Grammarformalisms

Metagrammar and XMG

Wolfgang Maier

SFB 441

June 25, 2007

Outline

1. LTAG
   - The situation and what's wrong with it
   - The solution: Metagrammars
2. XMG
   - Formalism and Tool
   - How it works

LTAG: The facts

A Lexicalized Tree-Adjoining Grammar consists of

- A finite set of elementary trees, consisting of initial trees and auxiliary trees. Each of the trees is associated with at least one non-empty lexical element, its *anchor*
- Two operations to combine the trees (adjunction and substitution)

Elementary trees contain redundancies

- LTAG disposes only of two operations (adjunction and substitution) to combine trees, eventually restricted by a feature mechanism (FTAG)
- Most of the information that an LTAG bears is contained in the elementary trees
- These are very redundant: many of them contain identical tree fragments (bearing a linguistical value, e.g. *subject*)

No generalization possible

- When creating the grammar, it should be sufficient to write down once any subtree that occurs in two or more elementary trees

but:

- This is not possible: There is no way to access tree fragments directly and incorporate them in different trees
- We can't express the fact that a certain subtree (e.g. expressing the *subject*) is identical across different elementary trees.

$\Downarrow$

Some standard linguistic generalizations are impossible to express!

Cumbersome maintenance

- An LTAG grammar is an unsorted accumulation of trees
- Even small changes to the grammar can involve touching thousands of trees

$\Downarrow$

Grammar maintenance is a time-consuming and error-prone task!

The solution: Metagrammars

- Metagrammars (MGs) introduce an additional layer of abstraction at the level of the elementary trees
- The set of elementary trees is described as a set of abstract tree classes (the *core grammar*).
- Tree classes are instantiated to generate the concrete trees (the *expanded grammar*).

The Advantage of MGs

- With MGs, we can factor out tree fragments we want to use in different trees
- Together with a combination operation, we can use the tree fragments as "building blocks" for the construction of complete trees.

Useful for LTAG

With all trees being accessible through disjunct tree classes, the previously mentioned difficulties are gone.

- *Abstraction*: Writing down **once** a subtree that occurs in various elementary trees (like *subject*) is enough → it can be reused
- *Grammar maintenance*: Instead of chaning the same subtree in thousands of trees, with MGs, changing a single abstract tree class is enough

XMG

- Name of the metagrammar formalism and of the implementation of a metagrammar compiler
- created at LORIA, Nancy, France
- written in Oz/Mozart (`http://www.mozart-oz.org`), available at `http://sourcesup.cru.fr/xmg/`
- Other metagrammar implementations have been created elsewhere, but XMG is the most elaborate one

Combining tree fragments

The elementary tree of a transitive verb can intuitively be expressed as a combination of three tree fragments:

$$TransitiveVerb = Subject \wedge ActiveVerb \wedge Object$$

where

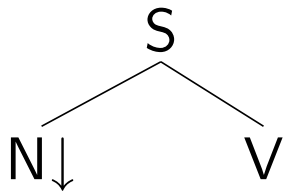
$$Subject = CanonicalSubject \vee RelativeSubject$$

and

$$Object = CanonicalObject \vee WhObject$$

Tree descriptions

The tree fragments of the core grammar are expressed as tree descriptions. *CanonicalSubject*,



is represented by the formula

$$(S \rightarrow N) \wedge (S \rightarrow V) \wedge (N > V)$$

where $\rightarrow$ stands for dominance and $>$ for precedence

Logical programming

Key idea: XMG interprets tree fragments as logic programs.

- Tree fragments (like *CanonicalSubject*) are *facts*
- We can state *rules* that allow or forbid the combination of *facts* (e.g. $Subject = CanonicalSubject \vee RelativeSubject$)
- We can issue *queries* to ask for combinations to take place.