A new approach to control and raising

Ralph Debusmann
Programming Systems Lab
Saarland University

Research program

- To get us from words to what they mean (in a logical formalism, e.g. CLLS)
- Requirements:
  - Simplicity
  - Concurrency
- To this end: need an appropriate grammar formalism

Some grammar formalisms

GB (Chomsky 86)  
HPSG (Pollard/Sag 94)  
FGD (Sgall et al 86)  
LFG (Bresnan/Kaplan 82)  
MTT (Melcuk 88)  
TAG (Joshi 87)  
CCG (Ades/Steedman 82)

- So plenty of them already exist. Why not simply pick one of them?

Problems of existing grammar formalisms

- Tend to conflate levels of representation
  - Syntactic function
  - Word order
  - Predicate-argument structure
- Lack of language-independence (esp. Problems with free word order languages)
- Lack of simplicity (matter of taste)

Solving these problems

- Claim: dependency-based formalisms have the prerequisites to solve these problems:
  - Levels of representation distinguished more properly
  - Less problems with free word order languages (in fact FGD and MTT developed for Czech/Russian)
  - Simpler
- But do they really?

Unfortunately not

- Dependency-based formalisms like FGD and MTT have serious problems:
  - Lack of declarativity (esp. when it comes to handling word order)
  - No (concurrent) syntax-semantics interface
From PS to DG

- Current dependency-based grammar formalisms cannot solve the problems of PS-based ones
- One idea: equip PS-based grammar formalisms with ideas from dependency-based ones

In fact: PS-based grammar formalisms have picked up more and more ideas from dependency-based ones:
- GB: X-bar theory
- HPSG: DEPS-feature in new versions
- f-structure
- Derivation tree represents dependency-like structure

PS and MS-DOS

- But why shall we keep the PS-backbone at all (like hanging on to MS-DOS)?
- E.g. TAG: why do we need the derived tree when all we need for semantics construction is encoded in the derivation tree?
- What we do: develop a new dependency-based grammar formalism: Topological Dependency Grammar (TDG)

TDG solves some problems

- Lack of declarativity (esp. when it comes to handling word order):
  - TDG 2001: Debusmann 2001 MSc, Duchier/Debusmann 2001 ACL
- No (concurrent) syntax-semantics interface:
  - TDG 2002+: PhD research, Korthals/Debusmann 2002 COLING
- Today: introduce the main building block of the interface: the argument structure level

TDG architecture

- Multiple clearly separated levels:
  - Dependency tree
  - Topology tree
  - Argument-structure
- Constraint-based, concurrent
- Well-formedness conditions:
  - Shape vs. lexicalized constraints
  - Within vs. across constraints

Dependency tree level

- Basic assumptions:
  - 1:1 mapping from word occurrences to nodes
  - Labeled directed edges (labels: syntactic functions)
Well-formedness conditions:

Shape constraints

- General constraints on the shape of the structure
- E.g., the dependency tree must be a tree (in the graph-theoretical sense), and the argument structure must be a DAG

Lexicalized constraints

- Each node is assigned a lexical entry
- The lexical entry contains lexical attributes and their values
- Lexical ambiguity dealt with nicely by the Selection Constraint (Duchier 99)
- Lexicalized constraints: make statements about the lexical attributes

Example lexical entries

```
trix = in  
  top ...  
  arg ...
```

```
sleep = in  
  top ...  
  arg ...
```

Lexicalized constraints: The out-constraint

- Determines number and type of outgoing edges:

  \[ l \in \text{out}(v) \text{ then } 1 \text{ outgoing edge labeled } l \]

  \[ l \notin \text{out}(v) \text{ then } 0 \text{ outgoing edges labeled } l \]

Lexicalized constraints: The in-constraint

- Determines type of incoming edge:

  \[ v \overset{l} \rightarrow v' \text{ only if } l \in \text{in}(v') \]

In and out-constraints: example
Within vs. across-constraints

- So far: well-formedness conditions for the three levels kept separate: only within-level constraints
- How can we establish a mutual relationship between the levels?

Across-constraints

- Topology/Dependency
  - Flattening
  - Barriers
- Argument Structure/Dependency
  - Linking

Levels of representation

The argument structure-level

- Dependency trees are already close to semantic argument structure, but not close enough
- E.g. control:
  - Who is the sleeper?
The argument structure-level

- We introduce the argument structure-level to represent the argument structure information:

```
  Peter tries to sleep  Peter tries to sleep
```

Linking the argument structure to the dependency tree

- How do the argument structure and the dependency tree relate to each other?
- Idea: semantic arguments are realized by syntactic functions (e.g. the tryer is realized by the subject)

The link-feature

- The link-feature describes a function from semantic arguments to sets of syntactic functions which realize them, e.g.:

```

\[
\begin{array}{l}
  \text{trier} = \\
  \quad \text{to} \quad \text{in} \\
  \quad \text{try} \quad \text{out} \\
  \quad \text{in} \quad \text{\{subj,try\}} \\
  \quad \text{in} \quad \text{\{\}} \\
  \quad \text{subject} \quad \text{\{tryer,trial\}} \\
  \quad \text{link} \quad \text{\{tryer \rightarrow \text{subj,trial} \rightarrow \text{\{vaid\}}\}} \\
\end{array}
\]

```

Linking constraint (first version)

- Semantic arguments may only be realized by appropriate syntactic functions:

```
\[ v \xrightarrow{n} v' \text{ only if } v \xrightarrow{\varphi} v' \land \varphi \in \text{link}(v) \]
```

Control

- The linking principle given does not license the following analysis:

```
  Peter tries to sleep  Peter tries to sleep
```

- There is no edge corresponding to the sleeper-edge in the dependency tree

What happened?

- Sleep does not have a subject in the dependency tree but has a sleeper-argument in the argument structure
- Tries offers its subject for the embedded verb to take
- Sleep takes the subject of tries as ist subject and realizes the sleeper therewith
The offer-feature

- The offer-feature is a set of syntactic functions offered by a control-verb (for embedded verbs to take as their subject)

| in | subj, out
|---|---
| out | subj, var
| try | subj, try

Example: object-control

- Here: persuades offers its object for sleep to take as the sleeper-argument

Control vs. Raising

- Raising verbs do also offer a syntactic function for an embedded verb to take, but this argument is not theirs on the argument structure-level

Examples: object-control

| in | subj, out
|---|---
| out | subj, var
| try | subj, try

Control vs. Raising

| in | subj, out
|---|---
| out | subj, var
| try | subj, try

Linking constraint (second version)

- Semantic arguments may only be realized by appropriate syntactic functions
- Embedded verbs can take offered dependents of control verbs as their argument

\[ v \xrightarrow{\text{offer}} v' \text{ only if } v \xrightarrow{\varphi} v' \land \varphi \in \text{linko}(v) \]

\[ \text{or } v'' \xrightarrow{...} v' \land \varphi \in \text{offer}(v'') \land \text{subj} \in \text{linko}(v) \]
Conclusions

- Introduced the TDG grammar formalism
- Went the first step towards a concurrent syntax-semantics interface: addition of the argument structure
- Control and raising-phenomena dealt with rather elegantly
- For the curious: parser implementation available on www.mozart-oz.org

Outlook

- Extend the syntax-semantics interface (esp. to handle modifiers more properly)
- Use argument structure to construct a semantics in a logical formalism (first choice: CLLS; CHORUS-project)
- Investigate in which ways concurrency can prove useful