Trees Abound
Part I: Tree Automata

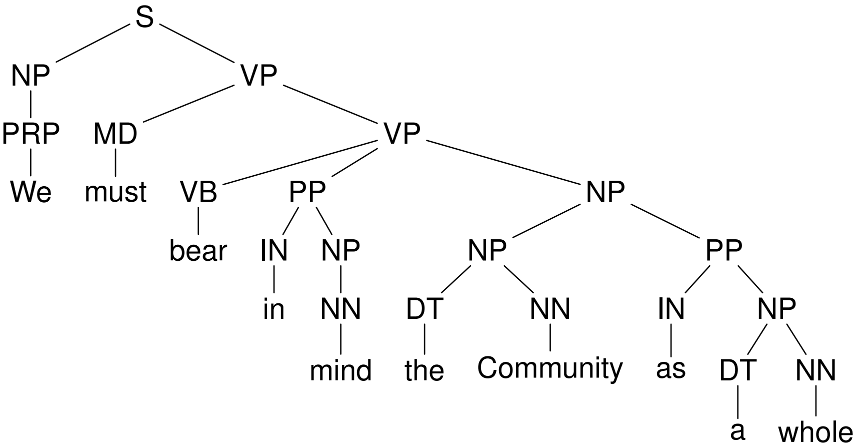
Andreas Maletti

Institute for Natural Language Processing
Universität Stuttgart, Germany

maletti@ims.uni-stuttgart.de

Paris — September 26, 2012

Trees?


S
NP
VP
PRP
MD
VP
We
must
VB
PP
NP
bear
IN
NP
NP
PP
in
NN
DT
NN
IN
NP
mind
the
Community
as
DT
NN
a
whole

Tree Language

How to represent a set of trees?

- enumerate them
- enumerate them cleverly (e.g., add sharing)
- parse forest of a CFG

Tree Language

How to represent a set of trees?

- ~~enumerate them~~
- enumerate them cleverly (e.g., add sharing)
- parse forest of a CFG

Tree Language

How to represent a set of trees?

- ~~enumerate them~~
- enumerate them cleverly (e.g., add sharing)
- parse forest of a CFG

Parse Forest of a CFG

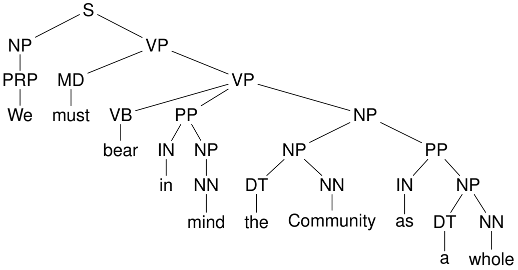
Example

$$
\begin{aligned}
\mathsf{S} &\to \mathsf{NP}\ \mathsf{VP} & \qquad \mathsf{VP} &\to \mathsf{MD}\ \mathsf{VP} \\
\mathsf{NP} &\to \mathsf{NP}\ \mathsf{PP} & \qquad \mathsf{VP} &\to \mathsf{VB}\ \mathsf{PP}\ \mathsf{NP} \\
\mathsf{MD} &\to \mathsf{must} & & \ldots
\end{aligned}
$$

Parse Forest of a CFG

Example

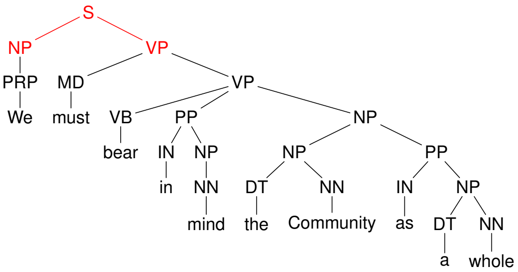
$$
\begin{aligned}
S &\to NP\ VP & \qquad VP &\to MD\ VP \\
NP &\to NP\ PP & \qquad VP &\to VB\ PP\ NP \\
MD &\to \text{must} & \qquad &\ldots
\end{aligned}
$$



Parse Forest of a CFG

Example

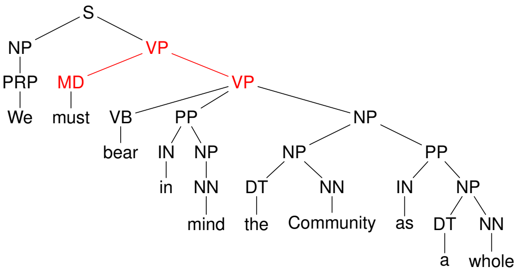
$$S \to NP\ VP \qquad VP \to MD\ VP$$
$$NP \to NP\ PP \qquad VP \to VB\ PP\ NP$$
$$MD \to must \qquad \ldots$$



Parse Forest of a CFG

Example

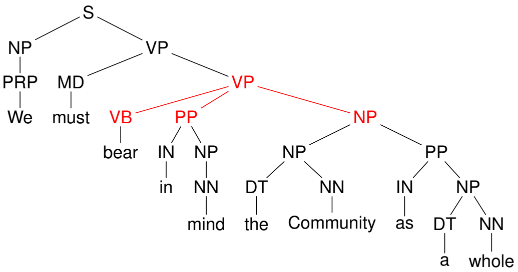
$$S \to NP\ VP \qquad VP \to MD\ VP$$
$$NP \to NP\ PP \qquad VP \to VB\ PP\ NP$$
$$MD \to must \qquad \ldots$$



Parse Forest of a CFG

Example

$$S \to NP\ VP \qquad VP \to MD\ VP$$
$$NP \to NP\ PP \qquad VP \to VB\ PP\ NP$$
$$MD \to must \qquad \dots$$



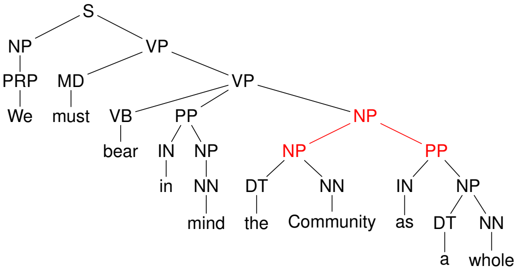
Parse Forest of a CFG

Example

$$\mathrm{S} \to \mathrm{NP}\ \mathrm{VP} \qquad \mathrm{VP} \to \mathrm{MD}\ \mathrm{VP}$$

$$\mathrm{NP} \to \mathrm{NP}\ \mathrm{PP} \qquad \mathrm{VP} \to \mathrm{VB}\ \mathrm{PP}\ \mathrm{NP}$$

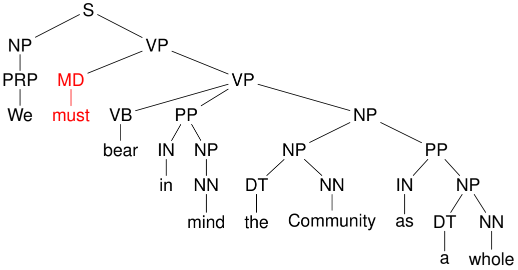
$$\mathrm{MD} \to \mathrm{must} \qquad \ldots$$



Parse Forest of a CFG

Example

$$
\begin{aligned}
S &\to NP\ VP & \qquad VP &\to MD\ VP \\
NP &\to NP\ PP & \qquad VP &\to VB\ PP\ NP \\
\color{red}{MD} &\color{red}{\to must} & & \dots
\end{aligned}
$$



Local Tree Grammar

Definition (Gécseg, Steinby 1984)

A local tree grammar G is a finite set of CFG productions (together with a start nonterminal S)

Definition (Generated tree language)

$L(G)$ contains exactly the trees in which

- the root is labeled S
- "label $\rightarrow$ child labels" is a production of G for each internal node

Local Tree Grammar

Definition (Gécseg, Steinby 1984)

A local tree grammar G is a finite set of CFG productions (together with a start nonterminal S)

Definition (Generated tree language)

$L(G)$ contains exactly the trees in which

- the root is labeled S
- "label $\rightarrow$ child labels" is a production of G for each internal node

Local Tree Grammar

Theorem

Local tree grammars recognize exactly the parse forests of CFG

Properties

- ✓ simple
- ✓ no ambiguity (unique explanation for each recognized tree)
- ✗ not closed under BOOLEAN operations (union/intersection/complement: ✗/✓/✗)
- ✗ not closed under (non-injective) relabelings
- ✗ …

Local Tree Grammar

Theorem

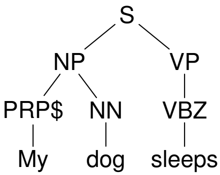
Local tree grammars recognize exactly the parse forests of CFG

Properties

- ✓ simple
- ✓ no ambiguity (unique explanation for each recognized tree)
- ✗ not closed under BOOLEAN operations (union/intersection/complement: ✗/✓/✗)
- ✗ not closed under (non-injective) relabelings
- ✗ …

Local Tree Grammar

No ambiguity



is in $L(G)$ if and only if all the productions in it are in G

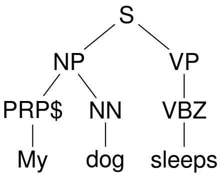
Local Tree Grammar

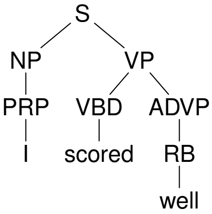
Theorem

Local tree languages are not closed under union

Proof.

The following single-element tree languages are local:





But their union is not local as it must also recognize:

My dog scored well **I sleeps* □

Tree Language

How to represent a set of trees?

- ~~enumerate them~~
- enumerate them cleverly (e.g., add sharing)
- parse forest of a CFG

Tree Language

How to represent a set of trees?

- ~~enumerate them~~
- enumerate them cleverly (e.g., add sharing)
- ~~parse forest of a CFG~~

Tree Language

How to represent a set of trees?

- ~~enumerate them~~
- enumerate them cleverly (e.g., add sharing)
- ~~parse forest of a CFG~~
- tree substitution grammar

Local Tree Grammar

Generalization

- CFG production $\mathrm{L} \rightarrow \mathrm{R}_1\ \mathrm{R}_2\ \mathrm{R}_3$ represented by tree

$$\begin{array}{ccc} & \mathrm{L} & \\ \diagup & | & \diagdown \\ \mathrm{R}_1 & \mathrm{R}_2 & \mathrm{R}_3 \end{array}$$

- “Glue” fragments together to obtain larger trees:

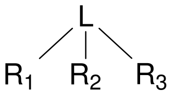
$$\mathrm{S}$$

- But why only small tree fragments?

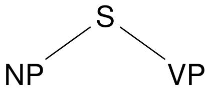
Local Tree Grammar

Generalization

- CFG production $\mathrm{L} \to \mathrm{R}_1\ \mathrm{R}_2\ \mathrm{R}_3$ represented by tree



- "Glue" fragments together to obtain larger trees:

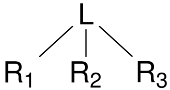


- But why only small tree fragments?

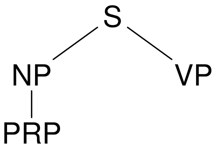
Local Tree Grammar

Generalization

- CFG production $L \to R_1\ R_2\ R_3$ represented by tree



- "Glue" fragments together to obtain larger trees:

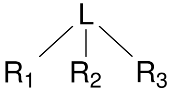


- But why only small tree fragments?

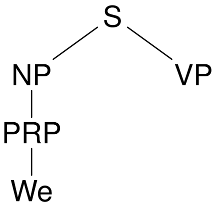
Local Tree Grammar

Generalization

- CFG production $L \to R_1\ R_2\ R_3$ represented by tree



- “Glue” fragments together to obtain larger trees:

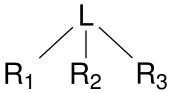


- But why only small tree fragments?

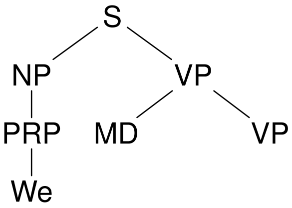
Local Tree Grammar

Generalization

- CFG production $L \to R_1\ R_2\ R_3$ represented by tree



- "Glue" fragments together to obtain larger trees:

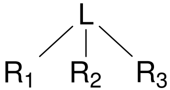


- But why only small tree fragments?

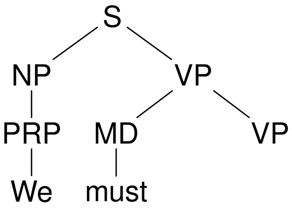
Local Tree Grammar

Generalization

- CFG production $L \to R_1\ R_2\ R_3$ represented by tree



- "Glue" fragments together to obtain larger trees:

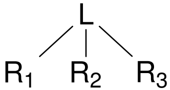


- But why only small tree fragments?

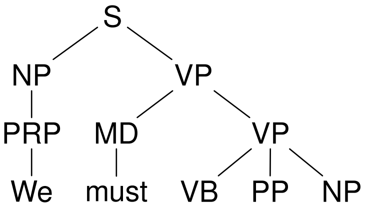
Local Tree Grammar

Generalization

- CFG production $L \to R_1 \ R_2 \ R_3$ represented by tree



- "Glue" fragments together to obtain larger trees:

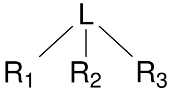


- But why only small tree fragments?

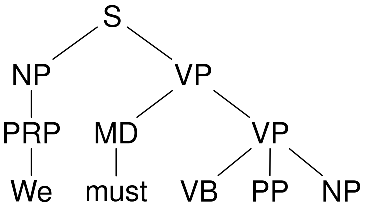
Local Tree Grammar

Generalization

- CFG production $L \to R_1\ R_2\ R_3$ represented by tree



- "Glue" fragments together to obtain larger trees:



- But why only small tree fragments?

Tree Substitution Grammar

Definition

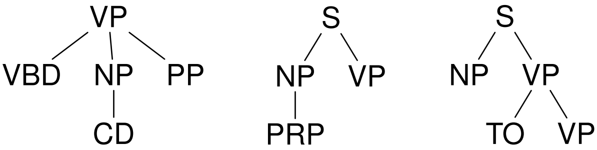
A tree substitution grammar is a finite set of tree fragments (together with a start nonterminal S)

Tree Substitution Grammar

Definition

A tree substitution grammar is a finite set of tree fragments (together with a start nonterminal S)

Example (Typical fragments [Post, ACL 2011])



Tree Substitution Grammar

Theorem

local tree languages $\subsetneq$ *tree substitution languages*

Proof.

Can trivially express all finite tree languages □

Remarks

- can express many finite-distance dependencies
- *extended domain of locality*

Tree Substitution Grammar

Properties

- ✓ simple
- ✓ more expressive than local tree grammars
- ✗ ambiguity (several explanations for a recognized tree)
- ✗ not closed under BOOLEAN operations
  (union/intersection/complement: ✗/✗/✗)
- ✗ not closed under (non-injective) relabelings
- ✗ …

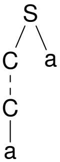
Tree Substitution Grammar

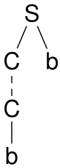
Theorem

Tree substitution languages are not closed under union

Proof.

Counterexample must be infinite $\rightsquigarrow$ artificial example





$L_1 = \{\mathrm{S}(\mathrm{C}^n(a), a) \mid n \in \mathbb{N}\}$ $\qquad$ $L_2 = \{\mathrm{S}(\mathrm{C}^n(b), b) \mid n \in \mathbb{N}\}$

Their union is not a tree substitution language $\square$

Tree Substitution Grammar

Theorem

Tree substitution languages are not closed under intersection

Proof.

Ideas? □

Tree Substitution Grammar

Experiment [Post, Gildea, ACL 2009]

grammar	size	Prec.	Recall	F1
CFG	46k	75.37	70.05	72.61
“spinal” TSG	190k	80.30	78.10	79.18
“minimal subset” TSG	2,560k	76.40	78.29	77.33

(on WSJ Sect. 23)

Tree Language

How to represent a set of trees?

- ~~enumerate them~~
- enumerate them cleverly (e.g., add sharing)
- ~~parse forest of a CFG~~
- tree substitution grammar

Tree Language

How to represent a set of trees?

- ~~enumerate them~~
- enumerate them cleverly (e.g., add sharing)
- ~~parse forest of a CFG~~
- tree substitution grammar
- tree substitution grammar with latent variables

Tree Substitution Grammar with Latent Variables

Definition (Shindo et al., ACL 2012 best paper)

A tree substitution grammar with latent variables is
a tree substitution grammar together with a functional relabeling

Remark

Typically symbols that are relabeled to X are written as X-*n*

Tree Substitution Grammar with Latent Variables

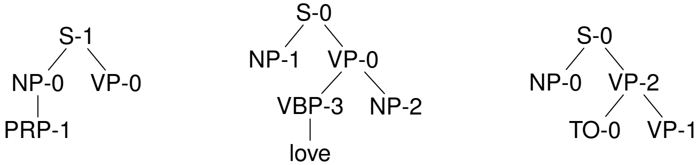
Definition (Shindo et al., ACL 2012 best paper)

A tree substitution grammar with latent variables is
a tree substitution grammar together with a functional relabeling

Remark

Typically symbols that are relabeled to X are written as X-n

Example (Typical fragments)



Tree Substitution Grammar with Latent Variables

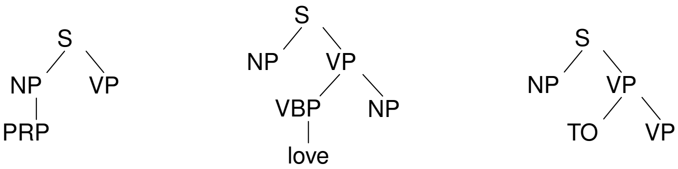
Definition (SHINDO et al., ACL 2012 best paper)

A tree substitution grammar with latent variables is
a tree substitution grammar together with a functional relabeling

Remark

Typically symbols that are relabeled to X are written as X-*n*

Example (Typical fragments)



Tree Substitution Grammar with Latent Variables

Experiment [Shindo et al., ACL 2012 best paper]

grammar	F1 score $\|w\| \leq 40$	F1 score full
TSG [Post, Gildea, 2009]	82.6	
TSG [Cohn et al., 2010]	85.4	84.7
CFGlv [Collins, 1999]	88.6	88.2
CFGlv [Petrov, Klein, 2007]	90.6	90.1
CFGlv [Petrov, 2010]		91.8
TSGlv (single)	91.6	91.1
TSGlv (multiple)	92.9	92.4
Discriminative Parsers		
Carreras et al., 2008		91.1
Charniak, Johnson, 2005	92.0	91.4
Huang, 2008	92.3	91.7

Tree Language

How to represent a set of trees?

- ~~enumerate them~~
- enumerate them cleverly (e.g., add sharing)
- ~~parse forest of a CFG~~
- tree substitution grammar
- tree substitution grammar with latent variables

Tree Language

How to represent a set of trees?

- ~~enumerate them~~
- enumerate them cleverly (e.g., add sharing)
- ~~parse forest of a CFG~~
- tree substitution grammar
- tree substitution grammar with latent variables
- parse forest of a CFG with latent variables
- ...

Tree Language

How to represent a set of trees?

- ~~enumerate them~~
- enumerate them cleverly (e.g., add sharing)
- ~~parse forest of a CFG~~
- tree substitution grammar
- tree substitution grammar with latent variables
- parse forest of a CFG with latent variables
- ...

Let us look at a really old model

Overview

Regular Tree Grammar

Definition (Brainerd, 1969)

A regular tree grammar is a tuple $G = (Q, \Sigma, I, P)$ with

- alphabet of nonterminals Q
- alphabet of terminals Σ
- initial nonterminals $I \subseteq Q$
- finite set of productions $P \subseteq Q \times T_\Sigma(Q)$

Remark

Instead of (q, r) we write $q \to r$

Regular Tree Grammar

Example

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$
- $\Sigma = \{\mathrm{VP}, \mathrm{NP}, \mathrm{S}, \dots\}$
- $I = \{q_0\}$
- and the following productions:

$$q_4 \to \mathrm{VP}(q_5, \mathrm{NP}(q_2), q_3) \qquad q_0 \to \mathrm{S}(\mathrm{NP}(q_1), q_4) \qquad q_0 \to \mathrm{S}(q_6, \mathrm{VP}(q_2, q_4))$$

Regular Tree Grammar

Definition (Derivation Semantics)

Sentential forms: $t, u \in T_\Sigma(Q)$

$$\boxed{t \Rightarrow_G u}$$

if there exist position $w \in \mathrm{pos}(t)$ and production $q \to r \in P$

- $t = t[q]_w$
- $u = t[r]_w$

Definition (Recognized tree language)

$$L(G) = \{t \in T_\Sigma \mid \exists q \in I \colon q \Rightarrow_G^* t\}$$

Regular Tree Grammar

Definition (Derivation Semantics)

Sentential forms: $t, u \in T_\Sigma(Q)$

$$\boxed{t \Rightarrow_G u}$$

if there exist position $w \in \mathrm{pos}(t)$ and production $q \to r \in P$

- $t = t[q]_w$
- $u = t[r]_w$

Definition (Recognized tree language)

$$L(G) = \{t \in T_\Sigma \mid \exists q \in I \colon q \Rightarrow_G^* t\}$$

Regular Tree Grammar

Example (Productions)

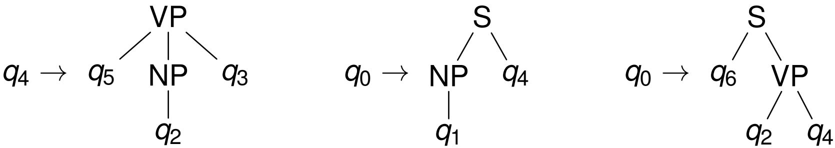
$$q_4 \to \mathrm{VP}(q_5, \mathrm{NP}(q_2), q_3) \qquad q_0 \to \mathrm{S}(\mathrm{NP}(q_1), q_4) \qquad q_0 \to \mathrm{S}(q_6, \mathrm{VP}(q_2, q_4))$$

Example (Derivation)

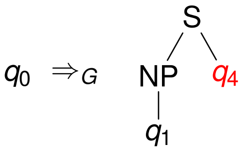
$$q_0$$

Regular Tree Grammar

Example (Productions)

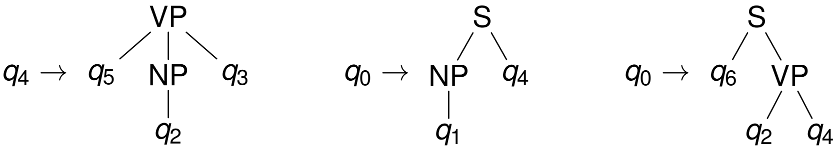


Example (Derivation)

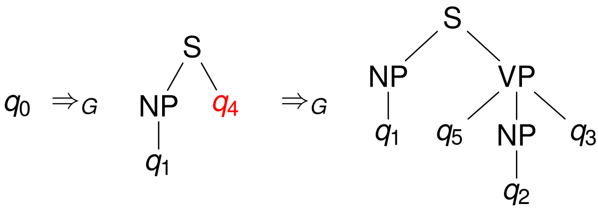


Regular Tree Grammar

Example (Productions)



Example (Derivation)



Regular Tree Grammar

Theorem

tree substitution languages $\subsetneq$ *regular tree languages*

Proof.

We can express the union counterexample easily $\square$

Remarks

- can organize finite information transport
  (even over unbounded distance)

Regular Tree Grammar

Properties

- ✓ simple
- ✓ more expressive than tree substitution grammars
- ✗ ambiguity (several explanations for a recognized tree)
- ✓ closed under all BOOLEAN operations
  (union/intersection/complement: ✓/✓/✓)
- ✓ closed under (non-injective) relabelings
- ✓ …

Regular Tree Grammar

Definition (Brainerd, 1969)

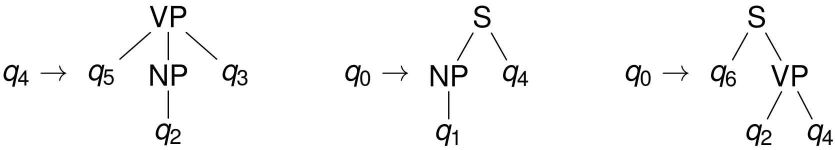
G is in normal form if $r = \sigma(q_1, \ldots, q_k)$ with $\sigma \in \Sigma$ and $q_1, \ldots, q_k \in Q$ for all $q \to r \in P$

Regular Tree Grammar

Definition (Brainerd, 1969)

G is in normal form if $r = \sigma(q_1, \dots, q_k)$ with $\sigma \in \Sigma$ and $q_1, \dots, q_k \in Q$ for all $q \to r \in P$

Example (Productions)



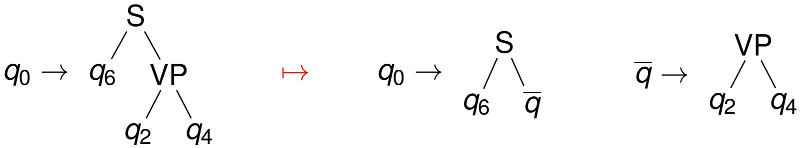
Regular Tree Grammar

Theorem (Brainerd, 1969)

Any G is equivalent to a regular tree grammar in normal form

Proof.

Simply cut large rules introducing new states

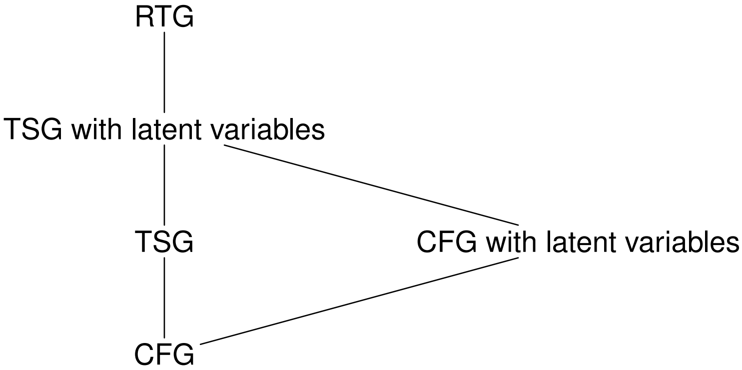


$\square$

Regular Tree Grammar

Theorem (Folk, Lore, 1972)

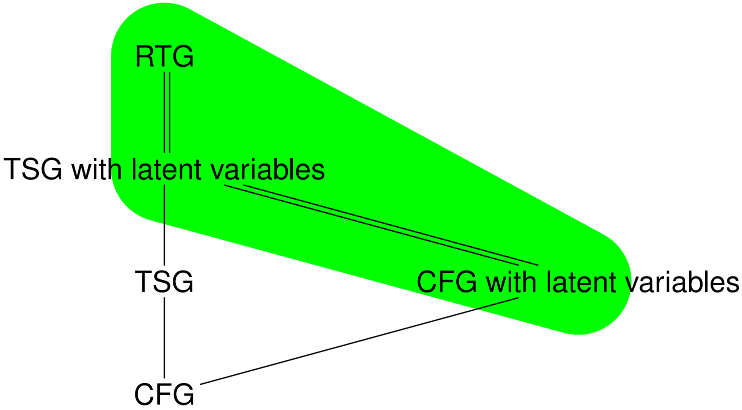
regular tree languages = *relabeled local tree languages*



Regular Tree Grammar

Theorem (Folk, Lore, 1972)

regular tree languages $=$ *relabeled local tree languages*



Berkeley Parser

Example (Berkeley parser — English grammar)

S-1 $\to$ ADJP-2 S-1	$0.0035453455987323125 \cdot 10^{0}$
S-1 $\to$ ADJP-1 S-1	$2.108608433271444 \cdot 10^{-6}$
S-1 $\to$ VP-5 VP-3	$1.6367163259885093 \cdot 10^{-4}$
S-2 $\to$ VP-5 VP-3	$9.724998692152419 \cdot 10^{-8}$
S-1 $\to$ PP-7 VP-0	$1.0686659961009547 \cdot 10^{-5}$
S-9 $\to$ “ NP-3	$0.012551243773149695 \cdot 10^{0}$

$\rightsquigarrow$ Regular tree grammar

Recent NLP Result

Corollary

The grammar of [SHINDO et al., ACL 2012 best paper] can be implemented in the BERKELEY *parser*

Remark

- the main contribution of SHINDO et al. is not the TSGlv
- it is probably the intricate 3-layer back-off model

Recent NLP Result

Corollary

*The grammar of [*SHINDO *et al., ACL 2012 best paper] can be implemented in the* BERKELEY *parser*

Remark

- the main contribution of SHINDO et al. is not the TSGlv
- it is probably the intricate 3-layer back-off model

Recent NLP Result

Corollary

The grammar of [SHINDO et al., ACL 2012 best paper] can be implemented in the BERKELEY *parser*

Remark

- the main contribution of SHINDO et al. is not the TSGlv
- it is probably the intricate 3-layer back-off model

Overview

Tree Automaton

Definition (Thatcher, 1970; Rounds, 1970)

tree automaton is a regular tree grammar in normal form

Remarks

- bottom-up: rules written as $X(q_1, \ldots, q_k) \to q$
- top-down: rules written as $q \to X(q_1, \ldots, q_k)$

Tree Automaton

Definition (THATCHER, 1970; ROUNDS, 1970)

tree automaton is a regular tree grammar in normal form

Remarks

- bottom-up: rules written as $\mathrm{X}(q_1, \ldots, q_k) \to q$
- top-down: rules written as $q \to \mathrm{X}(q_1, \ldots, q_k)$

Determinism

Definition

- top-down deterministic if $\forall q \in Q, k \in \mathbb{N}, \mathrm{X} \in \Sigma$
  $\exists$ at most one $q_1, \ldots, q_k \in Q$: $q \to \mathrm{X}(q_1, \ldots, q_k) \in P$
- bottom-up deterministic if $\forall k \in \mathbb{N}, \mathrm{X} \in \Sigma, q_1, \ldots, q_k \in Q$
  $\exists$ at most one $q \in Q$: $\mathrm{X}(q_1, \ldots, q_k) \to q \in P$

(red determines blue)

Theorem (Thatcher, Wright, 1968; Doner, 1970)

top-down deterministic $\subsetneq$ *bottom-up deterministic* $=$ *RTL*

Proof.

By a standard subset construction and a simple counterexample $\square$

Determinism

tree automata

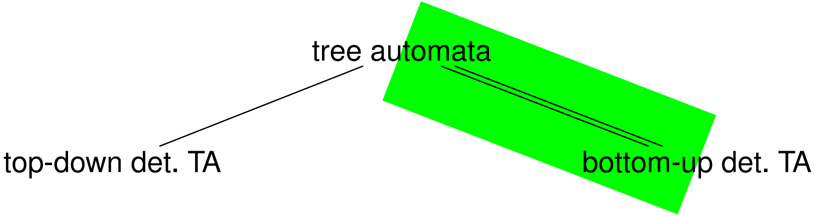
top-down det. TA

bottom-up det. TA

Remark

finite tree languages $\not\subseteq$ top-down deterministic

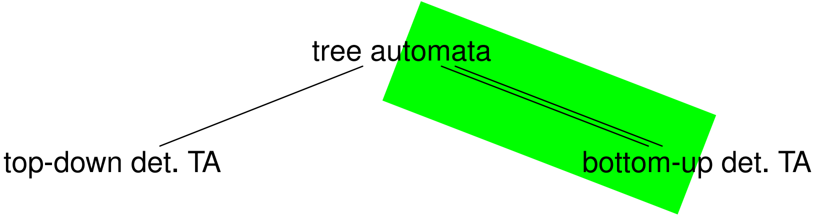
Determinism



Remark

finite tree languages $\not\subseteq$ top-down deterministic

Determinism



Remark

finite tree languages $\not\subseteq$ top-down deterministic

Operations on Regular Tree Languages

Theorem

Regular tree languages are closed under

- *all* BOOLEAN *operations*
- *substitution (quotients) and iteration*
- *(non-deterministic) relabelings*
- *linear homomorphisms*
- *inverse homomorphisms*

Operations on Regular Tree Languages

Theorem

Regular tree languages are closed under substitution

Definition

$L, L' \subseteq T_\Sigma$ tree languages and $\mathrm{X} \in \Sigma$

$$\boxed{L[\mathrm{X} \leftarrow L']}$$

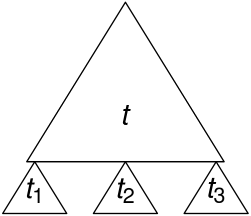
contains all trees obtained from a tree of L
by replacing each leaf labeled X by a tree of L'

Operations on Regular Tree Languages

Theorem

Regular tree languages are closed under substitution

$$\boxed{L[\mathrm{X} \leftarrow L']}$$



$t \in L$

$t_1, t_2, t_3 \in L'$

Efficient Representation

Definition

A tree automaton is minimal in $\mathcal{C}$
if all equivalent tree automata of $\mathcal{C}$ are at least as large

Theorem

Complexity of minimization problems:

outp. \ inp. model	DTA	NTA
$\mathcal{C} =$ DTA	NL	(EXPTIME)
$\mathcal{C} =$ NTA	PSPACE	PSPACE

Overview

Weighted Tree Automaton

Definition (BERSTEL, REUTENAUER, 1982)

A weighted tree automaton is a tree automaton together with a map $c\colon P \to S$

Semantics

- S forms a semiring $(S, +, \cdot, 0, 1)$
- production weights are multiplied $(\cdot)$ in a derivation
- weights of multiple (left-most) derivations for the same tree are summed $(+)$

Weighted Tree Automaton

Definition (Berstel, Reutenauer, 1982)

A weighted tree automaton is a tree automaton together with a map $c \colon P \to S$

Semantics

- S forms a semiring $(S, +, \cdot, 0, 1)$
- production weights are multiplied $(\cdot)$ in a derivation
- weights of multiple (left-most) derivations for the same tree are summed $(+)$

Weighted Tree Automaton

Definition (BERSTEL, REUTENAUER, 1982)

A weighted tree automaton is a tree automaton together with a map $c\colon P \to S$

Semantics

- S forms a semiring $(S, +, \cdot, 0, 1)$
- production weights are multiplied $(\cdot)$ in a derivation
- weights of multiple (left-most) derivations for the same tree are summed $(+)$

Weighted Tree Automaton

Definition (Berstel, Reutenauer, 1982)

A weighted tree automaton is a tree automaton
together with a map $c \colon P \to S$

Semantics

- S forms a semiring $(S, +, \cdot, 0, 1)$
- production weights are multiplied $(\cdot)$ in a derivation
- weights of multiple (left-most) derivations
  for the same tree are summed $(+)$

Weighted Tree Automaton

Remarks

- BERKELEY parser uses weighted tree automata
- but has a best-derivation semantics

Theoretical research

- Minimization wrt. best-derivation semantics
- Minimization wrt. n-best-derivation semantics
- Foundational investigation of those semantics

Weighted Tree Automaton

Remarks

- BERKELEY parser uses weighted tree automata
- but has a best-derivation semantics

Theoretical research

- Minimization wrt. best-derivation semantics
- Minimization wrt. n-best-derivation semantics
- Foundational investigation of those semantics

Weighted Tree Automaton

Remarks

- BERKELEY parser uses weighted tree automata
- but has a best-derivation semantics

Theoretical research

- Minimization wrt. best-derivation semantics
- Minimization wrt. n-best-derivation semantics
- Foundational investigation of those semantics

Weighted Tree Automaton

Remarks

- BERKELEY parser uses weighted tree automata
- but has a best-derivation semantics

Theoretical research

- Minimization wrt. best-derivation semantics
- Minimization wrt. *n*-best-derivation semantics
- Foundational investigation of those semantics

Weighted Tree Automaton

Remarks

- BERKELEY parser uses weighted tree automata
- but has a best-derivation semantics

Theoretical research

- Minimization wrt. best-derivation semantics
- Minimization wrt. *n*-best-derivation semantics
- Foundational investigation of those semantics

Bisimulation Minimization

(needs additive cancellation)

Experiment with BERKELEY parser

	states		productions	
English grammar	1,133	100%	1,842,218	100%
backward minimal	548	48%	626,600	34%
forward minimal	791	70%	767,153	42%
backward/forward minimal	366	32%	272,675	15%
forward/backward minimal	381	34%	309,845	17%
f/b/f/b minimal	375	33%	295,836	16%

These might be buggy

Bisimulation Minimization

(needs additive cancellation)

Experiment with BERKELEY parser

	states		productions	
English grammar	1,133	100%	1,842,218	100%
backward minimal	548	48%	626,600	34%
forward minimal	791	70%	767,153	42%
backward/forward minimal	366	32%	272,675	15%
forward/backward minimal	381	34%	309,845	17%
f/b/f/b minimal	375	33%	295,836	16%

These might be buggy

Full Minimization

Theorem (Berstel, Reutenauer, 1982)

Weighted tree automata over fields can effectively be minimized

Remarks

- even smaller than bisimulation-minimal WTA
- implementations for weighted string automata are efficient
- no implementation for WTA yet

Summary

How to represent a set of trees?

- ~~enumerate them~~
- ~~enumerate them cleverly (e.g., add sharing)~~
- ~~parse forest of a CFG~~
- ~~tree substitution grammar~~
- ~~tree substitution grammar with latent variables~~
- ~~parse forest of a CFG with latent variables~~
- ~~...~~

regular tree grammar

Summary

How to represent a set of trees?

- ~~enumerate them~~
- ~~enumerate them cleverly (e.g., add sharing)~~
- ~~parse forest of a CFG~~
- ~~tree substitution grammar~~
- ~~tree substitution grammar with latent variables~~
- ~~parse forest of a CFG with latent variables~~
- ~~...~~

regular tree grammar

Many theoretical results still to be tried in practice!

Trees Abound
Part II: Tree Transducers

Andreas Maletti

Institute for Natural Language Processing
Universität Stuttgart, Germany

maletti@ims.uni-stuttgart.de

Paris — September 26, 2012

From Automata to Transducers

Idea

Synchronous grammars have synchronous (linked) non-terminals that develop at the same time

Example

- join two productions $q_1 \to r_1$ and $q_2 \to r_2$ to $(q_1, q_2) \to (r_1, r_2)$
- demand $q_1 = q = q_2$ for simplicity and write $r_1 \xrightarrow{q} r_2$
- productions develop input and output trees at the same time

From Automata to Transducers

Idea

Synchronous grammars have synchronous (linked) non-terminals that develop at the same time

Example

- join two productions $q_1 \to r_1$ and $q_2 \to r_2$ to $(q_1, q_2) \to (r_1, r_2)$
- demand $q_1 = q = q_2$ for simplicity and write $r_1 \xrightarrow{q} r_2$
- productions develop input and output trees at the same time

From Automata to Transducers

Idea

Synchronous grammars have synchronous (linked) non-terminals that develop at the same time

Example

- join two productions $q_1 \to r_1$ and $q_2 \to r_2$ to $(q_1, q_2) \to (r_1, r_2)$
- demand $q_1 = q = q_2$ for simplicity and write $r_1 \stackrel{q}{\text{---}} r_2$
- productions develop input and output trees at the same time

From Automata to Transducers

Idea

Synchronous grammars have synchronous (linked) non-terminals that develop at the same time

Example

- join two productions $q_1 \to r_1$ and $q_2 \to r_2$ to $(q_1, q_2) \to (r_1, r_2)$
- demand $q_1 = q = q_2$ for simplicity and write $r_1 \stackrel{q}{\text{---}} r_2$
- productions develop input and output trees at the same time

From Automata to Transducers

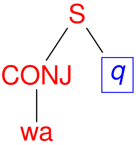
$\boxed{q}$ $\boxed{q}$

Used rule:

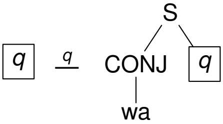
Next rule:

$$\boxed{q} \xrightarrow{q} \mathrm{S}(\mathrm{CONJ}(\mathrm{wa}), \boxed{q})$$

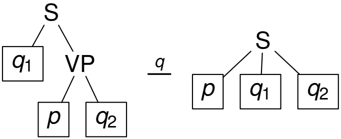
From Automata to Transducers



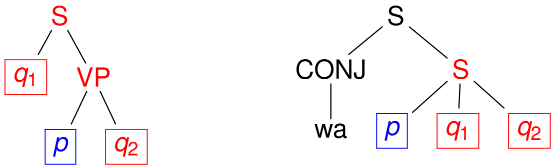
Used rule:



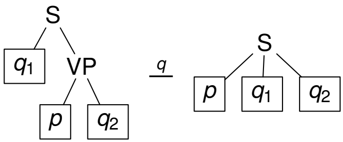
Next rule:



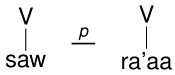
From Automata to Transducers



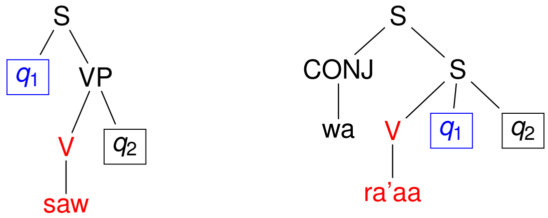
Used rule:



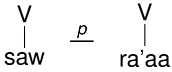
Next rule:



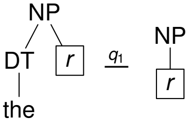
From Automata to Transducers



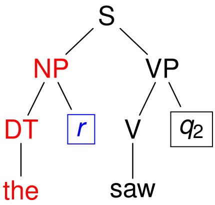
Used rule:

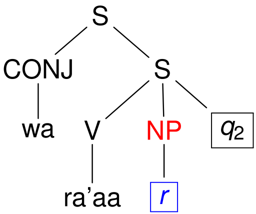


Next rule:

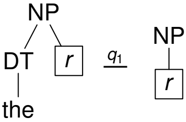


From Automata to Transducers

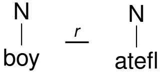




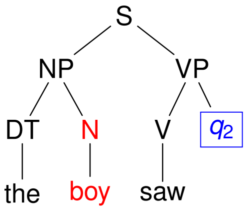
Used rule:

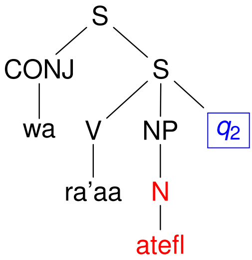


Next rule:

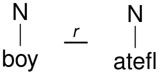


From Automata to Transducers

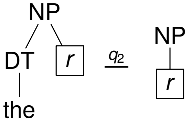




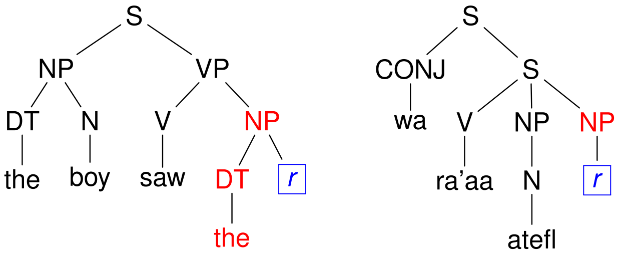
Used rule:



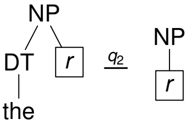
Next rule:



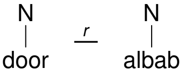
From Automata to Transducers



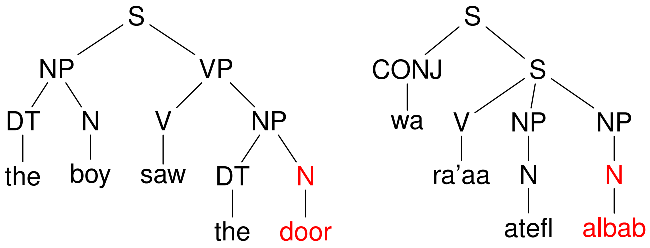
Used rule:



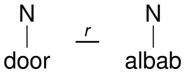
Next rule:



From Automata to Transducers



Used rule:



Next rule:

From Automata to Transducers

Remarks

- synchronization breaks the normalization proof
- the grammar/automaton model makes a difference

Output model: RTG and input model:

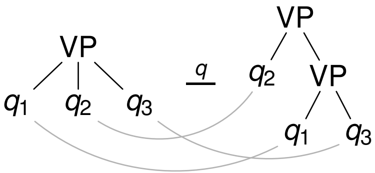
- NTA $\rightsquigarrow$ linear top-down tree transducer
- RTG $\rightsquigarrow$ linear extended top-down tree transducer

Overview

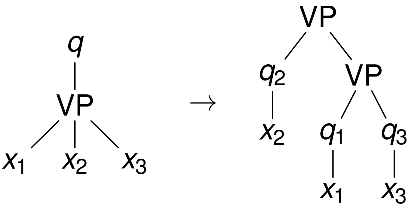
1. Quick Recall
2. Top-down Tree Transducers
3. Extended Top-down Tree Transducers
4. Extended Multi Bottom-up Tree Transducers

Rule Transformation

Synchronous grammar rule:



Top-down tree transducer rule:



Top-down Tree Transducer

Definition (THATCHER, 1970)

A top-down tree transducer is a system $M = (Q, \Sigma, \Delta, I, R)$ with

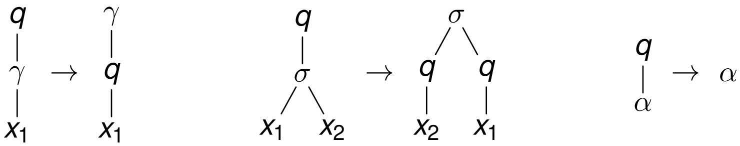
- alphabet of states Q
- input alphabet Σ; output alphabet Δ
- initial states $I \subseteq Q$
- finite set of rules $R \subseteq Q(\Sigma(X)) \times T_\Delta(Q(X))$ such that $\mathrm{var}(r) \subseteq \mathrm{var}(\ell)$ and ℓ is linear for all $(\ell, r) \in R$

Top-down Tree Transducer

Example

Mirror-image top-down tree transducer $(Q, \Sigma, \Sigma, Q, R)$ with

- $Q = \{q\}$
- $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$
- the following rules in R



Top-down Tree Transducer

Definition

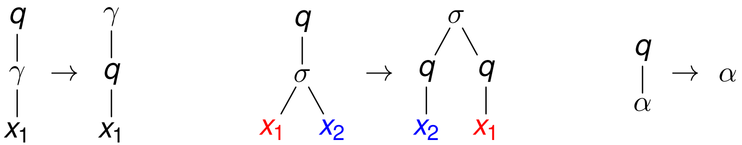
Sentential forms $\xi, \zeta \in T_\Delta(Q(T_\Sigma))$

$$\boxed{\xi \Rightarrow_M \zeta}$$

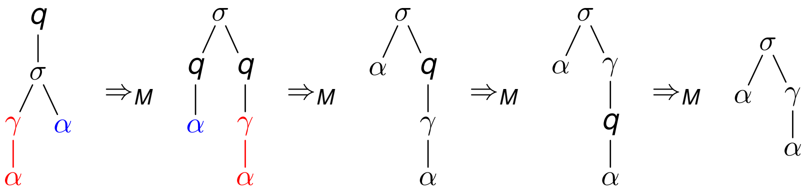
if there exist $\ell \to r \in R$, position $w \in \mathrm{pos}(\xi)$, substitution $\theta \colon X \to T_\Sigma$

- $\xi = \xi[\ell\theta]_w$
- $\zeta = \xi[r\theta]_w$

Derivation Example



Example



Derivation Semantics

Definition

$$M = \{\langle t, u\rangle \in T_\Sigma \times T_\Delta \mid \exists q \in I \colon q(t) \Rightarrow_M^* u\}$$

Derivation Semantics

Definition

$$M = \{\langle t, u\rangle \in T_\Sigma \times T_\Delta \mid \exists q \in I \colon q(t) \Rightarrow_M^* u\}$$

Example

Top-down tree transducer N with

$$\{\langle \sigma(t, u), \sigma(u, t)\rangle \mid t, u \in T_{\{\gamma, \alpha\}}\} \subseteq N$$

$$q(\gamma(x_1)) \to \gamma(q(x_1)) \qquad q(\sigma(x_1, x_2)) \to \sigma(q(x_2), q(x_1)) \qquad q(\alpha) \to \alpha$$

Syntactic Restrictions

Definition

Transducer $M = (Q, \Sigma, \Delta, I, R)$ is

- linear if r is linear for every $\ell \to r \in R$
- nondeleting if $\text{var}(r) = \text{var}(\ell)$ for every $\ell \to r \in R$
- strict if $r \notin Q(X)$ for every $\ell \to r \in R$

Syntactic Restrictions

Definition

Transducer $M = (Q, \Sigma, \Delta, I, R)$ is

- linear if r is linear for every $\ell \to r \in R$
- nondeleting if $\mathrm{var}(r) = \mathrm{var}(\ell)$ for every $\ell \to r \in R$
- strict if $r \notin Q(X)$ for every $\ell \to r \in R$

Example

Mirror-image transducer is linear, nondeleting, and strict (lns-TOP)

$$q(\gamma(x_1)) \to \gamma(q(x_1)) \qquad q(\sigma(x_1, x_2)) \to \sigma(q(x_2), q(x_1)) \qquad q(\alpha) \to \alpha$$

Expressive Power

Properties [ENGELFRIET, 1975]

- T1 "Copying of an input tree and processing the copies differently"
- T2 Cannot inspect deleted input tree

Remark

T2 has been addressed
$\rightsquigarrow$ top-down tree transducers with regular look-ahead [ENGELFRIET, 1977]

Expressive Power

Properties [ENGELFRIET, 1975]

T1 "Copying of an input tree and processing the copies differently"

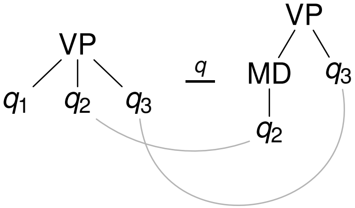
T2 Cannot inspect deleted input tree

Remark

T2 has been addressed
$\rightsquigarrow$ top-down tree transducers with regular look-ahead [ENGELFRIET, 1977]

Regular Look-Ahead

Can be simulated by allowing un-linked nonterminals on the input side



- these develop without effect on the output
- can generate any regular tree language

Composition

Definition (Comp)

$\tau \subseteq T_\Sigma \times T_\Delta$ and $\tau' \subseteq T_\Delta \times T_\Gamma$

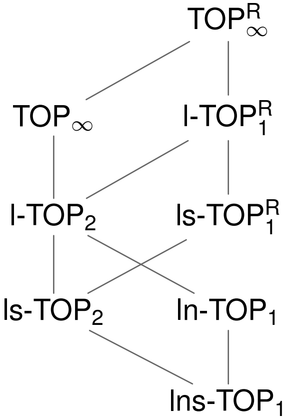
$$\boxed{\tau \,;\, \tau' = \{(s,u) \mid \exists t \in T_\Delta \colon (s,t) \in \tau, (t,u) \in \tau'\}}$$

Example (Double mirror-image)

$$N \,;\, N = \mathrm{id}$$

$$q(\gamma(x_1)) \to \gamma(q(x_1)) \qquad q(\sigma(x_1, x_2)) \to \sigma(q(x_2), q(x_1)) \qquad q(\alpha) \to \alpha$$

Expressive Power

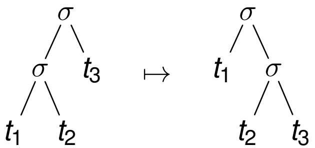


composition closure indicated in subscript

Desirable Properties

Rotations

$$\textsc{Rot} = \{\langle \sigma(\sigma(t_1, t_2), t_3),\ \sigma(t_1, \sigma(t_2, t_3))\rangle \mid t_1, t_2, t_3 \in T_\Sigma\}$$



Preservation of regularity (Pres)

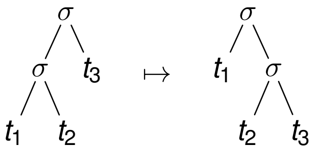
Given $\tau \subseteq T_\Sigma \times T_\Delta$ and $L \subseteq T_\Sigma$ regular, is $\tau(L)$ regular?

$$\boxed{\tau(L) = \{u \mid \exists t \in L\colon (t, u) \in \tau\}}$$

Desirable Properties

Rotations

$$\text{ROT} = \{\langle \sigma(\sigma(t_1, t_2), t_3),\ \sigma(t_1, \sigma(t_2, t_3)) \rangle \mid t_1, t_2, t_3 \in T_\Sigma\}$$



Preservation of regularity (PRES)

Given $\tau \subseteq T_\Sigma \times T_\Delta$ and $L \subseteq T_\Sigma$ regular, is $\tau(L)$ regular?

$$\boxed{\tau(L) = \{u \mid \exists t \in L \colon (t, u) \in \tau\}}$$

Summary

Model \ Criterion	ROT	SYM	PRES	PRES^{-1}	COMP
Ins-TOP	✗	✗	✓	✓	✓
In-TOP	✗	✗	✓	✓	✓
Is-TOP	✗	✗	✓	✓	✗$_2$
I-TOP	✗	✗	✓	✓	✗$_2$
Is-TOP^R	✗	✗	✓	✓	✓
I-TOP^R	✗	✗	✓	✓	✓
TOP	✓	✗	✗	✓	✗$_\infty$
TOP^R	✓	✗	✗	✓	✗$_\infty$

(SYM = symmetric)

Overview

Extended Top-down Tree Transducer

Definition (GRAEHL et al., 2009)

A top-down tree transducer is a system $M = (Q, \Sigma, \Delta, I, R)$

- finite set of states Q
- input alphabet Σ; output alphabet Δ
- initial states $I \subseteq Q$
- finite set of rules $R \subseteq Q(\Sigma(X)) \times T_\Delta(Q(X))$ such that $\mathrm{var}(r) \subseteq \mathrm{var}(\ell)$ and ℓ is linear for all $(\ell, r) \in R$

Extended Top-down Tree Transducer

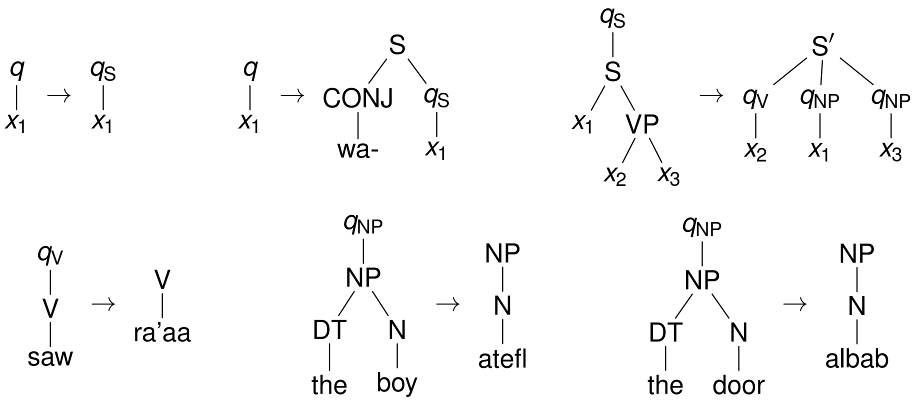
Definition (GRAEHL et al., 2009)

An extended top-down tree transducer is a system $M = (Q, \Sigma, \Delta, I, R)$

- finite set of states Q
- input alphabet Σ; output alphabet Δ
- initial states $I \subseteq Q$
- finite set of rules $R \subseteq Q(T_\Sigma(X)) \times T_\Delta(Q(X))$ such that $\text{var}(r) \subseteq \text{var}(\ell)$ and ℓ is linear for all $(\ell, r) \in R$

Extended Top-down Tree Transducer

Example



Extended Top-down Tree Transducer

Definition

Sentential forms $\xi, \zeta \in T_\Delta(Q(T_\Sigma))$

$$\boxed{\xi \Rightarrow_M \zeta}$$

if there exist $\ell \to r \in R$, position $w \in \mathrm{pos}(\xi)$, substitution $\theta \colon X \to T_\Sigma$

- $\xi = \xi[\ell\theta]_w$
- $\zeta = \xi[r\theta]_w$

Definition

$$M = \{\langle t, u\rangle \in T_\Sigma \times T_\Delta \mid \exists q \in I \colon q(t) \Rightarrow_M^* u\}$$

Extended Top-down Tree Transducer

Definition

Sentential forms $\xi, \zeta \in T_\Delta(Q(T_\Sigma))$

$$\boxed{\xi \Rightarrow_M \zeta}$$

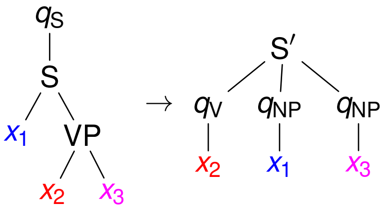
if there exist $\ell \to r \in R$, position $w \in \mathrm{pos}(\xi)$, substitution $\theta \colon X \to T_\Sigma$

- $\xi = \xi[\ell\theta]_w$
- $\zeta = \xi[r\theta]_w$

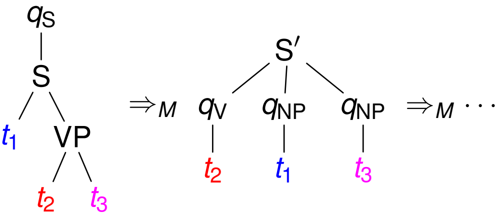
Definition

$$M = \{\langle t, u\rangle \in T_\Sigma \times T_\Delta \mid \exists q \in I \colon q(t) \Rightarrow_M^* u\}$$

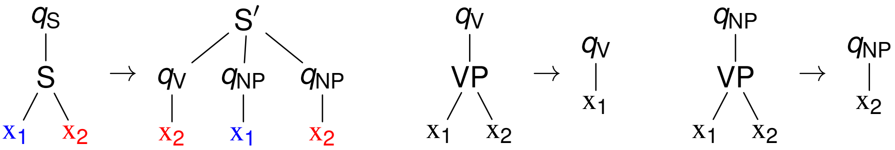
Derivation Example



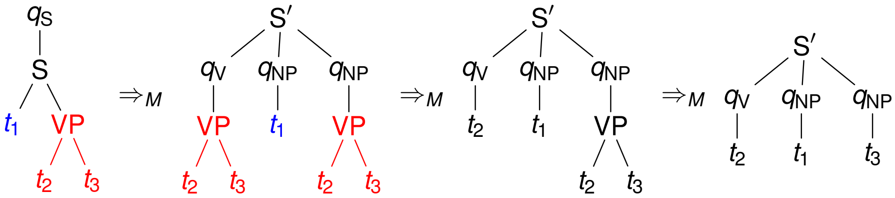
Example



Simulation by Copying and Deletion



Example



Syntactic Restrictions

Definition

Extended top-down tree transducer $M = (Q, \Sigma, \Delta, I, R)$ is

- linear, nondeleting, strict as before
- ε-free if $\ell \notin Q(X)$ for every $\ell \to r \in R$

Syntactic Restrictions

Definition

Extended top-down tree transducer $M = (Q, \Sigma, \Delta, I, R)$ is

- linear, nondeleting, strict as before
- ε-free if $\ell \notin Q(X)$ for every $\ell \to r \in R$

Syntactic Restrictions

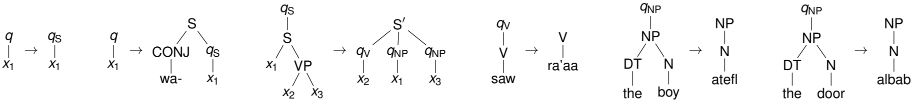
Definition

Extended top-down tree transducer $M = (Q, \Sigma, \Delta, I, R)$ is

- linear, nondeleting, strict as before
- ε-free if $\ell \notin Q(X)$ for every $\ell \to r \in R$

Example

Our example transducer is linear, nondeleting, ~~strict~~, and ~~ε-free~~



Expressive Power

Properties [Graehl et al., 2009]

- X1 Finite look-ahead
- X2 Deep attachment of variables
- X3 Infinitely many outputs for one input

Remark

T1 and T2 still apply

Expressive Power

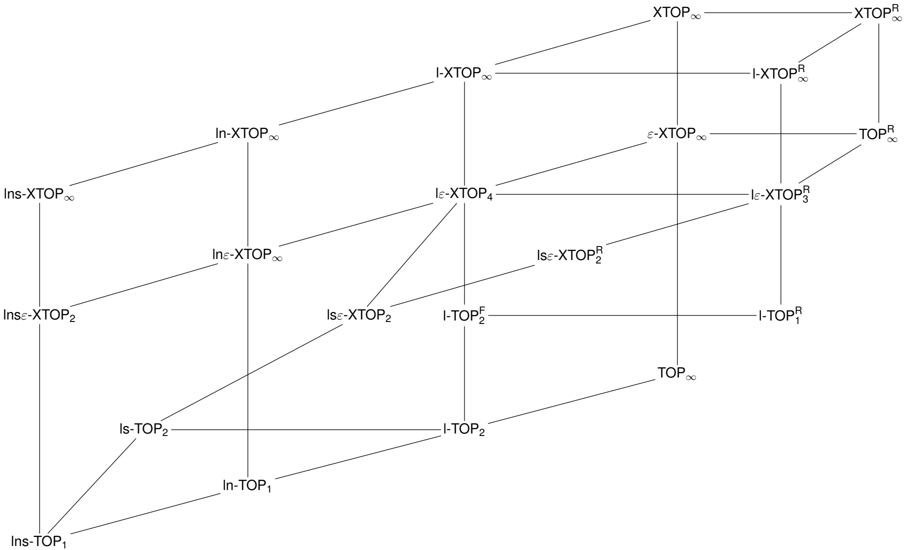
Properties [GRAEHL et al., 2009]

- X1 Finite look-ahead
- X2 Deep attachment of variables
- X3 Infinitely many outputs for one input

Remark

T1 and T2 still apply

Expressive Power



composition closure indicated in subscript

Summary

Model \ Criterion	ROT	SYM	PRES	PRES^{-1}	COMP
ln-TOP	✗	✗	✓	✓	✓
l-TOP	✗	✗	✓	✓	✗$_2$
l-TOP^R	✗	✗	✓	✓	✓
TOP^R	✓	✗	✗	✓	✗$_\infty$
lnsε-XTOP	✓	✓	✓	✓	✗$_2$
lns-XTOP	✓	✗	✓	✓	✗$_\infty$
lsε-XTOP$^{(R)}$	✓	✗	✓	✓	✗$_2$
lε-XTOP	✓	✗	✓	✓	✗$_4$
lε-XTOP^R	✓	✗	✓	✓	✗$_3$
(s)l-XTOP$^{(R)}$	✓	✗	✓	✓	✗$_\infty$
XTOP	✓	✗	✗	✓	✗$_\infty$
XTOP^R	✓	✗	✗	✓	✗$_\infty$

Overview

Extended Multi Bottom-up Tree Transducer

Definition

An extended multi bottom-up tree transducer $M = (Q, \Sigma, \Delta, F, R)$ with

- ranked alphabet of states Q
- input alphabet Σ; output alphabet Δ
- final states $F \subseteq Q_1$ (all unary)
- finite set of rules $R \subseteq T_\Sigma(Q(X)) \times Q(T_\Delta(X))$ such that $\text{var}(r) \subseteq \text{var}(\ell)$ and ℓ is linear for all $(\ell, r) \in R$

Properties

linear, nondeleting, strict, ε-free as before

Extended Multi Bottom-up Tree Transducer

Definition

An extended multi bottom-up tree transducer $M = (Q, \Sigma, \Delta, F, R)$ with

- ranked alphabet of states Q
- input alphabet Σ; output alphabet Δ
- final states $F \subseteq Q_1$ (all unary)
- finite set of rules $R \subseteq T_\Sigma(Q(X)) \times Q(T_\Delta(X))$ such that $\text{var}(r) \subseteq \text{var}(\ell)$ and ℓ is linear for all $(\ell, r) \in R$

Properties

linear, nondeleting, strict, ε-free as before

Extended Multi Bottom-up Tree Transducer

Example (Duplication)

Extended multi bottom-up tree transducer $(Q, \Sigma, \Sigma, \{f\}, R)$

- $Q = \{q^{(2)}, f^{(1)}\}$
- $\Sigma = \{\sigma, a, b, e\}$
- R contains:

$$e \to q(e, e) \qquad a(q(x_1, x_2)) \to q(a(x_1), a(x_2)) \qquad b(q(x_1, x_2)) \to q(b(x_1), b(x_2)) \qquad q(x_1, x_2) \to f(\sigma(x_1, x_2))$$

Properties

linear, nondeleting, strict, and ~~ε-free~~

Extended Multi Bottom-up Tree Transducer

Example (Duplication)

Extended multi bottom-up tree transducer $(Q, \Sigma, \Sigma, \{f\}, R)$

- $Q = \{q^{(2)}, f^{(1)}\}$
- $\Sigma = \{\sigma, a, b, e\}$
- R contains:

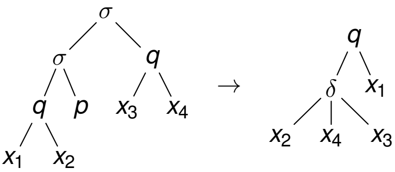
$$e \to q(e, e) \qquad a(q(x_1, x_2)) \to q(a(x_1), a(x_2)) \qquad b(q(x_1, x_2)) \to q(b(x_1), b(x_2)) \qquad q(x_1, x_2) \to f(\sigma(x_1, x_2))$$

Properties

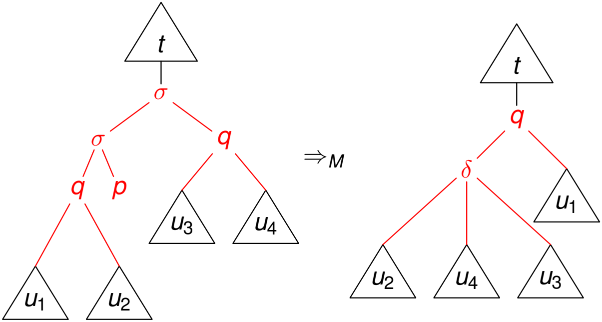
linear, nondeleting, strict, and ~~ε-free~~

Extended Multi Bottom-up Tree Transducer

Rule:



Derivation:



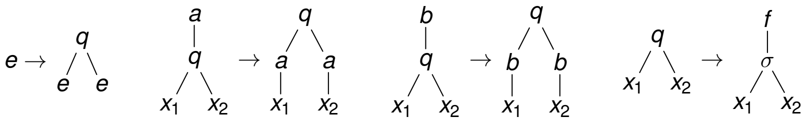
Extended Multi Bottom-up Tree Transducer

$$e \to q(e, e) \qquad a(q(x_1, x_2)) \to q(a(x_1), a(x_2)) \qquad b(q(x_1, x_2)) \to q(b(x_1), b(x_2)) \qquad q(x_1, x_2) \to f(\sigma(x_1, x_2))$$

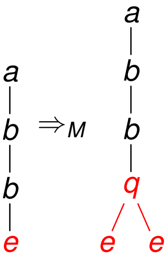
Example (Derivation)

$$a(b(b(e)))$$

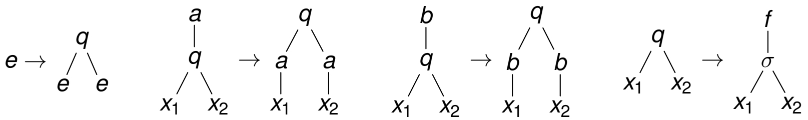
Extended Multi Bottom-up Tree Transducer



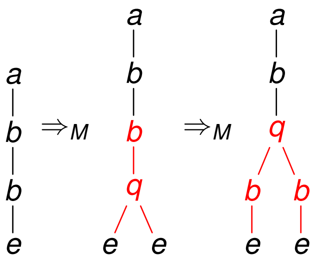
Example (Derivation)



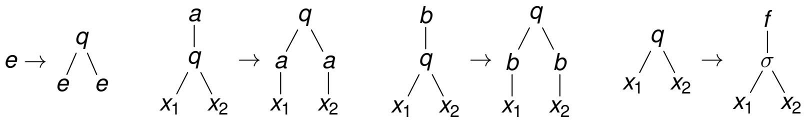
Extended Multi Bottom-up Tree Transducer



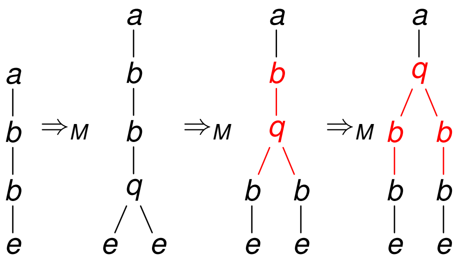
Example (Derivation)



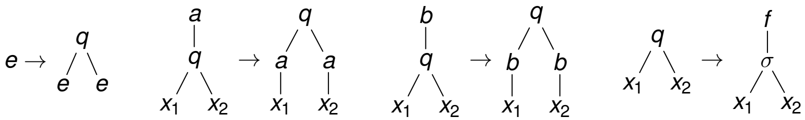
Extended Multi Bottom-up Tree Transducer



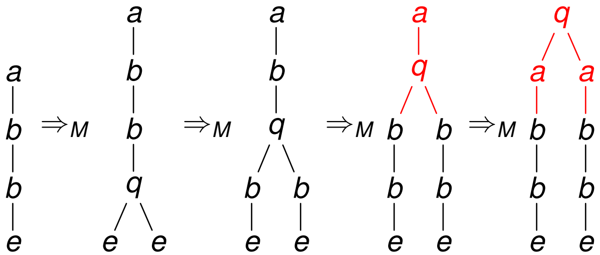
Example (Derivation)



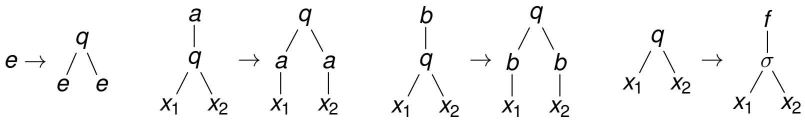
Extended Multi Bottom-up Tree Transducer



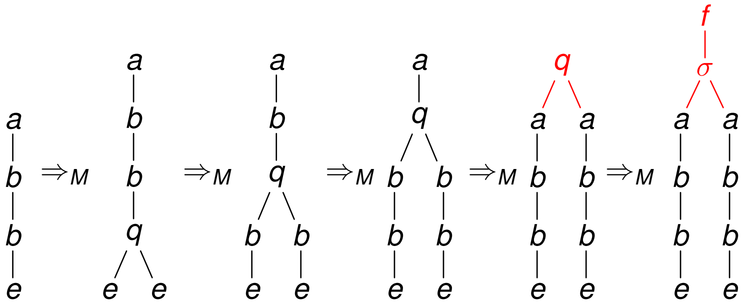
Example (Derivation)



Extended Multi Bottom-up Tree Transducer



Example (Derivation)



Extended Multi Bottom-up Tree Transducer

Definition

$$\tau_M = \{(t, u) \in T_\Sigma \times T_\Delta \mid \exists q \in F \colon t \Rightarrow_M^* q(u)\}$$

Extended Multi Bottom-up Tree Transducer

Definition

$$\tau_M = \{(t, u) \in T_\Sigma \times T_\Delta \mid \exists q \in F \colon t \Rightarrow_M^* q(u)\}$$

Example (Duplication)

It computes $\{(t, \sigma(t, t)) \mid t \in T_\Sigma\}$

Its image is not a regular tree language

Subclasses

Definition

Extended multi bottom-up tree transducer $(Q, \Sigma, \Delta, F, R)$ is

- extended bottom-up tree transducer if $Q = Q_1$
- multi bottom-up tree transducer if $\ell \in \Sigma(Q(X))$ for all $\ell \to r \in R$
- bottom-up tree transducer if both previous conditions hold

Subclasses

Definition

Extended multi bottom-up tree transducer $(Q, \Sigma, \Delta, F, R)$ is

- extended bottom-up tree transducer if $Q = Q_1$
- multi bottom-up tree transducer if $\ell \in \Sigma(Q(X))$ for all $\ell \to r \in R$
- bottom-up tree transducer if both previous conditions hold

Example (Duplication)

$$e \to q(e, e) \qquad a(q(x_1, x_2)) \to q(a(x_1), a(x_2)) \qquad b(q(x_1, x_2)) \to q(b(x_1), b(x_2)) \qquad q(x_1, x_2) \to f(\sigma(x_1, x_2))$$

Expressive Power

Theorem (Engelfriet et al. '09)

$$\text{l-XTOP}^{\text{R}} = \text{l-XBOT}$$

Proof.

Standard construction trading input-deletion for output-deletion
see l-TOP $\subseteq$ l-BOT by [Engelfriet '75] □

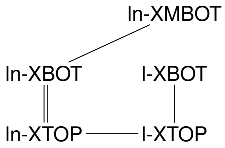
Expressive Power

Theorem (Engelfriet et al. ’09)

$$\text{l-XTOP}^{\text{R}} = \text{l-XBOT}$$

Proof.

Standard construction trading input-deletion for output-deletion
see l-TOP $\subseteq$ l-BOT by [Engelfriet ’75] $\square$



Expressive Power

Theorem (Engelfriet et al. '09)

$$\text{XMBOT} = \text{n-XMBOT}$$

Proof.

- guess subtrees that will be deleted
- process them in nullary states (i.e. look-ahead) □

Expressive Power

Theorem (ENGELFRIET et al. ’09)

$$\text{XMBOT} = n\text{-XMBOT}$$

Proof.

- guess subtrees that will be deleted
- process them in nullary states (i.e. look-ahead) $\square$

Expressive Power

Theorem (ENGELFRIET et al. '09)

$$\text{XMBOT} = \text{n-XMBOT}$$

Proof.

- guess subtrees that will be deleted
- process them in nullary states (i.e. look-ahead) $\square$

$$\begin{tikzcd}
 & \text{ln-XMBOT} \\
\text{ln-XBOT} \arrow[ur, dash] \arrow[d, equal] & \text{l-XBOT} \arrow[d, dash] \\
\text{ln-XTOP} \arrow[r, dash] & \text{l-XTOP}
\end{tikzcd}$$

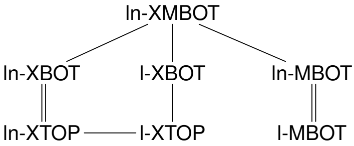
Expressive Power

Theorem (ENGELFRIET et al. ’09)

$$\text{XMBOT} = \text{n-XMBOT}$$

Proof.

- guess subtrees that will be deleted
- process them in nullary states (i.e. look-ahead) □



Expressive Power

Theorem (ENGELFRIET et al. ’09)

$$\varepsilon\text{-XMBOT} = \text{MBOT}$$

Proof.

- decompose large left-hand sides using “multi”-states
- attach finite effect of ε-rules □

Expressive Power

Theorem (ENGELFRIET et al. '09)

$$\varepsilon\text{-XMBOT} = \text{MBOT}$$

Proof.

- decompose large left-hand sides using "multi"-states
- attach finite effect of ε-rules □

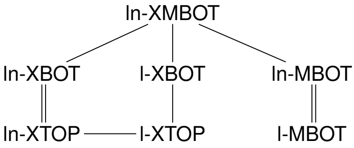
Expressive Power

Theorem (ENGELFRIET et al. '09)

$$\varepsilon\text{-XMBOT} = \text{MBOT}$$

Proof.

- decompose large left-hand sides using "multi"-states
- attach finite effect of ε-rules $\square$



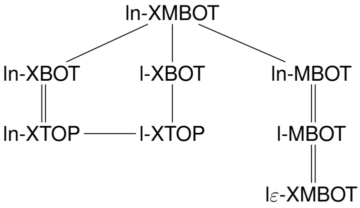
Expressive Power

Theorem (Engelfriet et al. '09)

$$\varepsilon\text{-XMBOT} = \text{MBOT}$$

Proof.

- decompose large left-hand sides using "multi"-states
- attach finite effect of ε-rules □



Expressive Power

Definition

XTOP M sensible if $|u| \in \mathcal{O}(|t|)$ for all $(t, u) \in M$

Theorem (Maletti '12)

$$\text{sensible XTOP} \subseteq \text{ln-MBOT}$$

Proof.

- use (essentially) construction of [Engelfriet, Maneth '03]
- obtain finitely copying ε-XTOP
- apply [Engelfriet et al. '09] to obtain lε-XMBOT
- previous theorems yield ln-MBOT □

Expressive Power

Definition

XTOP M sensible if $|u| \in \mathcal{O}(|t|)$ for all $(t, u) \in M$

Theorem (Maletti '12)

$$\text{sensible XTOP} \subseteq \text{ln-MBOT}$$

Proof.

- use (essentially) construction of [Engelfriet, Maneth '03]
- obtain finitely copying ε-XTOP
- apply [Engelfriet et al. '09] to obtain lε-XMBOT
- previous theorems yield ln-MBOT □

Expressive Power

Definition

XTOP M sensible if $|u| \in \mathcal{O}(|t|)$ for all $(t, u) \in M$

Theorem (Maletti '12)

$$\text{sensible XTOP} \subseteq \text{ln-MBOT}$$

Proof.

- use (essentially) construction of [Engelfriet, Maneth '03]
- obtain finitely copying ε-XTOP
- apply [Engelfriet et al. '09] to obtain lε-XMBOT
- previous theorems yield ln-MBOT □

Expressive Power

Corollary

All relevant extended top-down tree transducers can be simulated by linear and nondeleting extended multi bottom-up tree transducers

Further Properties

Theorem

$$\text{ln-MBOT} \not\subseteq \text{XTOP}^{\text{R}}$$

Theorem (GILDEA '12)

$$\underset{\text{out}}{\mathit{yield}}(\text{ln-MBOT}) = \text{LCFRS}$$

Further Properties

Theorem

$$\text{ln-MBOT} \not\subseteq \text{XTOP}^{\text{R}}$$

Theorem (GILDEA '12)

$$\underset{\text{out}}{\textit{yield}}(\text{ln-MBOT}) = \text{LCFRS}$$

Summary

Model \ Criterion	Rot	Sym	Pres	Pres^{-1}	Comp
ln-TOP	✗	✗	✓	✓	✓
l-TOP	✗	✗	✓	✓	$✗_2$
l-TOP$^{\mathrm{R}}$	✗	✗	✓	✓	✓
TOP$^{\mathrm{R}}$	✓	✗	✗	✓	$✗_\infty$
lnsε-XTOP	✓	✓	✓	✓	$✗_2$
lns-XTOP	✓	✗	✓	✓	$✗_\infty$
lsε-XTOP$^{(\mathrm{R})}$	✓	✗	✓	✓	$✗_2$
lε-XTOP	✓	✗	✓	✓	$✗_4$
lε-XTOP$^{\mathrm{R}}$	✓	✗	✓	✓	$✗_3$
(s)l-XTOP$^{(\mathrm{R})}$	✓	✗	✓	✓	$✗_\infty$
XTOP$^{(\mathrm{R})}$	✓	✗	✗	✓	$✗_\infty$
l(n)-XMBOT	✓	✗	✗	✓	✓
XMBOT	✓	✗	✗	✓	$✗_\infty$
reg.-preserving l-XMBOT	✓	✗	✓	✓	✓
invertable l-XMBOT	✓	✓	✓	✓	✓