The Computation of Minimalism

Gordon P. Hemsley

May 7, 2012

LING 667 – Computational Linguistics
Jeff Heinz
Background

Evolution of Syntax

Kobele 2006

Lexical Features and the Lexicon
Structure Derivation

This Project

Goals
Progress
The representation of the structure of language has changed extensively over the last 60 years or so.

In the mid-1990s, Chomsky introduced the idea of Minimalism, which revolves around the principle of economy.

The principle of economy means that the derivation of a sentence must be motivated by the simplest operations over the simplest units of motivation (features).

Kobele 2006 presents a modern representation of minimalism that is easily computable.
### Features Table

<table>
<thead>
<tr>
<th>MERGE</th>
<th>MOVE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Categorial</strong></td>
<td><strong>Selection</strong></td>
</tr>
<tr>
<td>f (C)</td>
<td>=f (S)</td>
</tr>
<tr>
<td>∗ f (C′)</td>
<td>=&gt; f (R)</td>
</tr>
<tr>
<td>f=&gt; (L)</td>
<td></td>
</tr>
</tbody>
</table>
Simple Lexicon

John has arrived.

John:: ∗d -k -q  
arrive::=d v  
-s::perf=> +k +q s  

have::=en perf  
-en::=>prog en  
ε::=>v prog
The **MERGE** and **MOVE** Operations

**MERGE**(α, β)  Combines derivations α and β iff the topmost feature of one is categorial and the topmost feature of the other selects for that category.

**MOVE**(α)  Moves the licensee constituent within α into the specifier position of the licensor head within α.
Derivation Notation

- Derivations use the following format:
  \((\text{SPEC}, \text{HEAD}, \text{COMP}) : \text{features, MOVING SUB-CONSTITUENTS}\)

- Lexemes are just derivations where the \text{SPEC} and \text{COMP} are implicitly \(\epsilon\).
  (This notation is differentiated by the use of ‘::’.)
Sample Derivation

John has arrived.

<table>
<thead>
<tr>
<th>#</th>
<th>Operation</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MERGE(arrive::=d v, John::*d -k -q)</td>
<td>(ε, arrive, ε) : v, John::*k -q</td>
</tr>
<tr>
<td>2</td>
<td>MERGE(ε::=&gt;v prog, 1)</td>
<td>(ε, arrive, ε) : prog, John::*k -q</td>
</tr>
<tr>
<td>3</td>
<td>MERGE(-en::=&gt;prog en, 2)</td>
<td>(ε, arrive -en, ε) : en, John::*k -q</td>
</tr>
<tr>
<td>4</td>
<td>MERGE(have::=en perf, 3)</td>
<td>(ε, have, arrive -en) : perf, John::*k -q</td>
</tr>
<tr>
<td>5</td>
<td>MERGE(-s::perf=&gt; +k +q s, 4)</td>
<td>(ε, ε, have -s arrive -en) : +k +q s, John::*k -q</td>
</tr>
<tr>
<td>6</td>
<td>MOVE(5)</td>
<td>(John, ε, have -s arrive -en) : +q s, John::*q</td>
</tr>
<tr>
<td>7</td>
<td>MOVE(6)</td>
<td>(John, ε, have -s arrive -en) : s</td>
</tr>
</tbody>
</table>
Kobele’s feature set and lexicon evolve as Chapter 2 goes on, with the final lexicon presented in Figure 2.16.

It is not immediately clear how the features he adds to lexemes later in the chapter affect the derivations earlier in the chapter.

My primary goal in this project is to replicate the derivations for all of the sentences mentioned over the course of the chapter.
Non-Goals

- Since I am interested in the syntactic derivation only, implementing the semantics that Kobele introduces in the second half of Chapter 2 is outside the scope of this project.
- Implementing the copying that Kobele discusses in Chapters 3 and 4 proved too ambitious for this project and was subsequently dropped.
In order to replicate the derivations, I first had to implement the features, the lexicon, and the MERGE and MOVE operations.

Most of the derivations presented over the course of Chapter 2 are obsolete with regard to the final lexicon presented in Figure 2.16.

Many lexemes used in examples throughout the chapter—even some used in actual derivations—are not present in that final lexicon.
I have implemented most of the derivations in Section 1 of Chapter 2, plus a few from Section 2. They have all worked out so far, once the lexicon was supplemented properly. Many of the derivations require a seemingly high number of steps. (The 7-step derivation presented here was the shortest one I’ve found.)
Future Work

- To complete this project, I have to finish implementing the derivations from Section 2 of Chapter 2.
- This seemingly will involve modifying my `MERGE` function to handle Kobele's separate operation for control merge.
- The functions seem to be easily extended to the copying Kobele introduces in Chapter 3.
Reference

- http://home.uchicago.edu/~gkobele/diss.html

Implementation

- https://bitbucket.org/gphemsley/kobele2006/