Weighted Multi Bottom-up Tree Transducers

Andreas Maletti

Universitat Rovira i Virgili
Tarragona, Spain

andreas.maletti@urv.cat

Dagstuhl — January 21, 2010
Synchronous Tree Substitution Grammars

Weight: 1
Synchronous Tree Substitution Grammars

Weight: 1 \cdot 0.5
Synchronous Tree Substitution Grammars

Weight: $1 \cdot 0.5 \cdot 0.25$
Synchronous Tree Substitution Grammars

Weight: $1 \cdot 0.5 \cdot 0.25 \cdot 0.03$
Motivation

Synchronous Tree Substitution Grammars

Weighted Multi Bottom-up Tree Transducers

Weight:  \( 1 \cdot 0.5 \cdot 0.25 \cdot 0.03 \cdot 0.25 \)
Synchronous Tree Substitution Grammars

Weighted Multi Bottom-up Tree Transducers

Weight: $1 \cdot 0.5 \cdot 0.25 \cdot 0.03 \cdot 0.25 \cdot 0.1$
Synchronous Tree Substitution Grammars

Weight: \[1 \cdot 0.5 \cdot 0.25 \cdot 0.03 \cdot 0.25 \cdot 0.1 \cdot 0.25\]
Synchronous Tree Substitution Grammars

Weight: $1 \cdot 0.5 \cdot 0.25 \cdot 0.03 \cdot 0.25 \cdot 0.1 \cdot 0.25 \cdot 0.05$
Synchronous Tree Substitution Grammars

Weight: 1 · 0.5 · 0.25 · 0.03 · 0.25 · 0.1 · 0.25 · 0.05

Note

Popular model in machine translation.
Synchronous Tree Substitution Grammars (cont’d)

Advantages
- simple and natural model
- easy to train (from linguistic resources)
- symmetric

(Obvious) Disadvantages
- computes joint-probability (→ generative story)
- no state behavior (→ local behavior)

Implementation
- extended top-down tree transducer in Tiburon
  [May, Knight ’06]
Synchronous Tree Substitution Grammars (cont’d)

Advantages
- simple and natural model
- easy to train (from linguistic resources)
- symmetric

(Obvious) Disadvantages
- computes joint-probability (→ generative story)
- no state behavior (→ local behavior)

Implementation
- extended top-down tree transducer in Tiburon
  [May, Knight ’06]
Synchronous Tree Substitution Grammars (cont’d)

Advantages
- simple and natural model
- easy to train (from linguistic resources)
- symmetric

(Obvious) Disadvantages
- computes joint-probability (→ generative story)
- no state behavior (→ local behavior)

Implementation
- extended top-down tree transducer in Tiburon
  [May, Knight ’06]
Synchronous Tree Substitution Grammars (cont’d)

Synchronous tree substitution grammar rule:

```
S               S
  NP^1          S
    VP           V
      NP^2      NP^1
        V        NP^2
```

Corresponding extended top-down tree transducer rule:

```
q_S              q_S
  S              S
    x_1          q_V
    VP          q_NP
      x_2        q_NP
      q_NP      q_NP
      x_3
```

Weighted Multi Bottom-up Tree Transducers
Extended Top-down Tree Transducer

Advantages
- input-driven model (can easily compute conditional probability)
- state behavior

Disadvantages (also of STSG)
- not binarizable
  [Aho, Ullman ’72; Zhang, Huang, Gildea, Knight ’06]
- inefficient input/output restriction (Bar-Hillel construction)
  [M., Satta ’10]
- not composable
  [Arnold, Dauchet ’82]
Extended Top-down Tree Transducer

Advantages

- input-driven model (can easily compute conditional probability)
- state behavior

Disadvantages (also of STSG)

- not binarizable
  [Aho, Ullman ’72; Zhang, Huang, Gildea, Knight ’06]
- inefficient input/output restriction (Bar-Hillel construction)
  [M., Satta ’10]
- not composable
  [Arnold, Dauchet ’82]
Extended Bottom-up Tree Transducer

Top-down tree transducer rule:

\[
\begin{array}{c}
q_S \\
S \\
\node{x_1} \\
\node{x_2} \\
\node{x_3} \\
VP \\
\node{x_2} \\
\node{x_1} \\
\node{x_3} \\
q_V \\
q_{NP} \\
q_{NP} \\
S \\
S
\end{array}
\]

Corresponding extended bottom-up tree transducer rule:

\[
\begin{array}{c}
q_{NP} \\
q_V \\
q_{NP} \\
VP \\
\node{x_1} \\
\node{x_2} \\
\node{x_3} \\
q_S \\
S \\
\node{x_2} \\
\node{x_1} \\
\node{x_3} \\
S
\end{array}
\]
Extended Bottom-up Tree Transducer (cont’d)

**Theorem**

*For every STSG we can construct an equivalent extended bottom-up tree transducer in linear time.*

**Question**

Do they have better properties?
Theorem

For every STSG we can construct an equivalent extended bottom-up tree transducer in linear time.

Question

Do they have better properties?
Extended Multi Bottom-up Tree Transducers

Roadmap

1. Motivation
2. Extended Multi Bottom-up Tree Transducers
3. Bar-Hillel Construction
4. Composition Construction
Syntax

Convention

Fix a commutative semiring \((S, +, \cdot, 0, 1)\).

Definition

Weighted extended multi bottom-up tree transducer (XMBOT) is a system \((Q, \Sigma, \Delta, F, R)\) with

- \(Q\) ranked alphabet of states
- \(\Sigma\) and \(\Delta\) ranked alphabets of input and output symbols
- \(F \subseteq Q_1\) final states
- \(R\) finite set of rules \(l \xrightarrow{w} r\) with \(w \in S\), linear \(l \in T_{\Sigma}(Q(X))\), linear \(r \in Q(T_{\Delta}(X))\) such that \(\text{var}(l) = \text{var}(r)\)

Extended Multi Bottom-up Tree Transducers

A. Maletti
Syntax

Convention

Fix a commutative semiring \((S, +, \cdot, 0, 1)\).

Definition

Weighted extended multi bottom-up tree transducer (XMBOT) is a system \((Q, \Sigma, \Delta, F, R)\) with

- \(Q\) ranked alphabet of states
- \(\Sigma\) and \(\Delta\) ranked alphabets of input and output symbols
- \(F \subseteq Q_1\) final states
- \(R\) finite set of rules \(l \xrightarrow{w} r\) with \(w \in S\), linear \(l \in T_{\Sigma}(Q(X))\), linear \(r \in Q(T_{\Delta}(X))\) such that \(\text{var}(l) = \text{var}(r)\)
Syntax (cont’d)

Definition

XMBOT \((Q, \Sigma, \Delta, F, R)\) is proper if \(\{l, r\} \not\subseteq Q(X)\) for every \(l \xrightarrow{w} r \in R\).
Definition

$\text{XMBOT} \ (Q, \Sigma, \Delta, F, R)$ is proper if $\{l, r\} \not\subseteq Q(X)$ for every $l \xrightarrow{w} r \in R$.

Example

Disallowed rule for properness:

\begin{equation*}
\begin{array}{c}
p \\
\leq \\
X_1 & X_2 & X_3
\end{array} \quad \xrightarrow{w} \quad \\
\begin{array}{c}
q \\
\leq \\
X_2 & X_1 & X_3
\end{array}
\end{equation*}
Syntax — An Example

- $Q = \{ f^{(1)}, q^{(2)} \}$ and $F = \{ f \}$
- $\Sigma = \{ a^{(1)}, b^{(1)}, e^{(0)} \}$ and $\Delta = \Sigma \cup \{ \sigma^{(2)} \}$
- the following rules (all with weight 1)

```
\begin{align*}
Q & = \{ f^{(1)}, q^{(2)} \} \text{ and } F = \{ f \} \\
\Sigma & = \{ a^{(1)}, b^{(1)}, e^{(0)} \} \text{ and } \Delta = \Sigma \cup \{ \sigma^{(2)} \} \\
\text{the following rules (all with weight 1)} \\
\end{align*}
```

```
\begin{align*}
q \quad & \rightarrow \quad a \quad a \\
\quad & x_1 \quad x_2 \\
\quad & x_1 \quad x_2 \\
\end{align*}
```

```
\begin{align*}
q \quad & \rightarrow \quad b \quad b \\
\quad & x_1 \quad x_2 \\
\quad & x_1 \quad x_2 \\
\end{align*}
```

```
\begin{align*}
q \quad & \rightarrow \quad e \quad e \\
\quad & x_1 \quad x_2 \\
\quad & x_1 \quad x_2 \\
\end{align*}
```

```
\begin{align*}
q \quad & \rightarrow \quad f \\
\quad & x_1 \quad x_2 \\
\quad & x_1 \quad x_2 \\
\end{align*}
```
Semantics

Extended Multi Bottom-up Tree Transducers

Weighted Multi Bottom-up Tree Transducers
Extended Multi Bottom-up Tree Transducers

Semantics

Weighted Multi Bottom-up Tree Transducers

A. Maletti
### Semantics

- $w t(\xi_1 \xrightarrow{w_1} M \cdots \xrightarrow{w_{n-1}} M \xi_n) = w_1 \cdots w_{n-1}$
- $w t(t, u) = \sum_{q \in F, d: t \xrightarrow{M, q(u)}^* d} w t(d)$
Semantics — An Example

Example

```
a
 b
 b
 e
```

Remark

Properness guarantees well-definedness.
Semantics — An Example

Example

a

b

b

q

e  e  e

Remark

Properness guarantees well-definedness.
Semantics — An Example

Example

Remark

Properness guarantees well-definedness.
Semantics — An Example

Example

\[
\begin{array}{c}
\text{a} \\
\text{b} \\
\text{e}
\end{array} 
\quad \Rightarrow_M 
\begin{array}{c}
\text{a} \\
\text{b} \\
\text{q} \\
\text{e} \quad \text{e}
\end{array} 
\quad \Rightarrow_M 
\begin{array}{c}
\text{a} \\
\text{b} \\
\q \\
\text{b} \quad \text{b}
\end{array} 
\quad \Rightarrow_M 
\begin{array}{c}
\text{a} \\
\text{b} \\
\text{q} \\
\text{b} \quad \text{b}
\end{array} 
\quad \Rightarrow_M 
\begin{array}{c}
\text{a} \\
\text{b} \\
\text{q} \\
\text{e} \quad \text{e}
\end{array}
\]

Remark

Properness guarantees well-definedness.
Semantics — An Example

Example

Remark

Properness guarantees well-definedness.
Semantics — An Example

Example

```
\begin{array}{c}
a
\end{array}
\quad
\begin{array}{c}
b
\end{array}
\quad
\begin{array}{c}
q
\end{array}
\quad
\begin{array}{c}
a
\end{array}
\quad
\begin{array}{c}
\sigma
\end{array}
```

Remark

Properness guarantees well-definedness.
Roadmap

1. Motivation

2. Extended Multi Bottom-up Tree Transducers

3. Bar-Hillel Construction

4. Composition Construction
One-Symbol Normal Form

Definition

XMBOT \((Q, \Sigma, \Delta, F, R)\) is in \textit{one-symbol normal form}\nif exactly one input or output symbol occurs in each rule.

Theorem

For every proper XMBOT there exists an equivalent XMBOT in
\textit{one-symbol normal form}. It can be constructed in linear time.

Corollary

For every proper XMBOT the transition from joint-distribution to
conditional-distribution is linear time.
One-Symbol Normal Form

Definition

$\text{XMBOT} (Q, \Sigma, \Delta, F, R)$ is in one-symbol normal form if exactly one input or output symbol occurs in each rule.

Theorem

For every proper XMBOT there exists an equivalent XMBOT in one-symbol normal form. It can be constructed in linear time.

Corollary

For every proper XMBOT the transition from joint-distribution to conditional-distribution is linear time.
One-Symbol Normal Form

Definition

\( \text{XMBOT} (Q, \Sigma, \Delta, F, R) \) is in one-symbol normal form if exactly one input or output symbol occurs in each rule.

Theorem

For every proper XMBOT there exists an equivalent XMBOT in one-symbol normal form. It can be constructed in linear time.

Corollary

For every proper XMBOT the transition from joint-distribution to conditional-distribution is linear time.
One-Symbol Normal Form (cont’d)

Rule not in one-symbol normal form:

Replacement rules for this rule:
Binarization

Definition
An XMBOT is fully binarized if each rule contains at most 3 states. (≤ 2 in each left-hand side)

Theorem
Every proper XMBOT can be fully binarized in linear time.

Proof.
First binarize the trees in the rules and then transform into one-symbol normal form.
Binarization

**Definition**
An XMBOT is **fully binarized** if each rule contains at most 3 states. (≤ 2 in each left-hand side)

**Theorem**
Every proper XMBOT can be fully binarized in linear time.

**Proof.**
First binarize the trees in the rules and then transform into one-symbol normal form.
Comparison

In general, STSG cannot be binarized, but people try ... [Zhang, Huang, Gildea, Knight ’06; DeNero, Pauls, Klein ’09]
Definition

The input product of a weighted tree transformation $\tau : T_{\Sigma} \times T_{\Delta} \to S$ with a power series $\varphi : \Sigma^* \to S$ is $\tau'(s, t) = \tau(s, t) \cdot \varphi(\text{yd}(s))$. 
Bar-Hillel Construction (cont’d)

Theorem

The input product of an XMBOT \( M \) with a WSA \( S \) can be computed in time \( O(|M| \cdot |S|^3) \).

Note

The output product of an XMBOT \( M \) with a WSA \( S \) can be computed in time \( O(|M| \cdot |S|^{2 \cdot \text{rk}(M) + 2}) \).
Bar-Hillel Construction (cont’d)

**Theorem**

The input product of an XMBOT $M$ with a WSA $S$ can be computed in time $O(|M| \cdot |S|^3)$.

**Note**

The output product of an XMBOT $M$ with a WSA $S$ can be computed in time $O(|M| \cdot |S|^{2rk(M)+2})$. 
Roadmap

1. Motivation

2. Extended Multi Bottom-up Tree Transducers

3. Bar-Hillel Construction

4. Composition Construction
Composition of STSG

Conclusion

STSGs are not composable!
Conclusion

STSGs are not composable!
Composition of STSG

Conclusion

STSGs are not composable!
Composition Construction

Definition

for XMBOT $M = (Q, \Sigma, \Gamma, F, R)$ and $N = (Q', \Gamma, \Delta, G, P)$ construct

$$M ; N = (Q(Q'), \Sigma, \Delta, F(G), R')$$

with three types of rules:

1. **input-consuming rules constructed from input-consuming rules of $R$** (with their weight)
2. **epsilon rules constructed from epsilon-rules of $P$**
3. **epsilon rules constructed from an epsilon rule of $R$ followed by an input consuming rule of $P$ (product of the weights)**
Example

Input consuming rule of $R$ and resulting rule:

\[
\begin{align*}
\sigma & \quad q_1 \quad q_2 \\
q_1 & \quad q_2 \quad w \\
q_1 & \quad q_2 \\
p_1 & \quad p_2 \\
p_1 & \quad p_2 \quad w \\
p_1 & \quad p_2 \\
x_1 & \quad x_2 \\
x_1 & \quad x_2 \\
x_1 & \quad x_2 \\
x_1 & \quad x_2
\end{align*}
\]
Example

Epsilon rule of $P$ and resulting rule:

$$p_1 \xrightarrow{w} \frac{p}{\alpha}$$

Diagram:

```
        q_1
       /   \
    p_1   p_2
   /     |
  x_1   x_2

w -->
```

```
       q_1
      /   \
    p_1   p_2
   /     |
  x_1   x_2
```

Weighted Multi Bottom-up Tree Transducers
Example

Epsilon rule of $R$ and input consuming of $P$ and resulting rule:

$$
q_1 \xrightarrow{w_1} x_1 x_2 \xrightarrow{\gamma} x_2 \xrightarrow{w_2} p_2 \xrightarrow{w_2} x_2
$$

$$
q_1 \xrightarrow{w_1} p_1 p_2 \xrightarrow{w_1} \gamma \xrightarrow{w_2} p_2 \xrightarrow{w_2} p
$$
Note

The constructed XMBOT might be non-proper.

Theorem

For all proper XMBOTs $M$ and $N$ such that

- $M$ has no cyclic input epsilon rules or
- $N$ has no cyclic output epsilon rules,

then there exists a proper XMBOT that computes the composition of the transformations computed by $M$ and $N$. 
Composition construction (cont’d)

Note
The constructed XMBOT might be non-proper.

Theorem
For all proper XMBOTs $M$ and $N$ such that
- $M$ has no cyclic input epsilon rules or
- $N$ has no cyclic output epsilon rules,
then there exists a proper XMBOT that computes the composition of the transformations computed by $M$ and $N$. 
### Summary

<table>
<thead>
<tr>
<th>Algorithm \ Device</th>
<th>STSG</th>
<th>XMBOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binarization</td>
<td>X</td>
<td>$O(</td>
</tr>
<tr>
<td>Input product</td>
<td>$O(</td>
<td>M</td>
</tr>
<tr>
<td>Output product</td>
<td>$O(</td>
<td>M</td>
</tr>
<tr>
<td>Composition</td>
<td>X</td>
<td>$O(</td>
</tr>
</tbody>
</table>
## Summary

<table>
<thead>
<tr>
<th>Algorithm \ Device</th>
<th>STSG</th>
<th>XMBOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binarization</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Input product</td>
<td>$O(</td>
<td>M</td>
</tr>
<tr>
<td>Output product</td>
<td>$O(</td>
<td>M</td>
</tr>
<tr>
<td>Composition</td>
<td>✓</td>
<td>$O(</td>
</tr>
<tr>
<td>Reversal</td>
<td>$O(</td>
<td>M</td>
</tr>
<tr>
<td>Pres. of REC</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
References (1/2)

- **Aho, Ullman**: The theory of parsing, translation, and compiling. Prentice Hall. 1972
- **Chiang, Knight**: An introduction to synchronous grammars. Tutorial at ACL. 2006
- **DeNero, Pauls, Klein**: Asynchronous binarization for synchronous grammars. In ACL, p. 141–144, 2009
- **Gécseg, Steinby**: *Tree Automata*. Akadémiai Kiadó, Budapest 1984
References (2/2)

- **Lilin**: Une généralisation des transducteurs d’états finis d’arbres: les S-transducteurs. Université de Lille. 1978
- **Maletti, Satta**: The Bar-Hillel construction for STSG. in preparation, 2010
- **May, Knight**: Tiburon — a weighted tree automata toolkit. In *CIAA*, LNCS 4094, p. 102–113, 2006

Thank you for your attention!