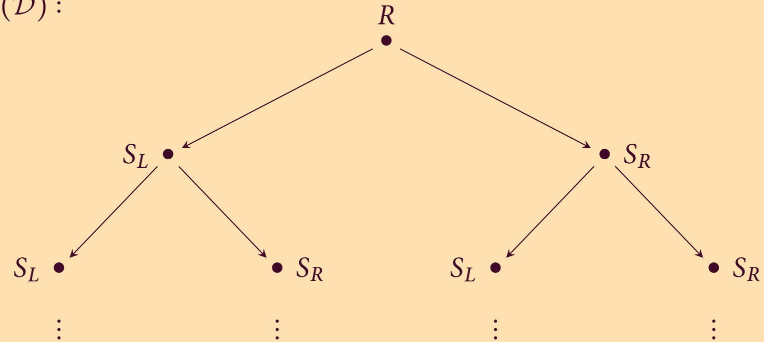
$\mathcal{S}_1(\mathcal{D})$:



Example: Binary Tree with Predicates

$$\mathcal{D} = \begin{cases} \Lambda^T = \Lambda^L \oplus R\bullet \oplus \Lambda^R \\ \Lambda^L = \Lambda^L \oplus S_L\bullet \oplus \Lambda^R \\ \Lambda^R = \Lambda^L \oplus S_R\bullet \oplus \Lambda^R \end{cases} \qquad f_{<}(i,j) = \begin{cases} \top & \text{if } i = 2 \\ & j \in \{1,3\} \\ \bot & \text{otherwise} \end{cases}$$

$\mathcal{S}_1(\mathcal{D})$:



$\mathcal{D}_m$-Decomposition

$$\mathcal{D} = \begin{cases} \vdots & & & & \vdots & & \\ \Lambda^m & = & \mathfrak{A}_1^m & \oplus & \dots & \oplus & \mathfrak{A}_k^m \\ \vdots & & & & \vdots & & \end{cases}$$

$$\mathcal{S}(\mathcal{D}) = (\mathfrak{A}_1, \dots, \mathfrak{A}_n)$$

A $\mathcal{D}_m$-**decomposition** of φ is a sequence

$$(\psi_1^1, \dots, \psi_k^1), \dots, (\psi_1^\ell, \dots, \psi_k^\ell)$$

such that

- $\mathfrak{A}_m \vDash \varphi \iff \exists i \forall j : \mathfrak{A}_m[j] \vDash \psi_j^i$
- $FV(\psi_j^i) \subseteq FV(\varphi)$
- $\mathrm{qr}(\psi_j^i) \leq \mathrm{qr}(\varphi)$

$\mathcal{D}_m$-Decomposition

$$\mathcal{D} = \begin{cases} \vdots & & & & \vdots & & \\ \Lambda^m & = & \mathfrak{A}_1^m & \oplus & \dots & \oplus & \mathfrak{A}_k^m \\ \vdots & & & & \vdots & & \end{cases}$$

$$\mathcal{S}(\mathcal{D}) = (\mathfrak{A}_1, \dots, \mathfrak{A}_n)$$

A $\mathcal{D}_m$-**decomposition** of φ is a sequence

$$(\psi_1^1, \dots, \psi_k^1), \dots, (\psi_1^\ell, \dots, \psi_k^\ell) \triangleq \bigvee_i \bigwedge_j \psi_j^i$$

such that

- $\mathfrak{A}_m \models \varphi \iff \exists i \forall j : \mathfrak{A}_m[j] \models \psi_j^i$
- $FV(\psi_j^i) \subseteq FV(\varphi)$
- $\mathrm{qr}(\psi_j^i) \leq \mathrm{qr}(\varphi)$

Computing a Decomposition

φ with $FV(\varphi) \in \{X_1, \dots, X_r\}$

$$\mathcal{D} = \begin{cases} \vdots & & & & \vdots & & & & \vdots \\ \Lambda^m & = & \mathfrak{A}_1^m & \oplus & \dots & \oplus & \mathfrak{A}_{k_1}^m & \text{with} & f_1^m, \dots, f_\ell^m \\ \vdots & & & & \vdots & & & & \vdots \end{cases}$$

Compute $\mathcal{D}_m$-decomposition by

(1) "split" variables in φ into several ones implicitly ranging over the components of the structure

(2) replace trivially true or false atoms and simplify (wrt. the restriction to components and the functions defining relations across components)

(3) compute the TNF (type normal form) of the resulting formula and transform it into DNF

Step 1 — Splitting

$$\mathcal{D} = \begin{cases} \vdots & & & & \vdots \\ \Lambda^m & = & \mathfrak{A}_1^m & \oplus & \dots & \oplus & \mathfrak{A}_k^m \\ \vdots & & & & \vdots \end{cases}$$

split(φ) is computed by transforming

$$\exists X\psi \;\mapsto\; \exists X^1 \dots X^k \psi[x \in X / \bigvee_{i=1}^{k} x \in X^i] \qquad \exists x\psi \;\mapsto\; \bigvee_{i=1}^{k} \exists x^i \psi[x/x^i]$$

$$\forall X\psi \;\mapsto\; \forall X^1 \dots X^k \psi[x \in X / \bigvee_{i=1}^{k} x \in X^i] \qquad \forall x\psi \;\mapsto\; \bigwedge_{i=1}^{k} \forall x^i \psi[x/x^i]$$

Semantics: x^i, X^i are implicitly restricted to the domain of the i-th component of the structure.

Step 1 — Splitting (on the Binary Tree)

$$\mathrm{split}\Big(\exists X \forall x \big(x \in X \to S_L x \wedge \forall y (y \prec x \to (Ry \vee S_R y))\big)\Big)$$

Step 1 — Splitting (on the Binary Tree)

$$\mathrm{split}\Big(\exists X \forall x \big(x \in X \to S_L x \wedge \forall y (y \prec x \to (Ry \vee S_R y))\big)\Big)$$

$$\begin{aligned}
= \exists X^1 \exists X^2 \exists X^3 \forall x^1 & \left(\bigvee_{i=1}^{3} x^1 \in X^i \to S_L x^1 \wedge \left(\begin{array}{r} \forall y^1 (y^1 \prec x^1 \to (Ry^1 \vee S_R y^1)) \\ \wedge \forall y^2 (y^2 \prec x^1 \to (Ry^2 \vee S_R y^2)) \\ \wedge \forall y^3 (y^3 \prec x^1 \to (Ry^3 \vee S_R y^3)) \end{array} \right) \right) \\
\wedge \forall x^2 & \left(\bigvee_{i=1}^{3} x^2 \in X^i \to S_L x^2 \wedge \left(\begin{array}{r} \forall y^1 (y^1 \prec x^2 \to (Ry^1 \vee S_R y^1)) \\ \wedge \forall y^2 (y^2 \prec x^2 \to (Ry^2 \vee S_R y^2)) \\ \wedge \forall y^3 (y^3 \prec x^2 \to (Ry^3 \vee S_R y^3)) \end{array} \right) \right) \\
\wedge \forall x^3 & \left(\bigvee_{i=1}^{3} x^3 \in X^i \to S_L x^3 \wedge \left(\begin{array}{r} \forall y^1 (y^1 \prec x^3 \to (Ry^1 \vee S_R y^1)) \\ \wedge \forall y^2 (y^2 \prec x^3 \to (Ry^2 \vee S_R y^2)) \\ \wedge \forall y^3 (y^3 \prec x^3 \to (Ry^3 \vee S_R y^3)) \end{array} \right) \right)
\end{aligned}$$

Step 2 — Reducing the Split Formula

$$\text{split}\Big(\exists X \forall x\big(x \in X \to S_L x \wedge \forall y(y \prec x \to (Ry \vee S_R y))\big)\Big)$$

$$= \exists X^1 \exists X^2 \exists X^3 \forall x^1 \left(\bigvee_{i=1}^{3} x^1 \in X^i \to S_L x^1 \wedge \left(\begin{array}{r} \forall y^1(y^1 \prec x^1 \to (Ry^1 \vee S_R y^1)) \\ \wedge \forall y^2(y^2 \prec x^1 \to (Ry^2 \vee S_R y^2)) \\ \wedge \forall y^3(y^3 \prec x^1 \to (Ry^3 \vee S_R y^3)) \end{array} \right)\right)$$

$$\wedge \forall x^2 \left(\bigvee_{i=1}^{3} x^2 \in X^i \to S_L x^2 \wedge \left(\begin{array}{r} \forall y^1(y^1 \prec x^2 \to (Ry^1 \vee S_R y^1)) \\ \wedge \forall y^2(y^2 \prec x^2 \to (Ry^2 \vee S_R y^2)) \\ \wedge \forall y^3(y^3 \prec x^2 \to (Ry^3 \vee S_R y^3)) \end{array} \right)\right)$$

$$\wedge \forall x^3 \left(\bigvee_{i=1}^{3} x^3 \in X^i \to S_L x^3 \wedge \left(\begin{array}{r} \forall y^1(y^1 \prec x^3 \to (Ry^1 \vee S_R y^1)) \\ \wedge \forall y^2(y^2 \prec x^3 \to (Ry^2 \vee S_R y^2)) \\ \wedge \forall y^3(y^3 \prec x^3 \to (Ry^3 \vee S_R y^3)) \end{array} \right)\right)$$

$x^j \in X^i \equiv \bot$
if $i \neq j$

Step 2 — Reducing the Split Formula

$$\mathrm{split}\Big(\exists X \forall x\big(x \in X \to S_L x \wedge \forall y(y < x \to (Ry \vee S_R y))\big)\Big)$$

$$= \exists X^1 \exists X^2 \exists X^3 \forall x^1 \left(x^1 \in X^1 \to S_L x^1 \wedge \begin{pmatrix} \forall y^1(y^1 < x^1 \to (Ry^1 \vee S_R y^1)) \\ \wedge \forall y^2(y^2 < x^1 \to (Ry^2 \vee S_R y^2)) \\ \wedge \forall y^3(y^3 < x^1 \to (Ry^3 \vee S_R y^3)) \end{pmatrix} \right)$$
$$\wedge \forall x^2 \left(x^2 \in X^2 \to S_L x^2 \wedge \begin{pmatrix} \forall y^1(y^1 < x^2 \to (Ry^1 \vee S_R y^1)) \\ \wedge \forall y^2(y^2 < x^2 \to (Ry^2 \vee S_R y^2)) \\ \wedge \forall y^3(y^3 < x^2 \to (Ry^3 \vee S_R y^3)) \end{pmatrix} \right)$$
$$\wedge \forall x^3 \left(x^3 \in X^3 \to S_L x^3 \wedge \begin{pmatrix} \forall y^1(y^1 < x^3 \to (Ry^1 \vee S_R y^1)) \\ \wedge \forall y^2(y^2 < x^3 \to (Ry^2 \vee S_R y^2)) \\ \wedge \forall y^3(y^3 < x^3 \to (Ry^3 \vee S_R y^3)) \end{pmatrix} \right)$$

Step 2 — Reducing the Split Formula

$$\text{split}\Big(\exists X \forall x\big(x \in X \to S_L x \land \forall y(y \prec x \to (Ry \lor S_R y))\big)\Big)$$

$$= \exists X^1 \exists X^2 \exists X^3 \forall x^1 \left(x^1 \in X^1 \to S_L x^1 \land \begin{pmatrix} \forall y^1(y^1 \prec x^1 \to (Ry^1 \lor S_R y^1)) \\ \land \forall y^2(y^2 \prec x^1 \to (Ry^2 \lor S_R y^2)) \\ \land \forall y^3(y^3 \prec x^1 \to (Ry^3 \lor S_R y^3)) \end{pmatrix} \right)$$

$$\land \forall x^2 \left(x^2 \in X^2 \to S_L x^2 \land \begin{pmatrix} \forall y^1(y^1 \prec x^2 \to (Ry^1 \lor S_R y^1)) \\ \land \forall y^2(y^2 \prec x^2 \to (Ry^2 \lor S_R y^2)) \\ \land \forall y^3(y^3 \prec x^2 \to (Ry^3 \lor S_R y^3)) \end{pmatrix} \right)$$

$$\land \forall x^3 \left(x^3 \in X^3 \to S_L x^3 \land \begin{pmatrix} \forall y^1(y^1 \prec x^3 \to (Ry^1 \lor S_R y^1)) \\ \land \forall y^2(y^2 \prec x^3 \to (Ry^2 \lor S_R y^2)) \\ \land \forall y^3(y^3 \prec x^3 \to (Ry^3 \lor S_R y^3)) \end{pmatrix} \right)$$

for $i \neq j: \quad y^i \prec x^j \equiv f_\prec(i, j)$

Step 2 — Reducing the Split Formula

$$\text{split}\Big(\exists X \forall x\big(x \in X \to S_L x \wedge \forall y(y \prec x \to (Ry \vee S_R y))\big)\Big)$$

$$= \exists X^1 \exists X^2 \exists X^3 \quad \forall x^1\left(x^1 \in X^1 \to S_L x^1 \wedge \left(\begin{array}{r} \forall y^1(y^1 < x^1 \to (Ry^1 \vee S_R y^1)) \\ \wedge \forall y^2(Ry^2 \vee S_R y^2) \end{array}\right)\right)$$

$$\wedge \forall x^2\Big(x^2 \in X^2 \to S_L x^2 \wedge \big(\forall y^2(y^2 < x^2 \to (Ry^2 \vee S_R y^2))\big)\Big)$$

$$\wedge \forall x^3\left(x^3 \in X^3 \to S_L x^3 \wedge \left(\begin{array}{r} \forall y^2(Ry^2 \vee S_R y^2) \\ \wedge \forall y^3(y^3 < x^3 \to (Ry^3 \vee S_R y^3)) \end{array}\right)\right)$$

Step 3 — Type Normal Form

Positive Boolean combination of formulas of the form

$$\tau = R_i(\overline{x}) \mid \neg R_i(\overline{x}) \mid x \in X \mid x \notin X \mid$$
$$\exists x \mathcal{B}^+(\tau) \mid \exists X \mathcal{B}^+(\tau) \mid \forall x \mathcal{B}^+(\tau) \mid \forall X \mathcal{B}^+(\tau)$$

such that for **each** τ_i in the Boolean combination $\mathcal{B}^+(\tau_i)$, $x \in FV(\tau_i)$.

Step 3 — Type Normal Form

Positive Boolean combination of formulas of the form

$$\tau = R_i(\overline{x}) \mid \neg R_i(\overline{x}) \mid x \in X \mid x \notin X \mid$$
$$\exists x \mathcal{B}^+(\tau) \mid \exists X \mathcal{B}^+(\tau) \mid \forall x \mathcal{B}^+(\tau) \mid \forall X \mathcal{B}^+(\tau)$$

such that for **each** τ_i in the Boolean combination $\mathcal{B}^+(\tau_i)$, $x \in FV(\tau_i)$.

Example $\varphi = \exists x \big(Px \wedge (Qy \vee Rx) \big)$

Step 3 — Type Normal Form

Positive Boolean combination of formulas of the form

$$\tau = R_i(\overline{x}) \mid \neg R_i(\overline{x}) \mid x \in X \mid x \notin X \mid$$
$$\exists x \mathcal{B}^+(\tau) \mid \exists X \mathcal{B}^+(\tau) \mid \forall x \mathcal{B}^+(\tau) \mid \forall X \mathcal{B}^+(\tau)$$

such that for **each** τ_i in the Boolean combination $\mathcal{B}^+(\tau_i)$, $x \in FV(\tau_i)$.

Example

$$\varphi = \exists x\big(Px \wedge (Qy \vee Rx)\big)$$
$$\equiv \exists x\big((Px \wedge Qy) \vee (Px \wedge Rx)\big)$$

Step 3 — Type Normal Form

Positive Boolean combination of formulas of the form

$$\tau = R_i(\overline{x}) \mid \neg R_i(\overline{x}) \mid x \in X \mid x \notin X \mid$$
$$\exists x \mathcal{B}^+(\tau) \mid \exists X \mathcal{B}^+(\tau) \mid \forall x \mathcal{B}^+(\tau) \mid \forall X \mathcal{B}^+(\tau)$$

such that for **each** τ_i in the Boolean combination $\mathcal{B}^+(\tau_i)$, $x \in FV(\tau_i)$.

Example

$$\begin{aligned} \varphi &= \exists x \big(Px \wedge (Qy \vee Rx)\big) \\ &\equiv \exists x \big((Px \wedge Qy) \vee (Px \wedge Rx)\big) \\ &\equiv \exists x (Px \wedge Qy) \vee \exists x (Px \wedge Rx) \end{aligned}$$

Step 3 — Type Normal Form

Positive Boolean combination of formulas of the form

$$\tau = R_i(\overline{x}) \mid \neg R_i(\overline{x}) \mid x \in X \mid x \notin X \mid$$
$$\exists x \mathcal{B}^+(\tau) \mid \exists X \mathcal{B}^+(\tau) \mid \forall x \mathcal{B}^+(\tau) \mid \forall X \mathcal{B}^+(\tau)$$

such that for **each** τ_i in the Boolean combination $\mathcal{B}^+(\tau_i)$, $x \in FV(\tau_i)$.

Example

$$\begin{aligned} \varphi &= \exists x\big(Px \wedge (Qy \vee Rx)\big) \\ &\equiv \exists x\big((Px \wedge Qy) \vee (Px \wedge Rx)\big) \\ &\equiv \exists x(Px \wedge Qy) \vee \exists x(Px \wedge Rx) \\ \mathrm{TNF}(\varphi) &= \big(Qy \wedge \exists x Px\big) \vee \exists x\big(Px \wedge Rx\big) \end{aligned}$$

Step 3 — Type Normal Form

Lemma

For each φ there exists an equivalent ψ in TNF such that

- $\mathrm{qr}(\psi) \leq \mathrm{qr}(\varphi)$
- $\mathrm{atoms}(\psi) \subseteq \mathrm{atoms}(\varphi)$.

Step 3 — Type Normal Form

Lemma

For each φ there exists an equivalent ψ in TNF such that

- $\mathrm{qr}(\psi) \leq \mathrm{qr}(\varphi)$
- $\mathrm{atoms}(\psi) \subseteq \mathrm{atoms}(\varphi)$.

Lemma

Let φ be in TNF, $V_1, \ldots, V_n$ a partition of the variables such that variables appearing in the same atom are in the same V_i.
Then φ is a Boolean combination of formulas τ such that each τ contains only atoms with variables from one of the sets V_i.

$\rightsquigarrow$ TNF-conversion "sorts" subformulas according to the components they speak about, and we obtain a DNF $\bigvee_i \bigwedge_j \psi^i_j$.

Model Checking Algorithm

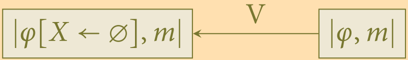
Given: a WMSO sentence φ and $\mathcal{D}$.
Problem: check whether $\mathcal{S}_m(\mathcal{D}) \vDash \varphi$.

- **Atomic sentences** $\top$ and $\bot$: trivial
- **Boolean combinations**: evaluate subformulas
- $\exists X \varphi(X)$ **or** $\exists x \varphi(x)$:
  determine the **winner** of the game $\mathcal{G}^{\exists}(\varphi, i)$ between
  the Verifier and the Falsifier
- $\forall X \varphi(X)$ **or** $\forall x \varphi(x)$:
  check $\neg\exists X \neg\varphi(X)$ by determining the **loser** of $\mathcal{G}^{\exists}(\neg\varphi, i)$

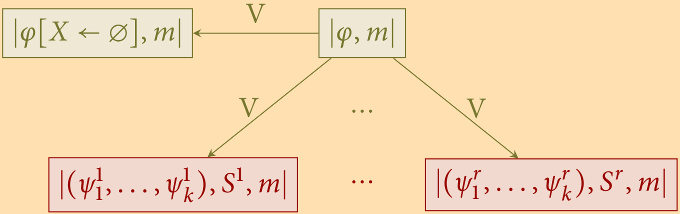
Game $\mathcal{G}^{\exists}(\varphi, m)$ for $\exists X \varphi(X)$

$|\varphi, m|$

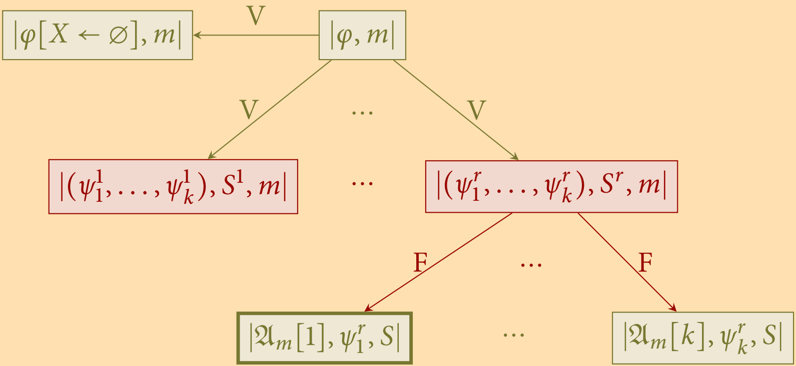
Game $\mathcal{G}^{\exists}(\varphi, m)$ for $\exists X \varphi(X)$



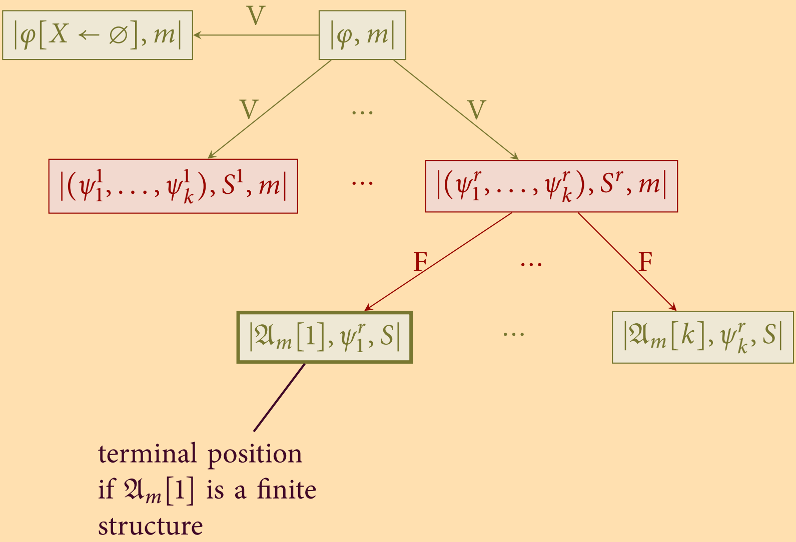
Game $\mathcal{G}^{\exists}(\varphi, m)$ for $\exists X \varphi(X)$



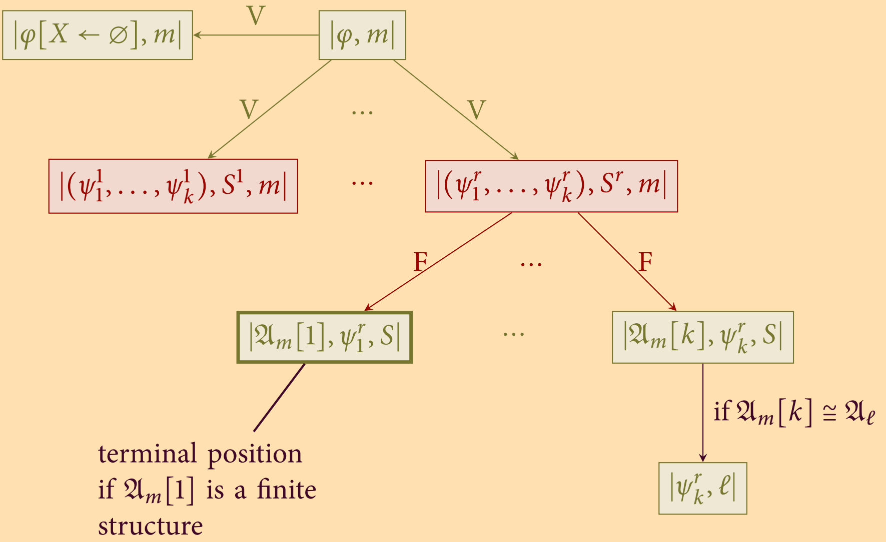
Game $\mathcal{G}^{\exists}(\varphi, m)$ for $\exists X \varphi(X)$



Game $\mathcal{G}^{\exists}(\varphi, m)$ for $\exists X \varphi(X)$



Game $\mathcal{G}^{\exists}(\varphi, m)$ for $\exists X \varphi(X)$



Unbounding Quantifier

$UX\varphi(X)$: "for all $n \in \mathbb{N}$, $\exists X\varphi(X)$ with X finite and $|X| \geq n$"

- introduced by Bojańczyk in 2004
- WMSO+U proved to be decidable on trees only with very restricted quantification patterns
- general decidability results for WMSO+U only for words [B. 2009]

Unbounding Quantifier

$UX\varphi(X)$: "for all $n \in \mathbb{N}$, $\exists X\varphi(X)$ with X finite and $|X| \geq n$"

- introduced by Bojańczyk in 2004
- WMSO+U proved to be decidable on trees only with very restricted quantification patterns
- general decidability results for WMSO+U only for words [B. 2009]

A modification of the winning condition yields a game checking $UX\varphi(X)$.

$\rightsquigarrow$ **Decidability of the WMSO+U theory of inductive structures.**

Implementation

Technical details

- We use MiniSat for **CNF ↔ DNF conversions** (following McMillan)
- Prototype implemented in OCaml (still to be improved)
- Try to **simplify formulas** at each step

Test cases

(1) Formulas of Presburger arithmetic
(representing binary numbers by finite sets in $(\omega, <)$)

(2) artificially constructed Horn formulas of the form

$$\varphi_n := \exists X \forall x_1 \dots \forall x_n$$
$$(x_1 \in X \to x_2 \in X) \wedge \dots \wedge (x_{n-1} \in X \to x_n \in X)$$

Preliminary results

- **Case (1)** slower than Mona; **Case (2)** faster.
- Speed depends on **simplification strategy** and **memoisation**
- **Future work:** better use of dynamic programming

Conclusion

Formula manipulations can simulate automata for WMSO

Conclusion

Formula manipulations can simulate automata for WMSO

Thank You