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
Motivation

Trees?

We must bear in mind the Community as 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

\[
\begin{align*}
S & \rightarrow \text{NP VP} & \text{VP} & \rightarrow \text{MD VP} \\
\text{NP} & \rightarrow \text{NP PP} & \text{VP} & \rightarrow \text{VB PP NP} \\
\text{MD} & \rightarrow \text{must} \\
\end{align*}
\]
Parse Forest of a CFG

Example

\[
\begin{align*}
S & \rightarrow NP \ VP \\
NP & \rightarrow NP \ PP \\
MD & \rightarrow must \\
VP & \rightarrow MD \ VP \\
VP & \rightarrow VB \ PP \ NP \\
& \ldots
\end{align*}
\]
Motivation

Parse Forest of a CFG

Example

S → NP VP
NP → NP PP
MD → must
VP → MD VP
VP → VB PP NP

Trees Abound — Part I: Tree Automata
A. Maletti
Parse Forest of a CFG

Example

\[
S \rightarrow \text{NP } \text{VP} \\
\text{NP} \rightarrow \text{NP } \text{PP} \\
\text{MD} \rightarrow \text{must} \\
\text{VP} \rightarrow \text{MD } \text{VP} \\
\text{VP} \rightarrow \text{VB } \text{PP } \text{NP} \\
\ldots
\]

Trees Abound — Part I: Tree Automata
Motivation

Parse Forest of a CFG

Example

\[ S \rightarrow NP \ VP \]
\[ NP \rightarrow NP \ PP \]
\[ MD \rightarrow \text{must} \]
\[ VP \rightarrow MD \ VP \]
\[ VP \rightarrow VB \ PP \ NP \]
\[ \ldots \]

Trees Abound — Part I: Tree Automata
Parse Forest of a CFG

Example

\[ S \rightarrow NP \ VP \]
\[ NP \rightarrow NP \ PP \]
\[ MD \rightarrow must \]

\[ VP \rightarrow MD \ VP \]
\[ VP \rightarrow VB \ PP \ NP \]

Trees Abound — Part I: Tree Automata
**Motivation**

**Parse Forest of a CFG**

**Example**

\[
\begin{align*}
S &\rightarrow NP \ VP \\
NP &\rightarrow NP \ PP \\
VP &\rightarrow MD \ VP \\
VP &\rightarrow VB \ PP \ NP \\
MD &\rightarrow \text{must} \\
\end{align*}
\]

Trees Abound — Part I: Tree Automata

A. Maletti · 4
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
Motivation

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:

- \( S \)
- \( NP \)
- \( PRP\$ \)
- \( My \)
- \( NN \)
- \( dog \)
- \( VP \)
- \( VBZ \)
- \( sleeps \)
- \( S \)
- \( NP \)
- \( PRP \)
- \( I \)
- \( VBD \)
- \( scored \)
- \( RB \)
- \( well \)

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 $L \rightarrow R_1 R_2 R_3$ represented by tree
  
  $L$
  
  $R_1$
  
  $R_2$
  
  $R_3$

- “Glue” fragments together to obtain larger trees:
  
  $S$

But why only small tree fragments?
Local Tree Grammar

Generalization

- CFG production $L \rightarrow R_1 R_2 R_3$ represented by tree
  \[ \begin{array}{c}
  \text{L} \\
  \text{R}_1 \quad \text{R}_2 \quad \text{R}_3
  \end{array} \]

- “Glue” fragments together to obtain larger trees:
  \[ \begin{array}{c}
  \text{S} \\
  \text{NP} \quad \text{VP}
  \end{array} \]

- But why only small tree fragments?
Local Tree Grammar

Generalization

- CFG production \( L \rightarrow R_1 \ R_2 \ R_3 \) represented by tree

```
    L
   / \  /
 R_1 R_2 R_3
```

- “Glue” fragments together to obtain larger trees:

```
    S
   /  \
 NP  VP
   /    \
PRP               
```

- But why only small tree fragments?
Local Tree Grammar

Generalization

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

```
   L
   /\  \\
  R_1  R_2  R_3
```

- "Glue" fragments together to obtain larger trees:

```
   S
   /\  \\
  NP  VP
   /  \\
PRP  \\
   We
```

- But why only small tree fragments?
Local Tree Grammar

Generalization

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

```
         L
        /|
       / |\
  R_1    R_2  R_3
```

- “Glue” fragments together to obtain larger trees:

```
  S
 /|
/ |\
NP VP
 /|
/ |\
PRP MD VP
 /|
/ |\
We
```

But why only small tree fragments?
Local Tree Grammar

Generalization

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

  $\begin{align*}
  &L \\
  &\quad \downarrow \\
  &R_1 \quad R_2 \quad R_3
  \end{align*}$

- “Glue” fragments together to obtain larger trees:

  $\begin{align*}
  &S \\
  &\quad \downarrow \\
  &NP \quad VP \\
  &\quad \downarrow \quad \downarrow \\
  &PRP \quad MD \quad VP \\
  &\quad \downarrow \quad \downarrow \\
  &We \quad must
  \end{align*}$

- But why only small tree fragments?
Local Tree Grammar

Generalization

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

```
    L
   / \  /
R_1  R_2 R_3
```

- “Glue” fragments together to obtain larger trees:

```
    S
   /  
NP VP
   /    /
PRP MD VP
  /  / 
We must VB PP NP
```

But why only small tree fragments?
Local Tree Grammar

Generalization

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

```
  L
 / \   / \   / \\
R_1  R_2  R_3
```

- “Glue” fragments together to obtain larger trees:

```
  S
   /  \\
NP   VP
    /     \\
PRP  MD   VP
   /   /     \\
We  must  PP   NP
```

- 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])

- VP
  - VBD
  - NP
    - CD
  - PP
- S
  - NP
    - PRP
  - VP
- S
  - NP
  - TO
  - VP
Tree Substitution Grammar

Theorem

\[ \text{local tree languages} \subseteq \text{tree substitution languages} \]

Proof.

Can trivially express all finite tree languages

Remarks

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

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 $\implies$ artificial example

\[
L_1 = \{ S(C^n(a), a) \mid n \in \mathbb{N} \} \\
L_2 = \{ S(C^n(b), b) \mid n \in \mathbb{N} \}
\]

Their union is not a tree substitution language
Tree Substitution Grammar

Theorem

Tree substitution languages are not closed under intersection

Proof.

Ideas?
# Tree Substitution Grammar

## Experiment [POST, GILDEA, ACL 2009]

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Size</th>
<th>Prec.</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFG</td>
<td>46k</td>
<td>75.37</td>
<td>70.05</td>
<td>72.61</td>
</tr>
<tr>
<td>“spinal” TSG</td>
<td>190k</td>
<td>80.30</td>
<td>78.10</td>
<td>79.18</td>
</tr>
<tr>
<td>“minimal subset” TSG</td>
<td>2,560k</td>
<td>76.40</td>
<td>78.29</td>
<td>77.33</td>
</tr>
</tbody>
</table>

(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)

```
S-1
  NP-0   VP-0
    PRP-1

S-0
  NP-1   VP-0
    VBP-3   NP-2
      love

S-0
  NP-0   VP-2
    TO-0   VP-1
```
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]

<table>
<thead>
<tr>
<th>Grammar</th>
<th>F1 score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$</td>
</tr>
<tr>
<td>TSG [Post, Gildea, 2009]</td>
<td>82.6</td>
</tr>
<tr>
<td>TSG [Cohn et al., 2010]</td>
<td>85.4</td>
</tr>
<tr>
<td>CFG lv [Collins, 1999]</td>
<td>88.6</td>
</tr>
<tr>
<td>CFG lv [Petrov, Klein, 2007]</td>
<td>90.6</td>
</tr>
<tr>
<td>CFG lv [Petrov, 2010]</td>
<td></td>
</tr>
<tr>
<td>TSG lv (single)</td>
<td>91.6</td>
</tr>
<tr>
<td>TSG lv (multiple)</td>
<td>92.9</td>
</tr>
</tbody>
</table>

Discriminative Parsers

| CARRERAS et al., 2008 | 91.1 |
| CHARNIAK, JOHNSON, 2005 | 92.0 | 91.4 |
| HUANG, 2008            | 92.3 | 91.7 |
Motivation

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

1. Motivation

2. Regular Tree Grammars

3. Theoretical Properties

4. Excursion
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 $\Sigma$
- initial nonterminals $I \subseteq Q$
- finite set of productions $P \subseteq Q \times T_{\Sigma}(Q)$

Remark

Instead of $(q, r)$ we write $q \rightarrow r$
Regular Tree Grammar

Example

- \( Q = \{ q_0, q_1, q_2, q_3, q_4, q_5, q_6 \} \)
- \( \Sigma = \{ \text{VP, NP, S, \ldots} \} \)
- \( I = \{ q_0 \} \)
- and the following productions:

\[
\begin{align*}
q_4 & \rightarrow q_5 \\
q_0 & \rightarrow \text{NP} q_4 \\
q_0 & \rightarrow S q_6 \\
q_0 & \rightarrow S \text{NP} \\
q_0 & \rightarrow q_6 \text{VP} \\
\end{align*}
\]
Regular Tree Grammar

Definition (Derivation Semantics)
Sentential forms: \( t, u \in T_\Sigma(Q) \)

\[ t \Rightarrow_G u \]

if there exist position \( w \in \text{pos}(t) \) and production \( q \rightarrow r \in P \)
- \( t = t[q]_w \)
- \( u = t[r]_w \)

Definition (Recognized tree language)

\[ L(G) = \{ t \in T_\Sigma \mid \exists q \in I : q \Rightarrow^*_G t \} \]
Regular Tree Grammar

Definition (Derivation Semantics)

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

\[ t \Rightarrow_G u \]

if there exist position \( w \in \text{pos}(t) \) and production \( q \rightarrow r \in P \)

- \( t = t[q]_w \)
- \( u = t[r]_w \)

Definition (Recognized tree language)

\[ L(G) = \{ t \in T_\Sigma \mid \exists q \in I : q \Rightarrow^*_G t \} \]
Regular Tree Grammar

Example (Productions)

\[ q_4 \rightarrow q_5 \text{ VP } q_3 \]
\[ q_2 \]
\[ q_0 \rightarrow q_1 \text{ NP } q_4 \]
\[ q_0 \rightarrow q_6 \text{ S } q_4 \]

Example (Derivation)

\[ q_0 \]
Regular Tree Grammar

Example (Productions)

\[ q_4 \rightarrow q_5 NP q_3 \]
\[ q_0 \rightarrow NP q_4 \]
\[ q_0 \rightarrow VP q_6 VP \]

Example (Derivation)

\[ q_0 \Rightarrow G NP q_4 \]

Trees Abound — Part I: Tree Automata
Regular Tree Grammar

Example (Productions)

\[
q_4 \rightarrow q_5 \\
\text{NP} \\
q_2
\]

\[
q_0 \rightarrow q_1 \\
\text{NP} \\
q_4
\]

\[
q_0 \rightarrow q_6 \\
\text{VP} \\
q_2 \\
q_4
\]

Example (Derivation)

\[
q_0 \Rightarrow G \\
\text{NP} \\
q_4 \Rightarrow G \\
q_1
\]

\[
q_0 \Rightarrow G \\
\text{NP} \\
q_1 \\
q_5 \\
\text{VP} \\
q_3 \\
q_2
\]
Regular Tree Grammar

Theorem

\[ \text{tree substitution languages} \subseteq \text{regular tree languages} \]

Proof.

We can express the union counterexample easily.

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 $q \rightarrow r \in P$
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 \( q \rightarrow r \in P \)

Example (Productions)

\[
\begin{align*}
q_4 & \rightarrow q_5 \quad NP \quad q_3 \\
\quad q_2 & \\
q_0 & \rightarrow NP \quad q_4 \\
\quad q_1 & \\
q_0 & \rightarrow S \\
\quad q_6 & \\
\quad q_2 & \quad VP \quad q_4
\end{align*}
\]
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

\[
q_0 \rightarrow q_6 \quad VP \quad q_2 \quad q_4 \\
q_0 \rightarrow S \quad \quad q_6 \quad q \\
\bar{q} \rightarrow VP \quad q_2 \quad q_4
\]
Theorem (FOLK, LORE, 1972)

regular tree languages = relabeled local tree languages
Theorem (FOLK, LORE, 1972)

regular tree languages = relabeled local tree languages
Berkeley Parser

Example (Berkeley parser — English grammar)

\[
\begin{align*}
S-1 & \rightarrow \text{ADJP-2 } S-1 & 0.0035453455987323125 \cdot 10^0 \\
S-1 & \rightarrow \text{ADJP-1 } S-1 & 2.108608433271444 \cdot 10^{-6} \\
S-1 & \rightarrow \text{VP-5 VP-3} & 1.6367163259885093 \cdot 10^{-4} \\
S-2 & \rightarrow \text{VP-5 VP-3} & 9.724998692152419 \cdot 10^{-8} \\
S-1 & \rightarrow \text{PP-7 VP-0} & 1.0686659961009547 \cdot 10^{-5} \\
S-9 & \rightarrow \text{“ NP-3} & 0.012551243773149695 \cdot 10^0 \\
\end{align*}
\]

\[\Rightarrow \text{Regular tree grammar}\]
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 TSGIv
- it is probably the intricate 3-layer back-off model
Overview

1. Motivation

2. Regular Tree Grammars

3. Theoretical Properties

4. Excursion
Tree Automaton

Definition (THATCHER, 1970; ROUNDS, 1970)

A tree automaton is a regular tree grammar in normal form.

Remarks
- **bottom-up**: rules written as $X(q_1, \ldots, q_k) \rightarrow q$
- **top-down**: rules written as $q \rightarrow X(q_1, \ldots, q_k)$
Tree Automaton

Definition (THATCHER, 1970; ROUND, 1970)

A tree automaton is a regular tree grammar in normal form.

Remarks

- **bottom-up**: rules written as $X(q_1, \ldots, q_k) \rightarrow q$
- **top-down**: rules written as $q \rightarrow X(q_1, \ldots, q_k)$
## Determinism

### Definition

- **top-down deterministic** if \( \forall q \in Q, k \in \mathbb{N}, X \in \Sigma \) there exists at most one \( q_1, \ldots, q_k \in Q \): \( q \rightarrow X(q_1, \ldots, q_k) \in P \)

- **bottom-up deterministic** if \( \forall k \in \mathbb{N}, X \in \Sigma, q_1, \ldots, q_k \in Q \) there exists at most one \( q \in Q \): \( X(q_1, \ldots, q_k) \rightarrow q \in P \)

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

**top-down deterministic \( \subset \) bottom-up deterministic = RTL**

### Proof.

By a standard subset construction and a simple counterexample \( \square \)
Determinism

Remark

finite tree languages $\not\subset$ top-down deterministic
Determinism

Remark
finite tree languages \( \not\subseteq \) top-down deterministic
Determinism

Remark
finite tree languages $\not\subseteq$ top-down deterministic
Theoretical Properties

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
Theoretical Properties

Operations on Regular Tree Languages

Theorem

Regular tree languages are closed under substitution

Definition

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

$L[X \leftarrow L']$

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

Operations on Regular Tree Languages

**Theorem**

Regular tree languages are closed under substitution

\[ L[X \leftarrow L'] \]

- \( t \in L \)
- \( t_1, t_2, t_3 \in L' \)
Efficient Representation

Definition
A tree automaton is **minimal** in \( C \) if all equivalent tree automata of \( C \) are at least as large.

Theorem

Complexity of minimization problems:

<table>
<thead>
<tr>
<th>outp. \ inp. model</th>
<th>DTA</th>
<th>NTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C = \text{DTA} )</td>
<td>NL</td>
<td>(EXPTIME)</td>
</tr>
<tr>
<td>( C = \text{NTA} )</td>
<td>PSPACE</td>
<td>PSPACE</td>
</tr>
</tbody>
</table>
Overview

1. Motivation

2. Regular Tree Grammars

3. Theoretical Properties

4. Excursion
Weighted Tree Automaton

**Definition (BERSTEL, REUTENAUER, 1982)**

A weighted tree automaton is a tree automaton together with a map $c: P \rightarrow S$

**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 $c : P \to S$.

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 \( c: P \to S \)

**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 \( c : P \rightarrow S \)

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

<table>
<thead>
<tr>
<th></th>
<th>states</th>
<th>productions</th>
</tr>
</thead>
<tbody>
<tr>
<td>English grammar</td>
<td>1,133</td>
<td>1,842,218</td>
</tr>
<tr>
<td>backward minimal</td>
<td>548</td>
<td>626,600</td>
</tr>
<tr>
<td>forward minimal</td>
<td>791</td>
<td>767,153</td>
</tr>
<tr>
<td>backward/forward minimal</td>
<td>366</td>
<td>272,675</td>
</tr>
<tr>
<td>forward/backward minimal</td>
<td>381</td>
<td>309,845</td>
</tr>
<tr>
<td>f/b/f/b minimal</td>
<td>375</td>
<td>295,836</td>
</tr>
</tbody>
</table>

These might be buggy
### Bisimulation Minimization

(needs additive cancellation)

**Experiment with BERKELEY parser**

<table>
<thead>
<tr>
<th></th>
<th>states</th>
<th>productions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>English grammar</strong></td>
<td>1,133</td>
<td>1,842,218</td>
</tr>
<tr>
<td>backward minimal</td>
<td>548</td>
<td>626,600</td>
</tr>
<tr>
<td>forward minimal</td>
<td>791</td>
<td>767,153</td>
</tr>
<tr>
<td>backward/forward minimal</td>
<td>366</td>
<td>272,675</td>
</tr>
<tr>
<td>forward/backward minimal</td>
<td>381</td>
<td>309,845</td>
</tr>
<tr>
<td>f/b/f/b minimal</td>
<td>375</td>
<td>295,836</td>
</tr>
</tbody>
</table>

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
- ...

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
Quick Recall

From Automata to Transducers

Idea

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

Example

- join two productions $q_1 \rightarrow r_1$ and $q_2 \rightarrow r_2$ to $(q_1, q_2) \rightarrow (r_1, r_2)$
- 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 $q_1 \rightarrow r_1$ and $q_2 \rightarrow r_2$ to $(q_1, q_2) \rightarrow (r_1, r_2)$
- Demand $q_1 = q = q_2$ for simplicity and write $r_1 \overset{q}{\rightarrow} r_2$
- Productions develop input and output trees at the same time.
Quick Recall

From Automata to Transducers

Idea

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

Example

- join two productions $q_1 \rightarrow r_1$ and $q_2 \rightarrow r_2$ to \((q_1, q_2) \rightarrow (r_1, r_2)\)
- 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
Quick Recall

From Automata to Transducers

Idea

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

Example

- join two productions $q_1 \rightarrow r_1$ and $q_2 \rightarrow r_2$ to $(q_1, q_2) \rightarrow (r_1, r_2)$
- 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

Used rule: \[ q \]

Next rule: \[
\begin{array}{c}
S \\
\downarrow \\
CONJ \\
\downarrow \\
wa \\
\end{array}
\]

\[ q \xrightarrow{q} q \]
From Automata to Transducers

Used rule:

Next rule:
Quick Recall

From Automata to Transducers

Used rule:

Next rule:

Trees Abound — Part II: Tree Transducers

A. Maletti
Quick Recall

From Automata to Transducers

Used rule:

```
  V  p  V
saw  ra’aa
```

Next rule:

```
    NP
   /   r
  DT   r
 the
```

Trees Abound — Part II: Tree Transducers
Quick Recall

From Automata to Transducers

Used rule:

Next rule:

Trees Abound — Part II: Tree Transducers

A. Maletti
From Automata to Transducers

**Used rule:**

```
  N  r
boy ateel
```

**Next rule:**

```
  NP r
  DT the
  q2
```

**Diagram:**

- S
  - NP
    - DT the
    - N boy
  - VP
    - V saw
- S
  - CONJ wa
  - S
    - V ra’aa
    - NP
      - N ateel
From Automata to Transducers

Used rule:

Next rule:

Trees Abound — Part II: Tree Transducers

A. Maletti
Used rule:

```
  N  r  N
  door  r  albab
```

Next rule:

```
  S
  \[ CONJ \]
  wa
  V
  ra’aa
  NP
  ateel
  NP
  N
  door
  NP
  N
  albab
```
Quick Recall

From Automata to Transducers

Remarks

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

Output model: RTG and input model:

- NTA $\xrightarrow{}$ linear top-down tree transducer
- RTG $\xrightarrow{}$ 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:

\[
\text{VP} \quad q_1 \quad q_2 \quad q_3 \\
q_2 \quad \text{VP} \\
q_1 \quad q_3
\]

Top-down tree transducer rule:

\[
\text{VP} \quad x_1 \quad x_2 \quad x_3 \\
q \quad \text{VP} \\
q_2 \quad \text{VP} \\
q_1 \quad q_3
\]
Definition (THATCHER, 1970)

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

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

\[
\begin{align*}
q & \rightarrow q \\
\gamma & \rightarrow q \\
\gamma x_1 & \rightarrow q x_1 \\
\sigma & \rightarrow q \sigma q \\
\sigma x_1 & \rightarrow q x_2 q x_1 \\
\alpha & \rightarrow \alpha
\end{align*}
\]
Top-down Tree Transducer

Definition

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

If there exist $\ell \to r \in R$, position $w \in \text{pos}(\xi)$, substitution $\theta : X \to T_\Sigma$

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

Example
Derivation Semantics

Definition

\[ M = \{ \langle t, u \rangle \in T_\Sigma \times T_\Delta \mid \exists q \in I: q(t) \Rightarrow^*_M u \} \]
Derivation Semantics

Definition

\[ M = \{ \langle t, u \rangle \in T_\Sigma \times T_\Delta \mid \exists q \in I: q(t) \Rightarrow^*_M u \} \]

Example

Top-down tree transducer \( N \) with

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

\[
\begin{align*}
q & \xrightarrow{\gamma} q \\
\gamma & \xrightarrow{x_1} q \\
\end{align*}
\]

\[
\begin{align*}
q & \xrightarrow{\sigma} q \\
x_1 & \xrightarrow{\sigma} x_1 \\
x_2 & \xrightarrow{\sigma} x_2 \\
\end{align*}
\]

\[
\begin{align*}
q & \xrightarrow{\alpha} \alpha \\
\end{align*}
\]
Syntactic Restrictions

Definition

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

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

Definition

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

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

Example

Mirror-image transducer is **linear**, **nondeleting**, and **strict** (Ins-TOP)
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
⇝ 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
⇝ 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)

\[ \tau \subseteq T_\Sigma \times T_\Delta \text{ and } \tau' \subseteq T_\Delta \times T_\Gamma \]

\[ \tau ; \tau' = \{(s, u) \mid \exists t \in T_\Delta : (s, t) \in \tau, (t, u) \in \tau'\} \]

Example (Double mirror-image)

\[ N ; N = \text{id} \]

Trees Abound — Part II: Tree Transducers
Expressive Power

composition closure indicated in subscript
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?

\[ \tau(L) = \{ u \mid \exists t \in L: (t, u) \in \tau \} \]
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?

\[ \tau(L) = \{ u \mid \exists t \in L : (t, u) \in \tau \} \]
## Summary

<table>
<thead>
<tr>
<th>Model \ Criterion</th>
<th>ROT</th>
<th>SYM</th>
<th>PRES</th>
<th>PRES$^{-1}$</th>
<th>COMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ins-TOP</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>In-TOP</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Is-TOP</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×₂</td>
</tr>
<tr>
<td>I-TOP</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>×₂</td>
</tr>
<tr>
<td>Is-TOP$^R$</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>I-TOP$^R$</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TOP</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>×$\infty$</td>
</tr>
<tr>
<td>TOP$^R$</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>×$\infty$</td>
</tr>
</tbody>
</table>

(SYM = symmetric)
Overview

1. Quick Recall

2. Top-down Tree Transducers

3. Extended Top-down Tree Transducers

4. Extended Multi Bottom-up Tree Transducers
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 $\Sigma$; output alphabet $\Delta$
- initial states $I \subseteq Q$
- finite set of rules $R \subseteq Q(\Sigma(X)) \times T_\Delta(Q(X))$ such that $\text{var}(r) \subseteq \text{var}(\ell)$ and $\ell$ 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 $\Sigma$; output alphabet $\Delta$
- 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 $\ell$ is linear for all $(\ell, r) \in R$
Extended Top-down Tree Transducer

Example

Extended Top-down Tree Transducer

Example

Extended Top-down Tree Transducer

Example
Extended Top-down Tree Transducer

Definition

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

\[\xi \Rightarrow_M \zeta\]

if there exist $\ell \rightarrow r \in R$, position $w \in \text{pos}(\xi)$, substitution $\theta: X \rightarrow T_\Sigma$

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

Definition

\[M = \{\langle t, u \rangle \in T_\Sigma \times T_\Delta \mid \exists q \in I: q(t) \Rightarrow_M^* u\}\]
Extended Top-down Tree Transducer

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

\[ \xi \Rightarrow_M \zeta \]

if there exist $\ell \rightarrow r \in R$, position $w \in \text{pos}(\xi)$, substitution $\theta: X \rightarrow T_\Sigma$

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

Definition

\[ M = \{ \langle t, u \rangle \in T_\Sigma \times T_{\Delta} \mid \exists q \in I: q(t) \Rightarrow^*_M u \} \]
Derivation Example

Example

Trees Abound — Part II: Tree Transducers
Simulation by Copying and Deletion

Example

Trees Abound — Part II: Tree Transducers
Extended Top-down Tree Transducers

Syntactic Restrictions

Definition

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

- **linear, nondeleting, strict** as before
- $\varepsilon$-free if $\ell \notin Q(X)$ for every $\ell \rightarrow r \in R$
Syntactic Restrictions

Definition

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

- **linear, nondeleting, strict** as before
- **$\varepsilon$-free** if $\ell \notin Q(X)$ for every $\ell \rightarrow r \in R$
Syntactic Restrictions

Definition

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

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

Example

Our example transducer is **linear, nondeleting, strict, and $\varepsilon$-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

<table>
<thead>
<tr>
<th>Model \ Criterion</th>
<th>ROT</th>
<th>SYM</th>
<th>PRES</th>
<th>PRES$^{-1}$</th>
<th>COMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ln-TOP</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>I-TOP</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗$^{2}$</td>
</tr>
<tr>
<td>I-TOP$^{R}$</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TOP$^{R}$</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗$^{∞}$</td>
</tr>
<tr>
<td>Ins$^{ε}$-XTOP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗$^{2}$</td>
</tr>
<tr>
<td>Ins-XTOP</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗$^{∞}$</td>
</tr>
<tr>
<td>ls$^{ε}$-XTOP$^{(R)}$</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗$^{2}$</td>
</tr>
<tr>
<td>l$^{ε}$-XTOP</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗$^{4}$</td>
</tr>
<tr>
<td>l$^{ε}$-XTOP$^{R}$</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗$^{3}$</td>
</tr>
<tr>
<td>(s)l$^{ε}$-XTOP$^{(R)}$</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗$^{∞}$</td>
</tr>
<tr>
<td>XTOP</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗$^{∞}$</td>
</tr>
<tr>
<td>XTOP$^{R}$</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗$^{∞}$</td>
</tr>
</tbody>
</table>
Overview

1. Quick Recall

2. Top-down Tree Transducers

3. Extended Top-down Tree Transducers

4. Extended Multi Bottom-up Tree Transducers
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 $\Sigma$; output alphabet $\Delta$
- 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 $\ell$ is linear for all $(\ell, r) \in R$

Properties

- linear, nondeleting, strict, $\varepsilon$-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 $\Sigma$; output alphabet $\Delta$
- 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 $\ell$ is linear for all $(\ell, r) \in R$

Properties

linear, nondeleting, strict, $\varepsilon$-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:

\[
\begin{align*}
e &\rightarrow q \\
q &\rightarrow a \\
a &\rightarrow q \\
a &\rightarrow a \\
b &\rightarrow q \\
b &\rightarrow b \\
b &\rightarrow b \\
q &\rightarrow f \\
x_1 &\rightarrow x_1 \\
x_2 &\rightarrow x_2 \\
x_1 &\rightarrow x_1 \\
x_2 &\rightarrow x_2 \\
f &\rightarrow \sigma \\
x_1 &\rightarrow x_1 \\
x_2 &\rightarrow x_2
\end{align*}
\]

Properties

linear, nondeleting, strict, and \(\epsilon\)-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:

\[
\begin{align*}
e & \rightarrow q \\
q & \rightarrow a \qquad \rightarrow b \\
a & \rightarrow q \quad \quad \quad \rightarrow q \\
a & \rightarrow q \quad \quad \quad \rightarrow q \\
b & \rightarrow q \quad \quad \quad \rightarrow q \\
b & \rightarrow q \quad \quad \quad \rightarrow q \\
\end{align*}
\]

Properties

linear, nondeleting, strict, and \(\varepsilon\)-free
Extended Multi Bottom-up Tree Transducer

Rule:

\[
\begin{array}{c}
\sigma \\
q \\
p \\
x_3 \\
x_4 \\
\rightarrow \\
q \\
\delta \\
x_1 \\
x_2 \\
x_4 \\
x_3 \\
\end{array}
\]

Derivation:

\[
\begin{array}{c}
t \\
\sigma \\
q \\
p \\
x_3 \\
x_4 \\
\Rightarrow M \\
t \\
q \\
\delta \\
u_1 \\
u_2 \\
u_4 \\
u_3 \\
\end{array}
\]
Extended Multi Bottom-up Tree Transducer

Example (Derivation)
Extended Multi Bottom-up Tree Transducer

Example (Derivation)
Extended Multi Bottom-up Tree Transducer

\[ e \rightarrow q \]
\[ \quad \rightarrow a \quad \rightarrow b \]
\[ \quad \quad \rightarrow a \quad \rightarrow b \]
\[ \quad x_1 \quad x_2 \quad x_1 \quad x_2 \quad x_1 \quad x_2 \]

Example (Derivation)

\[ a \rightarrow b \quad \rightarrow a \quad \rightarrow b \]
\[ b \rightarrow M \quad b \rightarrow M \quad q \rightarrow M \]
\[ b \quad q \quad b \quad b \]
\[ e \quad e \quad e \quad e \]
Extended Multi Bottom-up Tree Transducers

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

\[ \tau_M = \{(t, u) \in T_\Sigma \times T_\Delta \mid \exists q \in F : t \Rightarrow^*_M q(u)\} \]
Extended Multi Bottom-up Tree Transducer

Definition

\[ \tau_M = \{(t, u) \in T_\Sigma \times T_\Delta \mid \exists q \in F : t \Rightarrow^*_M q(u)\} \]

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 \(\ell \rightarrow r \in R\)
- bottom-up tree transducer if both previous conditions hold

Example (Duplication)

\[
e \rightarrow \quad q \\
\quad e \\
x_1 \quad q \\
x_2 \\

a \rightarrow \quad q \\
\quad a \\
x_1 \quad a \\
x_2 \\

b \rightarrow \quad q \\
\quad b \\
x_1 \quad b \\
x_2 \\

x_1 \quad x_2 \\

f \rightarrow \quad \sigma \\
x_1 \\
x_2
\]
Expressive Power

Theorem (ENGELFRIET et al. ’09)

\[ \text{I-XTOP}^R = \text{I-XBOT} \]

Proof.

Standard construction trading input-deletion for output-deletion
see \( \text{I-TOP} \subseteq \text{I-BOT} \) by [ENGELFRIET ’75]
Expressive Power

Theorem (ENGELFRIET et al. ’09)

\[ I-XTOP^R = I-XBOT \]

Proof.

Standard construction trading input-deletion for output-deletion see \( I-TOP \subseteq I-BOT \) by [ENGELFRIET ’75]
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)

\[ \text{XMBOT} = \text{n-XMBOT} \]

Proof.

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

**Theorem (ENGELFRIET et al. '09)**

\[ \text{XMBOT} = \text{n-XMBOT} \]

**Proof.**

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

Theorem (ENGELFRIET et al. ’09)

\[ \text{XMBOT} = \text{n-XMBOT} \]

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 $$\varepsilon$$-rules
Expressive Power

Theorem (ENGELFRIET et al. ’09)

$$\varepsilon$$-XMBOT = MBOT

Proof.

- decompose large left-hand sides using “multi”-states
- attach finite effect of $$\varepsilon$$-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 $$\varepsilon$$-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 \( \varepsilon \)-rules
Expressive Power

Definition

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

Theorem (MALETTI ’12)

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

Proof.

- use (essentially) construction of [ENGELFRIET, MANETH ’03]
- obtain finitely copying $\varepsilon$-XTOP
- apply [ENGELFRIET et al. ’09] to obtain $l\varepsilon$-XMBOT
- previous theorems yield ln-MBOT
Expressive Power

Definition

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

Theorem (M\textsc{aletti} ‘12)

sensible XTOP $\subseteq$ ln-MBOT

Proof.

- use (essentially) construction of [Engelfriet, Maneth ’03]
- obtain finitely copying $\varepsilon$-XTOP
- apply [Engelfriet et al. ’09] to obtain $I\varepsilon$-XMBOT
- previous theorems yield ln-MBOT
Expressive Power

Definition

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

Theorem (MALETTI ’12)

sensible XTOP $\subseteq$ ln-MBOT

Proof.

- use (essentially) construction of [ENGELFRIET, MANETH ’03]
- obtain finitely copying $\varepsilon$-XTOP
- apply [ENGELFRIET et al. ’09] to obtain $l\varepsilon$-XMBOT
- previous theorems yield ln-MBOT
Corollary

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

Theorem

\[ \text{In-MBOT} \not\subseteq \text{XTOP}^R \]

Theorem (GILDEA ’12)

\[ \text{yield}(\text{In-MBOT}) = \text{LCFRS} \]

\[ \text{out} \]
Further Properties

Theorem

\[ \text{In-MBOT} \not\subseteq \text{XTOP}^R \]

Theorem (Gildea ’12)

\[ \text{yield} (\text{In-MBOT}) = \text{LCFRS} \]

\[ \text{out} \]
## Summary

<table>
<thead>
<tr>
<th>Model \ Criterion</th>
<th>ROT</th>
<th>SYM</th>
<th>PRES</th>
<th>PRES⁻¹</th>
<th>COMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ln-TOP</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>l-TOP</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x₂</td>
</tr>
<tr>
<td>l-TOP^R</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TOP^R</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x∞</td>
</tr>
<tr>
<td>lnε-XTOP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x₂</td>
</tr>
<tr>
<td>ln-XTOP</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x∞</td>
</tr>
<tr>
<td>lsε-XTOP^R</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x₂</td>
</tr>
<tr>
<td>lε-XTOP</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x₄</td>
</tr>
<tr>
<td>lε-XTOP^R</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x₃</td>
</tr>
<tr>
<td>(s)l-XTOP^R</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x∞</td>
</tr>
<tr>
<td>XTOP^R</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x∞</td>
</tr>
<tr>
<td>ln(n)-XMBOT</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XMBOT</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x∞</td>
</tr>
<tr>
<td>reg.-preserving l-XMBOT</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>invertable l-XMBOT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>