Synchronous Grammars

David Chiang

TAG+11

26 Sep 2012
Synchronous grammars are a way of simultaneously generating pairs of recursively related strings (or trees)
Synchronous grammars

were originally invented for programming language compilation

for i := 1 to 10 do
begin
    n := n + i
end

mov ax, 1
loop: add bx, ax
cmp ax, 10
jle loop
Synchronous grammars have been used for syntax-based machine translation.

I open the box    watashi wa hako wo akemasu
Synchronous grammars have been proposed as a way of doing semantic interpretation.

$I$ open the box \[ open'(me', box') \]
Synchronous grammars can do much fancier transformations than finite-state methods.

The boy stated that the student said that the teacher danced.

shoonen ga gakusei ga sensei ga odotta to itta to hanashita

boy student teacher danced that said that stated
Synchronous grammars can do much fancier transformations than finite-state methods

...that John saw Peter help the children swim

...dat Jan Piet de kinderen zag helpen zwemmen

John    Peter    the children    saw    help    swim
Overview

~ Definitions
~ Properties
~ Algorithms
~ Extensions
Definitions
Synchronous CFGs

\[
\begin{align*}
S & \rightarrow \text{NP } \text{VP} \\
\text{NP} & \rightarrow \text{I} \\
\text{NP} & \rightarrow \text{the box} \\
\text{VP} & \rightarrow \text{V } \text{NP} \\
\text{V} & \rightarrow \text{open}
\end{align*}
\]

\[
\begin{align*}
S & \rightarrow \text{NP } \text{VP} \\
\text{NP} & \rightarrow \text{Watashi wa} \\
\text{NP} & \rightarrow \text{Hako wo} \\
\text{VP} & \rightarrow \text{NP } \text{V} \\
\text{V} & \rightarrow \text{akemasu}
\end{align*}
\]
Synchronous CFGs

\[ S \rightarrow \text{NP}_1 \text{VP}_2, \text{NP}_1 \text{VP}_2 \]

\[ \text{NP} \rightarrow \text{I, watashi wa} \]

\[ \text{NP} \rightarrow \text{the box, hako wo} \]

\[ \text{VP} \rightarrow \text{V}_1 \text{NP}_2, \text{NP}_2 \text{V}_1 \]

\[ \text{V} \rightarrow \text{open, akemasu} \]
Synchronous CFGs

S1

NP2    VP3

I      V4       NP5

open   the      box

S1

NP2    VP3

watashi wa NP5

hako wo akemasu
Other notations

\[ VP \rightarrow (V_1 \text{ NP}_2, \text{ NP}_2 V_1) \]

\[ (VP \rightarrow V_1 \text{ NP}_2, VP \rightarrow \text{ NP}_2 V_1) \]

\[ VP \rightarrow \langle V \text{ NP} \rangle \]

Syntax directed translation schema (Aho and Ullman; Lewis and Stearns)

Inversion transduction grammar (Wu)
Limitations of synchronous CFGs

S
  NP
  John
  V
  misses
  NP
  Mary

S
  NP
  Marie
  V
  manque
  PP
    P
    à
    Jean
One solution

S₁

NP₂ misses NP₃
John Mary

S₁

NP₃ manque à NP₂
Marie Jean
Synchronous tree substitution grammars

S
  NP₁  VP
    V    NP₂
misses

S
  NP₂  VP
    V    PP
    P    NP₁
manque à

NP
  John
NP
  Jean

NP
  Mary
NP
  Marie
Synchronous tree substitution grammars

S
  NP₁  VP
    John V NP₂
       misses Mary

S
  NP₂  VP
    Marie V PP NP₁
       manque P à Jean
Limitations of synchronous TSGs

...dat Jan Piet de kinderen zag helpen zwemmen

...that John saw Peter help the children swim

This pattern extends to \( n \) nouns and \( n \) verbs
Limitations of synchronous TSGs

Peter help the children swim
Synchronous TAG

S

S1

NP2
the children

VP3
swim

S1

NP2
de kinderen

VP3

V
zwemmen

V
t
S
NP₂ VP₃
the children V
swim

S
S₁
NP₂ VP₃
Peter V S*
help

S
NP₂ VP₃
de kinderen V t
helpen

S
S₁
NP₂ VP₃
Piet S* V t
helpen
Peter helps the children swim.

Piet helps de kinderen zwemmen.
Peter helps the children swim.

Piet helps de kinderen swim.

John saw Jan zag.

Saw helped the children swim.
John saw Peter help the children swim.

Jan zag Piet helpen de kinderen zwemmen.
Synchronous TAGs & multicomponent TAGs

\[ \sim \text{Synchronous TAG} \]
(Shieber, 1994) \approx
set-local 2-component TAG

\[ \sim \text{Synchronous TAG} \]
(Shieber & Schabes, 1990) \approx
non-local 2-component TAG
Properties
Chomsky normal form

\[ X \rightarrow Y Z \]

\[ X \rightarrow \alpha \]
Chomsky normal form

\[ A \rightarrow B \ C \ D \ E \ F \]

rank 5
Chomsky normal form

\[ A \rightarrow [[[B \ C] \ D] \ E] \ F \]

\[ A \rightarrow V_1 \ F \]

\[ V_1 \rightarrow V_2 \ E \]

\[ V_2 \rightarrow V_3 \ D \]

\[ V_3 \rightarrow B \ C \]
A hierarchy of synchronous CFGs

1-CFG $\subseteq$ 2-CFG = 3-CFG = 4-CFG = …

1-SCFG $\subseteq$ 2-SCFG = 3-SCFG $\subseteq$ 4-SCFG $\subseteq$ …

\[ \text{ITG} \]
(Wu, 1997)
Synchronous CNF?

\[ A \rightarrow (B_1 \ C_2 \ D_3, \ C_2 \ D_3 \ B_1) \quad \text{rank 3} \]
Synchronous CNF?

### Rank 3

\[ A \rightarrow (B_1 [C_2 \ D_3], [C_2 \ D_3] B_1) \]

### Rank 2

\[ A \rightarrow (B_1 \ V_1, \ V_1 B_1) \]
\[ V_1 \rightarrow (C_1 \ D_2, \ C_1 \ D_2) \]
Synchronous CNF?

\[ A \rightarrow (B_1 \ C_2 \ D_3 \ E_4, \ C_2 \ E_4 \ B_1 \ D_3) \]

\[ A \rightarrow ([B_1 \ C_2] \ D_3 \ E_4, \ [C_2 \ E_4 \ B_1] \ D_3) \]

\[ A \rightarrow (B_1 \ [C_2 \ D_3] \ E_4, \ [C_2 \ E_4 \ B_1] \ D_3) \]

\[ A \rightarrow (B_1 \ C_2 \ [D_3 \ E_4], \ C_2 \ [E_4 \ B_1 \ D_3]) \]

rank 4
Synchronous CNF?

\[ A \rightarrow (B_1 \ C_2 \ D_3, \ C_2 \ D_3 \ B_1) \]

\[ A \rightarrow (B_1 \ C_2 \ D_3 \ E_4, \ C_2 \ E_4 \ B_1 \ D_3) \]
Inversion Transduction Grammar

\[ A \rightarrow (B_1 [C_2 [D_3 E_4]], [E_4 D_3] C_2] B_1) \]
A hierarchy of synchronous CFGs

\[ 1\text{-CFG} \preceq 2\text{-CFG} = 3\text{-CFG} = 4\text{-CFG} = \ldots \]

\[ 1\text{-SCFG} \preceq 2\text{-SCFG} = 3\text{-SCFG} \preceq 4\text{-SCFG} \preceq \ldots \]

\[ \Downarrow \quad \Downarrow \quad \Downarrow \]

ITG

(Wu, 1997)
A hierarchy of synchronous TAGs

\[ 1\text{-TAG} \preceq 2\text{-TAG} = 3\text{-TAG} = 4\text{-TAG} = \ldots \]

\[ 1\text{-STAG} \preceq 2\text{-STAG} = 3\text{-STAG} \preceq 4\text{-STAG} \preceq \ldots \]
Algorithms
Overview

- Translation
- Bitext parsing
Review: CKY

S → NP VP
NP → I
NP → the box
VP → V NP
V → open

I open the box
Review: CKY

\[
S \rightarrow NP \ VP \\
NP \rightarrow I \\
NP \rightarrow \text{the box} \\
VP \rightarrow V \ NP \\
V \rightarrow \text{open}
\]
Review: CKY

\[
S \rightarrow \text{NP VP}
\]
\[
\text{NP} \rightarrow \text{I}
\]
\[
\text{NP} \rightarrow \text{the box}
\]
\[
\text{VP} \rightarrow \text{V NP}
\]
\[
\text{V} \rightarrow \text{open}
\]
Review: CKY

\[
S \rightarrow NP \ VP \\
NP \rightarrow I \\
NP \rightarrow \text{the box} \\
VP \rightarrow V \ NP \\
V \rightarrow \text{open}
\]

\[
S
\]

\[
NP \quad VP
\]

\[
I \quad \text{open} \quad \text{the} \quad \text{box}
\]
Review: CKY

S → NP VP
NP → I
NP → the box
VP → V NP
V → open

S
I open the box
Review: CKY

$O(n^3)$ ways of matching

Diagram:

- $VP_4$
- $V_2$
- $NP_3$
- open
- the box
Translation

I open the box

$O(n^3)$
Translation

I open the box

watashi wa hako wo akemasu

I open the box

watashi wa hako wo akemasu
What about…

$\mathcal{O}(n^5)$ ways of combining?
Translation

A

B C D E

V1

V2

B C D E

flatten $O(n)$

translate $O(n)$

parse $O(n^3)$

A

C E D B

49
Bitext parsing

I open the box

watashi wa hako wo akemasu

I open the box

watashi wa hako wo akemasu
Bitext parsing

Consider rank-2 synchronous CFGs for now
Bitext parsing

I open the box

watashi wa hako wo akemasu
Bitext parsing

I open the box

watashi wa hako wo akemasu
Bitext parsing

I open the box

watashi wa hako wo akemasu
Bitext parsing

$\mathcal{O}(n^6)$ ways of matching
Bitext parsing

$\mathcal{O}(n^{10})$ ways of combining!
Summary so far

〜 Translation: essentially parsing on the source side, $\mathcal{O}(n^3)$

〜 Bitext parsing: polynomial in $n$ but worst-case exponential in rank, $\mathcal{O}(n^{2(r+1)})$
Parsing as intersection

~ Translation is like intersecting with a finite-state automaton on source side

~ Bitext parsing is like intersecting with FSAs on both sides
Translation with a language model is also like intersecting with FSAs on both sides.

Input string:

I open the box

$n$-gram language model
Extensions
Synchronous TAGs & multicomponent TAGs

≈ Synchronous TAG (Shieber, 1994) ≈
set-local 2-component TAG
Synchronous TAGs & multicomponent TAGs

\[ \simeq \text{Synchronous set-local } k\text{-component TAG} \]

\[ \simeq \text{set-local } 2k\text{-component TAG} \]
Synchronous TAGs &
multicomponent TAGs

\sim Set-local \ k\text{-component TAG} \colon set-local \ \ k'\text{-component TAG}

\approx set-local \ (k+k')\text{-component TAG}
Synchronous LCFRS

~ rank = “how many things a rule combines”

~ fanout = “how many pieces does each thing have” (CFG = 1, TAG = 2)

~ synchronize any \((r, f)\) and \((r, f')\) LCFRSs. Bitext parsing: \(\mathcal{O}(n^{(r+1)(f+f')})\)
Hyperedge replacement grammars

\[ T \rightarrow \text{instance} \uparrow \text{domain} \downarrow \text{agent} \]
\[ E \rightarrow \text{instance} \uparrow \text{me} \downarrow \]
\[ T \rightarrow \text{instance} \uparrow \text{agent} \downarrow \]

external node
(like a foot node)
Hyperedge replacement grammars


isomorphic copy of the graph formed by removing a graph fragment with XR is a tuple \( (\mathbf{E}, \mathbf{V}, \mathbf{N}) \), where \( \mathbf{E} \) is a distinguished edge, \( \mathbf{V} \) is a set of nodes, and \( \mathbf{N} \) is a set of terminals.

We will also make use of what we call a graph parsing function \( \mathcal{P} \).

The derivation for "I want to see" is:

\[
\begin{align*}
\text{I} & \to \text{want}' \\
\text{E} & \to \text{agent} \\
\text{T} & \to \text{instance} \\
\end{align*}
\]

Likewise, we do not need to store all of those of the input. In CKY, the two endpoints of the input are drawn on top, and the boundary nodes and edges of the recognized part of the input are in.

1. For the reflexive, transitive closure of \( \mathcal{P} \), it suffices to store the boundary nodes and edges of the recognized part of the input.
Hyperedge replacement grammars

\[ T \rightarrow \text{instance} \]

\[ E \]

\[ T \rightarrow \text{agent} \]

\[ \text{instance} \]

\[ \text{see'} \]

\[ E \]

\[ \text{domain} \]

\[ \text{want'} \]
Hyperedge replacement grammars

Assume, for each production \((H, R)\) of the input. In CKY, the two endpoints \(I_1, I_2\) are both labeled with the fix this notation for the reflexive, transitive closure of \(G\). We say that a HRG is a HRG, \(H\) is an edge-induced subgraph of \(\bigcup \{I_i \mid \exists j : (I_i, R, I_j) \in E\}\). If a graph fragment has two external nodes, \(e\) is a distinguished instance of the input. We present each parsing algorithm as a deductive system. We first present an algorithm very similar to that of...
Synchronous HERGs

\[ T \rightarrow \text{want}' \]
\[ E \rightarrow \text{agent} \]
\[ S \rightarrow \text{NP} \rightarrow \text{VP} \rightarrow \text{S} \]

**Example.** Below is a synchronous HRG/TSG:

\[
\begin{align*}
T & \rightarrow E \rightarrow T^2 \\
S & \rightarrow NP \rightarrow VBP \rightarrow S^2 \\
\end{align*}
\]

**4 Extensions**

A number of extensions of HRG and the HRG recognition algorithm immediately follow from the fact that derivations of HRGs are trees and form context-free sets.

4.1 Derivation forests
4.2 Weighted HRGs
We can associate a weight with each production of a HRG, so that the weight of a HRG derivation is the product of the weights of the productions used. If we require that the weights of all productions with the same left-hand-side sum to one, then the derivation weights form a probability distribution (Mosbah, 1996).

4.3 Synchronous HRGs
We can also build synchronous grammars that synchronize HRGs with other HRGs, or HRGs with other formalisms. Of particular interest for linguistics is to synchronize a HRG, modeling semantics, with a TSG or TAG, modeling syntax.

A synchronous HRG/TSG has productions of the form \( h A! R, B! Q, \leftrightarrow i \), where \( (A! R) \) is a HRG production, \( B! Q \) is a TSG production, and \( \leftrightarrow \) is a one-to-one correspondence between the nonterminal occurrences in \( R \) and those of \( Q \). If \( X \leftrightarrow Y \), we draw matching boxed indices (1, 2, ...) next to \( X \) and \( Y \).

References


Summary

~ Synchronous grammars are useful for various tasks: translation, understanding/generation

~ In some ways, they are straightforward (even trivial) extensions of standard formalisms

~ But they can add significant complexity