

Logic and Automata

Lecture 4
Tree Automata, MSO-Translations

Sebastian Maneth
NICTA & UNSW

Logic Summer School - Canberra, December 2006

Outline

1. Automata from strings to trees
2. Context-Free Sets
3. MSO Definable Translations
4. Applications

Recap

MSO = first-order logic (FO) + quantification over sets

Closed MSO formulae ϕ

Graph properties / languages $L(\phi) = \{ \text{graphs } g \mid g \models \phi \}$

$\rightarrow L(\phi) = \Sigma^* ?$
 $L(\phi) \neq \emptyset ?$ } **undecidable**

String properties / languages $L(\phi)$ is a regular language [Büchi1960]

$\rightarrow L(\phi) = \Sigma^* ?$
 $L(\phi) \neq \emptyset ?$ } **decidable**

Recap

MSO = first-order logic (FO) + quantification over sets

Closed MSO formulae ϕ

Graph properties / languages $L(\phi) = \{ \text{graphs } g \mid g \models \phi \}$

$\rightarrow L(\phi) = \Sigma^* ?$
 $L(\phi) \neq \emptyset ?$ } **undecidable**

String properties / languages $L(\phi)$ is a regular language [Büchi1960]

$\rightarrow L(\phi) = \Sigma^* ?$
 $L(\phi) \neq \emptyset ?$ } **decidable, complexity?**

Recap & into the Trees

MSO = first-order logic (FO) + quantification over sets

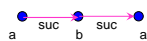
Closed MSO formulae ϕ

Graph properties / languages $L(\phi) = \{ \text{graphs } g \mid g \models \phi \}$

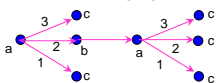
$\rightarrow L(\phi) = \Sigma^* ?$
 $L(\phi) \neq \emptyset ?$ } **undecidable**

String properties / languages $L(\phi)$ is a regular language [Büchi1960]

$\rightarrow L(\phi) = \Sigma^* ?$
 $L(\phi) \neq \emptyset ?$ } **decidable**



Tree properties / languages $L(\phi)$ is a **regular tree language** [Thatcher/Wright1968]



\rightarrow more than one successor

1. Automata: from Strings to Trees

Strings $A = (Q, \Sigma, q_0, F, \delta)$

- Q finite set of states
- Σ input alphabet
- $q_0 \in Q$ initial state
- $F \subseteq Q$ set of final states
- $\delta: Q \times \Sigma \rightarrow P(Q)$ transition function

1. Automata: from Strings to Trees

Strings $A = (Q, \Sigma, q_0, F, \delta)$

- Q finite set of **states**
- Σ input alphabet
- $q_0 \in Q$ initial state
- $F \subseteq Q$ set of final states
- $\delta: Q \times \Sigma \rightarrow P(Q)$ transition function

Trees Σ is a *ranked* alphabet (each symbol has a fixed arity)

For example

$\Sigma = \{d^{(3)}, e^{(2)}, a^{(0)}, b^{(0)}\}$ meaning $\text{arity}(d) = 3$
 $\text{arity}(e) = 2$
 $\text{arity}(a) = \text{arity}(b) = 0$

Written as expression: $d(a, e(a, b), b)$

Notation $T_\Sigma =$ set of all trees over Σ

tree over Σ

1. Automata: from Strings to Trees

Strings $A = (Q, \Sigma, q_0, F, \delta)$

- Q finite set of **states**
- Σ input alphabet
- $q_0 \in Q$ initial state
- $F \subseteq Q$ set of final states
- $\delta: Q \times \Sigma \rightarrow P(Q)$ transition function

Trees Σ is a *ranked* alphabet (each symbol has a fixed arity)

$\delta(q, d) \in P(Q \times \dots \times Q)$ for every $d \in \Sigma$

arity(d)

\rightarrow "(nondeterministic) **top-down tree automaton**"

Example $\Sigma = \{d^{(3)}, e^{(2)}, a^{(0)}, b^{(0)}\}$ $F = \{f\}$, q is initial state

$\delta(q, d) = \{(p, q, r), (p, q', r)\}$
 $\delta(q', e) = \{(p, r)\}$
 $\delta(p, a) = \{f\}$
 $\delta(r, b) = \{f\}$

1. Automata: from Strings to Trees

Strings $A = (Q, \Sigma, q_0, F, \delta)$

- Q finite set of **states**
- Σ input alphabet
- $q_0 \in Q$ initial state
- $F \subseteq Q$ set of final states
- $\delta: Q \times \Sigma \rightarrow P(Q)$ transition function

Trees Σ is a *ranked* alphabet (each symbol has a fixed arity)

$\delta(q, d) \in P(Q \times \dots \times Q)$ for every $d \in \Sigma$

arity(d)

\rightarrow "(nondeterministic) **top-down tree automaton**"

Example $\Sigma = \{d^{(3)}, e^{(2)}, a^{(0)}, b^{(0)}\}$ $F = \{f\}$, q is initial state

$\delta(q, d) = \{(p, q, r), (p, q', r)\}$
 $\delta(q', e) = \{(p, r)\}$
 $\delta(p, a) = \{f\}$
 $\delta(r, b) = \{f\}$

\leftarrow **determinize?** (just delete state q')

1. Automata: from Strings to Trees

Strings $A = (Q, \Sigma, q_0, F, \delta)$

- Q finite set of **states**
- Σ input alphabet
- $q_0 \in Q$ initial state
- $F \subseteq Q$ set of final states
- $\delta: Q \times \Sigma \rightarrow P(Q)$ transition function

Trees Σ is a *ranked* alphabet (each symbol has a fixed arity)

$\delta(q, d) \in P(Q \times \dots \times Q)$ for every $d \in \Sigma$

arity(d)

\rightarrow "(nondeterministic) **top-down tree automaton**"

Example $\Sigma = \{d^{(3)}, e^{(2)}, a^{(0)}, b^{(0)}\}$ $F = \{f\}$, q is initial state

$\delta(q, d) = \{(p, q, r)\}$
 $\delta(q, e) = \{(p, r)\}$
 $\delta(p, a) = \{f\}$
 $\delta(r, b) = \{f\}$

deterministic top-down tree automaton!

1. Automata: from Strings to Trees

Strings $A = (Q, \Sigma, q_0, F, \delta)$

- Q finite set of **states**
- Σ input alphabet
- $q_0 \in Q$ initial state
- $F \subseteq Q$ set of final states
- $\delta: Q \times \Sigma \rightarrow P(Q)$ transition function

Trees Σ is a *ranked* alphabet (each symbol has a fixed arity)

$\delta(q, d) \in P(Q \times \dots \times Q)$ for every $d \in \Sigma$

arity(d)

\rightarrow "(nondeterministic) **top-down tree automaton**"

Example $\Sigma = \{e^{(2)}, a^{(0)}, b^{(0)}\}$ $F = \{f\}$, q is initial state

$\delta(q, e) = \{(p, r), (r, p)\}$ $L(A) = \{e(a, b), e(b, a)\}$
 $\delta(p, a) = \{f\}$
 $\delta(r, b) = \{f\}$

\leftarrow **determinize???**

1. Tree Automata

\rightarrow Deterministic top-down tree automata are stupid!
 Cannot even accept the finite tree language $\{e(a, b), e(b, a)\} := \text{FIN}$

Better bottom-up tree automata!

(on strings, left-to-right automata give the same as right-to-left.)

$\delta_a() \in P(Q)$ for a of arity zero
 $\delta_d(q_1, \dots, q_k) \in P(Q)$ for d of arity k

$A = (Q, \Sigma, F, \delta)$

\uparrow

Example

$\delta_a() = \{p\}$
 $\delta_b() = \{q\}$
 $\delta_p(p, q) = \delta(q, p) = f$

No initial state needed...

Deterministic, and accepts FIN!

1. Tree Automata

→ Deterministic top-down tree automata are stupid.
 Cannot even accept the finite tree language $\{ e(a,b), e(b,a) \} := \text{FIN}$

Better **bottom-up tree automata!**

(on strings, left-to-right automata give the same as right-to-left..)

$\delta_a() \in P(Q)$ for a of arity zero
 $\delta_d(q_1, \dots, q_k) \in P(Q)$ for d of arity k

$A = (Q, \Sigma, F, \delta)$

↑

No initial state needed...

Example

$\delta_a() = \{ p \}$
 $\delta_b() = \{ q \}$
 $\delta_f(p, q) = \delta(q, p) = f$

Deterministic, and accepts FIN!

Subset construction works! Nondet BU-TA = Det. BU-TA (= Nondet. TD-TA)

BOTTOM-UP FINITE-STATE Tree AUTOMATA

→ can be *determinized* (but, exponential blow-up ☹)
 (and, \exists unique *minimal deterministic* automaton for every automaton)
 → allow constant memory scanning (→ stream-processing ☺)
 → accept the **regular Tree languages**

Regular Tree Languages have nice properties

- Characterized by**
- Finite-State **Tree** Automata
 - Regular **Tree** Expressions
 - Regular **Tree** Grammars
 - MSO sentences [Thatcher/Wright1968]
 - ...
- Closed under**
- Intersection
 - Union
 - Complement, ...

Decidable emptiness/equivalence/
 inclusion etc

BOTTOM-UP FINITE-STATE Tree AUTOMATA

→ can be *determinized* (but, exponential blow-up ☹)
 (and, \exists unique *minimal deterministic* automaton for every automaton)
 → allow constant memory scanning (→ stream-processing ☺)
 → accept the **regular Tree languages**

Regular Tree Languages have nice properties

- Characterized by**
- Finite-State **Tree** Automata
 - Regular **Tree** Expressions
 - Regular **Tree** Grammars
 - MSO sentences [Thatcher/Wright1968]
 - ...
- Closed under**
- Intersection
 - Union
 - Complement, ...

Decidable emptiness/equivalence/
 inclusion etc

Generalize *derivation trees*
 of **context-free grammars!**

BOTTOM-UP FINITE-STATE Tree AUTOMATA

→ can be *determinized* (but, exponential blow-up ☹)
 (and, \exists unique *minimal deterministic* automaton for every automaton)
 → ~~allow constant memory scanning (→ stream-processing ☺)~~
 → accept the **regular Tree languages**

Regular Tree Languages have nice properties

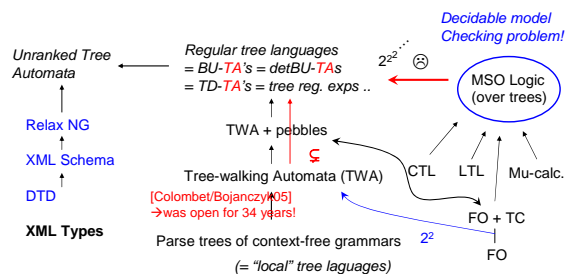
- Characterized by**
- Finite-State **Tree** Automata
 - Regular **Tree** Expressions
 - Regular **Tree** Grammars
 - MSO sentences [Thatcher/Wright1968]
 - ...
- Closed under**
- Intersection
 - Union
 - Complement, ...

Decidable emptiness/equivalence/
 inclusion etc

Generalize *derivation trees*
 of **context-free grammars!**

Tree Automata

→ Have MANY applications!



Decidability

MSO = first-order logic (FO) + quantification over **sets**

FO theory of finite graphs: $\{ \phi \in \text{FO} \mid g \models \phi \text{ for every finite graph } g \}$
 is undecidable.

MSO: If **C** contains infinitely many square grids,
 then MSO theory of **C** is undecidable.

→ **Restrict class C**, so that MSO theory becomes decidable!

(2) **C = TREES**

Theorem (Thatcher/Wright) *Set of trees is MSO definable if and only if it is accepted by a finite-state automaton.*

Corollary *MSO theory of trees is decidable.*

Decidability

→ Restrict class C , so that MSO theory becomes decidable!

(2) $C = \text{TREEs}$

Theorem (Thatcher/Wright) *Set of trees is MSO definable if and only if it is accepted by a finite-state automaton.*

Corollary MSO theory of trees is decidable.

Can we push this further?

Which class C of graphs, has a decidable MSO theory?

Decidability

→ Restrict class C , so that MSO theory becomes decidable!

(2) $C = \text{TREEs}$

Theorem (Thatcher/Wright) *Set of trees is MSO definable if and only if it is accepted by a finite-state automaton.*

Corollary MSO theory of trees is decidable.

Can we push this further?

Which class C of graphs, has a decidable MSO theory?

YES

Take $C =$ class of context-free graph languages /
graphs of bounded tree width /
graphs of bounded clique width

Decidability

Habel/Kreowski/Vogler, Metatheorems for Decision Problems on Hyperedge Replacement Graph Languages. *Acta Informatica*, 1989.

Courcelle, The monadic second-order logic of graphs. *Information and Computation*, 1990.

Arnborg/Lagergren/Seese, Easy Problems for Tree-Decomposable Graphs. *Journal of Algorithms*, 1991.

Habel/Kreowski/Lauteman, A comparison of compatible, finite, and inductive graph properties. *Theoret. Comput. Sci.*, 1993

Lengauer/Wanke, Efficient decision procedures for graph properties on context-free graph languages. *Journal of the ACM*, 1993.

2. Context-Free Sets

→ Universal Algebra

Σ finite set of operations on some domain D

Interpretation $\text{val}_\Sigma: T_\Sigma \rightarrow D$



$w \in D$

If all operations of arity ≥ 1 mean string concatenation then

$\text{CF}(\text{set of strings}) = \text{val}_\Sigma(\text{REGT}_\Sigma)$ [Mezei/Wright68]

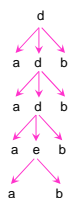
2. Context-Free Sets of Strings

$\Sigma = \{d^{(0)}, e^{(2)}, a^{(0)}, b^{(0)}\}$ $F = \{f\}$, q is initial state

$\delta(q, d) = \{(p, q, r)\}$

$\delta(q, e) = \{(p, r)\}$

$\delta(p, a) = \delta(r, b) = \{f\}$



Interpret

Internal nodes as string concatenation

$l(d)(w_1, w_2, w_3) = w_1 w_2 w_3$

$l(e)(w_1, w_2) = w_1 w_2$

Leaves as constants

Gives $a a a a b b b b$

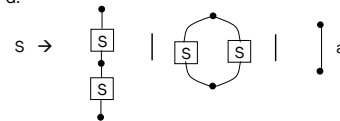
CF languages = "yields / frontiers" of the regular tree languages!
(string of leaf-labels)

2. Context-Free Sets of Graphs

Σ finite set of graph operations

Interpretation $\text{val}_\Sigma: T_\Sigma \rightarrow D$

G:



p

(arity 2)

q

(arity 2)

r

(arity 0)

$R: Z \rightarrow p(Z, Z) \mid q(Z, Z) \mid r$

$L(G) = \text{val}_\Sigma(L(R))$

Thus, the set of all series-parallel graphs is context-free!

Language Theory

	STRINGS	TREES	GRAPHS
Regular	finite automata regular expression regular grammars ...	→ natural → extensions → "	? ? ?
	MSO logic	→ "	very naturall!
Context-Free	EBNF (grammars) push-down automata ...	→ natural → extension	→ rather natural NR HR

2. Context-Free Sets of Graphs

(1) HR (Hyperedge Replacement) Graph Grammars
 (2) NR (Node Replacement) Graph Grammars

HR is basically = "graphs of bounded tree-width"

→ Note Set of ALL graphs is NOT context-free

$HR \cap MSO_{HR} \subseteq HR$
 $NR \cap MSO_{NR} \subseteq NR$

This is just like $CF \cap REG \subseteq CF$ ("triple construction")

But we will prove this in a more elegant way!

Because $HR/NR = MSOT_{HR/NR}(REGT)$
 "MSO Definable Translations"

Decidability

f : tree-to-graph function
 e.g., translates parse trees into control-flow graphs.

Question: given an MSO graph property p , can we verify $p(g)$ on the tree t ?

Depends on !

→ If f is an MSO definable translation, then answer is "YES"!

Application

MSOGT⁻¹ preserve MSO graph languages

Parse tree → **flow graph**

If $trans$ is an MSOT, then all MSO properties on the flow graph can effectively be transformed into MSO properties on the parse tree!

3. MSO Definable Translations

For which large class of tree-to-graph functions can we effectively translate MSO graph properties into MSO tree properties that work on the corresponding input trees?

→ MSO graph transductions (MSOT) [Courcelle, Engelfriet]

FST on strings	MSOT on graphs
closed under	closed under
• composition	• composition
• inverse	
Preserve	Preserve
• regular languages	• cf. graph languages
• cf. languages	• MSOT ⁻¹ preserve MSO graph lang's

CF graph languages = MSOT(T_Σ)

3. MSO Definable Translations

MSO graph translation $f : GR(\Sigma_1, \Gamma_1) \rightarrow GR(\Sigma_2, \Gamma_2)$

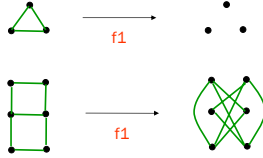
dom: domain formula $L(dom) = dom(f)$
 $N_x(x)$: node formulas for every $\sigma \in \Sigma_2$
 $E_\gamma(x, y)$: edge formulas for every $\gamma \in \Gamma_2$

→ Define output graph, in terms of input graph.
 Interpretation of one logical structure in another. (FMT: "reductions")

3. MSO Definable Translations

Example: edge complement f_1

dom \equiv true
 $N_\sigma(x) \equiv \text{lab}_\sigma(x)$ for all σ
 $E(x, y) \equiv \neg \text{edg}(x, y)$



3. MSO Definable Translations

MSO graph translation $f : \text{GR}(\Sigma_1, \Gamma_1) \rightarrow \text{GR}(\Sigma_2, \Gamma_2)$

dom: domain formula $L(\text{dom}) = \text{dom}(f)$
 $N_\sigma(x)$: node formulas for every $\sigma \in \Sigma_2$
 $E_\gamma(x, y)$: edge formulas for every $\gamma \in \Gamma_2$

Input graph $g = (V, E)$
 output graph $h = f(g) = (U, F)$

$U = \{ u \in V \mid \text{there is a unique } \sigma \in \Sigma_2 \text{ such that } (g, u) \models N_\sigma(x) \}$
 $E = \{ (u, \gamma, v) \mid \exists! \gamma \in \Gamma_2: (g, u, v) \models E_\gamma(x, y) \}$
 $\forall u \in U: \text{lab-}\sigma(u) \text{ iff } (g, u) \models N_\sigma(x)$

3. MSO Definable Translations

MSO graph translation $f : \text{GR}(\Sigma_1, \Gamma_1) \rightarrow \text{GR}(\Sigma_2, \Gamma_2)$

dom: domain formula $L(\text{dom}) = \text{dom}(f)$
 $N_\sigma(x)$: node formulas for every $\sigma \in \Sigma_2$
 $E_\gamma(x, y)$: edge formulas for every $\gamma \in \Gamma_2$

Input graph $g = (V, E)$ $\rightarrow \text{size}(f(g)) = |U|$
 output graph $h = f(g) = (U, F)$ $\text{is} \leq \text{size}(g) = |V|$

$U = \{ u \in V \mid \text{there is a unique } \sigma \in \Sigma_2 \text{ such that } (g, u) \models N_\sigma(x) \}$
 $E = \{ (u, \gamma, v) \mid \exists! \gamma \in \Gamma_2: (g, u, v) \models E_\gamma(x, y) \}$
 $\forall u \in U: \text{lab-}\sigma(u) \text{ iff } (g, u) \models N_\sigma(x)$

3. MSO Definable Translations

MSO graph translation $f : \text{GR}(\Sigma_1, \Gamma_1) \rightarrow \text{GR}(\Sigma_2, \Gamma_2)$

dom: domain formula $L(\text{dom}) = \text{dom}(f)$
 $N_\sigma(x)$: node formulas for every $\sigma \in \Sigma_2$
 $E_\gamma(x, y)$: edge formulas for every $\gamma \in \Gamma_2$

Input graph $g = (V, E)$ $\rightarrow \text{size}(f(g)) = |U|$
 output graph $h = f(g) = (U, F)$ $\text{is} \leq \text{size}(g) = |V|$

$U = \{ u \in V \mid \text{there is a unique } \sigma \in \Sigma_2 \text{ such that } (g, u) \models N_\sigma(x) \}$
 $E = \{ (u, \gamma, v) \mid \exists! \gamma \in \Gamma_2: (g, u, v) \models E_\gamma(x, y) \}$
 $\forall u \in U: \text{lab-}\sigma(u) \text{ iff } (g, u) \models N_\sigma(x)$

C : finite "copy" set $(\text{dom}, C, \{N_{\sigma,c}\}_{\sigma \in \Sigma_2, c \in C}, \{E_{\gamma,c,d}\}_{\gamma \in \Gamma_2, c,d \in C})$

$U = \{ (u, c) \in V \times C \mid \exists! \sigma \in \Sigma_2: (g, u) \models N_{\sigma,c}(x) \}$

$E = \{ ((u, c), \gamma, (v, d)) \mid \exists! \gamma \in \Gamma_2: (g, u, v) \models E_{\gamma,c,d}(x, y) \}$

$\forall u \in U: \text{lab-}\sigma(u) \text{ iff } (g, u) \models N_\sigma(x)$

3. MSO Definable Translations

MSO graph translation $f : \text{GR}(\Sigma_1, \Gamma_1) \rightarrow \text{GR}(\Sigma_2, \Gamma_2)$

dom: domain formula $L(\text{dom}) = \text{dom}(f)$
 $N_\sigma(x)$: node formulas for every $\sigma \in \Sigma_2$
 $E_\gamma(x, y)$: edge formulas for every $\gamma \in \Gamma_2$

Input graph $g = (V, E)$ $\rightarrow \text{size}(f(g)) = |U|$
 output graph $h = f(g) = (U, F)$ $\text{is} \leq \text{size}(g) = |V|$

$U = \{ u \in V \mid \text{there is a unique } \sigma \in \Sigma_2 \text{ such that } (g, u) \models N_\sigma(x) \}$
 $E = \{ (u, \gamma, v) \mid \exists! \gamma \in \Gamma_2: (g, u, v) \models E_\gamma(x, y) \}$
 $\forall u \in U: \text{lab-}\sigma(u) \text{ iff } (g, u) \models N_\sigma(x)$

C : finite "copy" set $(\text{dom}, C, \{N_{\sigma,c}\}_{\sigma \in \Sigma_2, c \in C}, \{E_{\gamma,c,d}\}_{\gamma \in \Gamma_2, c,d \in C})$

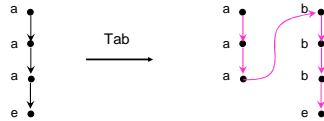
$U = \{ (u, c) \in V \times C \mid \exists! \sigma \in \Sigma_2: (g, u) \models N_{\sigma,c}(x) \}$

$E = \{ ((u, c), \gamma, (v, d)) \mid \exists! \gamma \in \Gamma_2: (g, u, v) \models E_{\gamma,c,d}(x, y) \}$

$\forall u \in U: \text{lab-}\sigma(u) \text{ iff } (g, u) \models N_\sigma(x) \rightarrow \text{size}(f(g)) \leq |C| \cdot \text{size}(g)$

3. MSO Definable Translations

MSO string-to-string transducer Tab



$C = \{ A, B \}$ $\text{dom} \equiv \text{string} \wedge (\forall x)(\text{lab-e}(x) \leftrightarrow \text{last}(x))$

$N_{a,A}(x) \equiv \text{lab-a}(x)$

$N_{b,B}(x) \equiv \text{lab-a}(x)$

$N_{e,A}(x) \equiv \text{false}$

$N_{e,B}(x) \equiv \text{lab-e}(x)$

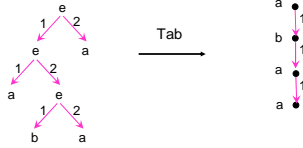
< all others are set to false... >

$E_{A,A}(x, y) \equiv E_{B,B}(x, y) \equiv \text{edge}(x, y)$

$E_{A,B}(x, y) \equiv (\exists z)(\text{edge}(x, z) \wedge \text{lab-e}(z)) \wedge \neg(\exists z) \text{edge}(z, y)$

3. MSO Definable Translations

MSO *tree-to-string* transducer Tyield



dom \equiv true

$N_1(x) \equiv N_2(x) \equiv \text{lab-}a(x)$

$E_1(x, y) \equiv (\exists u)(\exists v)(\exists w)$
 $\text{edge-}2^*(v, x) \wedge \text{edge-}1(u, v)$
 $\wedge \text{edge-}2(u, w) \wedge \text{edge-}1^*(w, y)$
 $\wedge \text{leaf}(x) \wedge \text{leaf}(y)$

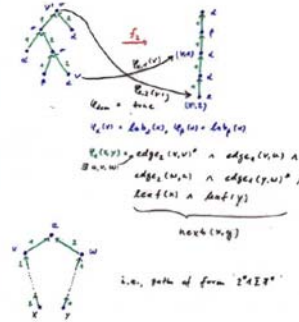
< all others are set to false...! >

MSO Definable Translations

Example

Translate a tree into its yield/frontier

Example: yield of a tree, f_2
 trees over the ranked alphabet
 $\Delta = \{a^{(1)}, a^{(2)}, a^{(3)}\}$



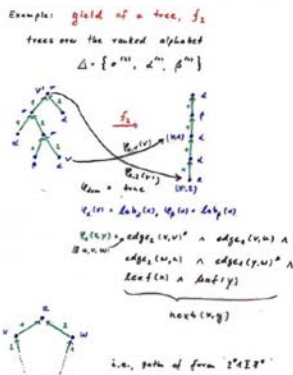
MSO Definable Translations

Example

Translate a tree into its yield/frontier

Corollary

Class of string languages generated by MSOT's includes the CF languages



MSO Definable Translations

Example

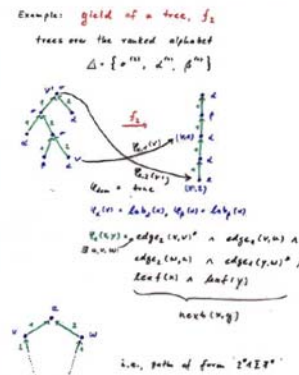
Translate a tree into its yield/frontier

Corollary

Class of string languages generated by MSOT's includes the CF languages

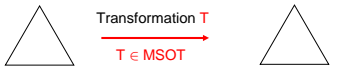
→ Strict superset Generate

{ $a^n b^n c^n d^n \mid n \geq 0$ }



3. MSO Definable Translations

Application Type Checking XML transformations



Input XML tree of type INP

Input XML tree of type OUT

XML types are regular tree languages → MSO-definable
 By sentence ϕ_{INP} and ϕ_{OUT}

(static) Type Checking Problem $T(\text{doctr}) \in \text{OUT}$ for all $\text{doctr} \in \text{INP}$?

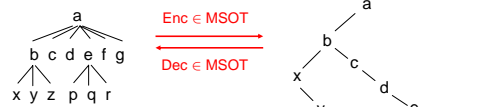
→ Decidable!

$T^{-1}(\neg\phi_{\text{OUT}})$ is MSO definable (by some ψ).

Check if $(\psi \wedge \phi_{\text{INP}})$ is satisfiable!

3. MSO Definable Translations

Application Translate MSO properties on unranked trees to ranked trees



"MSO definable coding"

Shows that unranked regular tree languages are binary tree decodings of ranked (binary) tree Languages.

Very useful! No need to "generalize" great part of tree language theory from ranked to unranked trees.

MSO Definable Tree Translations (MSOTT)

Have many applications!

- Equal to prim. rec. tree functions with params (=FPs on trees, using pattern matching = macro tree transducers) [Engelfriet/Maneth03]
- They characterize the (structural) **XML queries** (XQuery / XSLT) which are linear time computable. [Engelfriet/Maneth03a+b, Maneth03]
- They even have decidable equivalence [Engelfriet/Maneth06]

FACTS

MSOTT

- Closed under composition
- inverses preserve REGT

MSOTT(REGT) ← "most beautiful class of tree languages! ☺"
= MSOTT(REGT)

- Closed under MSOTT
- path languages(MSOTT(REGT))
= string languages(MSOTT(REGT))
= MSOTS(REGT)
- are **Parikh**