A Tree Transducer Model for Synchronous Tree-Adjoining Grammars

Andreas Maletti

Universitat Rovira i Virgili
Tarragona, Spain

andreas.maletti@urv.cat

Uppsala, Sweden — July 13, 2010
Synchronous Tree Substitution Grammar

S

CONJ

wa

Used rule

S — CONJ S

wa
Synchronous Tree Substitution Grammar

Used rule
Synchronous Tree Substitution Grammar

Used rule

\[
S \\
NP^1 \quad VP \\
V \quad NP^2 \\
saw
\]

\[
S \\
CONJ \quad S \\
V \quad NP^1 \quad NP^2 \\
w a \quad ra'aa
\]
Synchronous Tree Substitution Grammar

Used rule

A Tree Transducer Model for STAG
Synchronous Tree Substitution Grammar

Used rule

```
N  N
boy  atefl
```
Synchronous Tree Substitution Grammar

Used rule

A Tree Transducer Model for STAG
Synchronous Tree Substitution Grammar

Used rule

N   —   N

door   albab
Synchronous Tree Substitution Grammar (cont’d)

Advantages

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

Implementation

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

Synchronous tree substitution grammar rule:

```
NP^1  VP  NP^2  \[w\]  NP^1  NP^2
```

Corresponding extended top-down tree transducer rule:

```
q_S  S  q_V  q_{NP}  q_{NP}
```

A Tree Transducer Model for STAG

A. Maletti  ·  4
Synchronous Tree-Adjoining Grammar

A Tree Transducer Model for STAG
Synchronous Tree-Adjoining Grammar

```
S
 NP  VP

S
 NP  VP
```

**Used substitution rule**

```
S
 NP  VP

S
 NP  VP
```
Synchronous Tree-Adjoining Grammar

$$\begin{align*}
S & \rightarrow NP^1 \ VP \\
VP & \rightarrow V \ NP^2
\end{align*}$$

Used substitution rule

$$\begin{align*}
VP & \rightarrow V \ NP \\
\_ & \rightarrow \_ \\
VP & \rightarrow V \ NP
\end{align*}$$
Synchronous Tree-Adjoining Grammar

Used substitution rule

likes — aime
Synchronous Tree-Adjoining Grammar

Used substitution rule

A Tree Transducer Model for STAG
Synchronous Tree-Adjoining Grammar

Used substitution rule

\[
\begin{array}{c}
\text{N} \\
\text{candies}
\end{array}
\quad \rightarrow \quad
\begin{array}{c}
\text{N} \\
\text{bonbons}
\end{array}
\]
Synchronous Tree-Adjoining Grammar

Used adjunction rule

A Tree Transducer Model for STAG
Synchronous Tree-Adjoining Grammar

Used substitution rule

\[
\begin{align*}
\text{adj} &\quad \text{adj} \\
\text{red} &\quad \text{rouges}
\end{align*}
\]
Main Question

Theorem

*Every STSG is an STAG.*

Question

*Are they further related?*
Roadmap

1. Motivation
2. Explicit Substitution
3. Synchronous Tree-Adjoining Grammar
4. Main Result
5. Application
First-Order Substitution

Definition

t[ν₁ ← t₁, . . . , νₖ ← tₖ] denotes the result obtained by replacing (in parallel) all occurrences of leaves labelled vᵢ in t by tᵢ.

Example

```
S
 NP  VP
   V  NP
    saw

t
```

```
NP
   DT  N
    the

u
```

```
NP
   DT  N  V
    the  saw

S
 VP
 NP
   DT  N
    the

t[NP ← u]
```
Second-Order Substitution

Example

```
S
  NP  NP
  V   V

[NP ← · ]

NP  VP

S

NP  DT  N

the

saw
```

Explicit substitution

- keep an explicit representation of substitutions in tree
- any number of substitutions allowed at any level
Second-Order Substitution

Example

\[
\cdot [NP \leftarrow \cdot ]
\]

\[
S \rightarrow NP \rightarrow VP \rightarrow NP \rightarrow V \rightarrow NP \rightarrow DT \rightarrow N
\]

\[
saw \rightarrow \text{the}
\]

Evaluation

\[
eval(\cdot [x \leftarrow \cdot ](t, u)) = eval(t)[x \leftarrow eval(u)]
\]

\[
eval(\sigma(t_1, \ldots, t_k)) = \sigma(eval(t_1), \ldots, eval(t_k))
\]
Second-Order Substitution

Example

Evaluation

```
eval([x ← ·](t, u)) = eval(t)[x ← eval(u)]

eval(σ(t₁, ..., tₖ)) = σ(eval(t₁), ..., eval(tₖ))
```
Roadmap

1. Motivation
2. Explicit Substitution
3. Synchronous Tree-Adjoining Grammar
4. Main Result
5. Application
Tree-Adjoining Grammar

**Intuition**

A TAG has two types of rules:
- substitution rules (as in TSG)
- adjunction rules

**Example (Adjunction)**

```
NP  
DT   N
 les bonbons

N* ADJ
 rouges

NP  
DT   N
 les bonbons

ADJ
 rouges
```
Simplifications (see [SHIEBER '06])

- no substitution rules
- adjunction mandatory (if possible)
- each adjunction spot used at most once
- root nodes of auxiliary trees are never adjunction spots

Definition

A TAG is a finite set of

- derived trees (initial trees) and
- auxiliary trees (those containing a starred node)
Tree-Adjoining Grammar (cont’d)

Simplifications (see [SIEBER ’06])

- no substitution rules
- adjunction mandatory (if possible)
- each adjunction spot used at most once
- root nodes of auxiliary trees are never adjunction spots

Definition

A TAG is a finite set of

- derived trees (initial trees) and
- auxiliary trees (those containing a starred node)
Tree-Adjoining Grammar (cont’d)

Example

Derivation

String language

\{ wcw \mid w \in \{a, b\}^* \}
Synchronous Tree-Adjoining Grammar

Example

Initial tree pair

```
S
  T
  c
  -
S
  T
  c
```

Auxiliary tree pair

```
S
  -
  a
  S*
  a
S
  -
  S*
  a
```

Auxiliary tree pair

```
S
  -
  b
  S*
  b
S
  -
  S*
  b
```

Auxiliary tree pair
Example

Synchronous Tree-Adjoining Grammar (cont’d)

String translation

\{(wcw^R, wcw) \mid w \in \{a, b\}^*\}
Roadmap

1. Motivation
2. Explicit Substitution
3. Synchronous Tree-Adjoining Grammar
4. Main Result
5. Application
Simulation

Question

Can we simulate an STAG by some STSG?
Simulation of Adjunction

Example (TAG)

- Initial tree
- Auxiliary tree

Correspondence (TSG)

- Initial tree
- Auxiliary tree

A Tree Transducer Model for STAG

A. Maletti
Simulation of Adjunction (cont’d)

Example

\[ [S \star \leftarrow \cdot] \]

\[ S \]

\[ S \]

\[ T \]

\[ c \Rightarrow \]

\[ S \]

\[ S \]

\[ a \]

\[ [S \star \leftarrow \cdot] \]

\[ S \]

\[ S \]

\[ T \]

\[ c \Rightarrow \]

\[ S \]

\[ S \]

\[ b \]

\[ [S \star \leftarrow \cdot] \]

\[ S \]

\[ S \]

\[ T \]

\[ c \Rightarrow \]

\[ S \]

\[ S \]

\[ a \]

\[ [S \star \leftarrow \cdot] \]

\[ S \]

\[ S \]

\[ b \]

\[ [S \star \leftarrow \cdot] \]
Simulation of Adjunction (cont’d)

TSG result

Evaluation

Note

coincides with the result obtained by TAG
Simulation of Adjunction (cont’d)

**TSG result**

![Tree structure for TSG result]

**Evaluation**

![Tree structure for Evaluation]

Note

coincides with the result obtained by TAG
Simulation of Adjunction (cont’d)

TSG result

Evaluation

Note

coincides with the result obtained by TAG
Main Result

Theorem

For every TAG $G$ there exists a TSG $G'$ such that

$$L(G) = \{\text{eval}(t) \mid t \in L(G')\}$$
Main Result

Theorem

For every TAG $G$ there exists a TSG $G'$ such that

$$L(G) = \{ \text{eval}(t) \mid t \in L(G') \}$$

Theorem

For every STAG $G$ there exists a STSG $G'$ such that

$$T(G) = \{ (\text{eval}(t), \text{eval}(u)) \mid (t, u) \in T(G') \}$$
Main Result

Theorem

For every STAG $G$ there exists a STSG $G'$ such that

$$T(G) = \{(\text{eval}(t), \text{eval}(u)) \mid (t, u) \in T(G')\}$$

Proof.

\[ \text{STAG} \]
Main Result

Theorem

For every STAG $G$ there exists a STSG $G'$ such that

$$T(G) = \{(\text{eval}(t), \text{eval}(u)) \mid (t, u) \in T(G')\}$$

Proof.

![Diagram showing the relationship between STAG and STSG]
Main Result

Theorem
For every STAG $G$ there exists a STSG $G'$ such that

$$T(G) = \{(\text{eval}(t), \text{eval}(u)) \mid (t, u) \in T(G')\}$$

Proof.

A Tree Transducer Model for STAG
Main Result

Theorem
For every STAG $G$ there exists a STSG $G'$ such that

$$T(G) = \{(\text{eval}(t), \text{eval}(u)) \mid (t, u) \in T(G')\}$$

Proof.
Roadmap

1 Motivation

2 Explicit Substitution

3 Synchronous Tree-Adjoining Grammar

4 Main Result

5 Application
Application

Overview

- run an STAG in TIBURON (which can run STSGs)
- translate STSG algorithms to STAGs (factorization, etc.)
- integrate explicit substitution into semantics
- separate "context-free" and "context-sensitive" behavior
References

- **Arnold, Dauchet**: Morphismes et bimorphismes d’arbres. *Theoret. Comput. Sci.* 20. 1982
- **Chiang, Knight**: An introduction to synchronous grammars. Tutorial at ACL. 2006
- **Engelfriet, Vogler**: Macro tree transducers. *J. Comput. Syst. Sci.* 31. 1985
- **May, Knight**: TIBURON — a weighted tree automata toolkit. In CIAA, LNCS 4094. 2006
- **Nederhof**: Weighted parsing of trees. In *IWPT*. 2009
- **Shieber, Schabes**: Synchronous tree-adjoining grammars. *Computational Linguistics* 3. 1990
- **Shieber**: Unifying synchronous tree adjoining grammars and tree transducers via bimorphisms. In *EACL*. 2006

Thank you for your attention!