Groups for Surface Realization with Extensible Dependency Grammar

Ralph Debusmann

Programming Systems Lab
Saarland University, Saarbrücken, Germany
**Resurgent interest**

- resurgent interest in Dependency Grammar (DG) \( (\text{Tesnière} \ 59; \ \text{Sgall} \ et\ \text{al.} \ 86; \ \text{Mel’čuk} \ 88) \)

- core DG concepts incorporated into most grammar formalisms, also phrase structure-based (HPSG, LFG, TAG)

- new DG-based grammar formalisms \( (\text{Nasr} \ 95; \ \text{Heinecke} \ et\ \text{al.} \ 98; \ \text{Bröker} \ 99; \ \text{Gerdes} \ \text{and} \ \text{Kahane} \ 01; \ \text{Kruijff} \ 01; \ \text{Joshi} \ \text{and} \ \text{Rambow} \ 03) \)
A controversy

- assume a 1:1-correspondence between words and nodes in the dependency graph?
- simplifies the formalization of DGs substantially
- but: breaks when modeling semantics
- i.e. also breaks when what we want to is surface realization (generation), where we go from semantics to syntax
- one example: multiword expressions (MWEs): one semantic node corresponds to more than one word
Weakening the 1:1-assumption

- most DG grammarians interested in semantics have *weakened* the 1:1-assumption
- Tesnière: *nuclei* group together sets of nodes
- Sgall et al: *deletion* of solely syntactically motivated nodes
- Mel’čuk: *paraphrasing rules*
- but: these attempts to weaken the 1:1-correspondence have not yet been formalized declaratively
Extensible Dependency Grammar (XDG)

- new grammar formalism (Debusmann et al. 04) based on Topological Dependency Grammar (TDG) (Duchier and Debusmann 01)
- declaratively formalized
- formalization used directly in the XDG solver for parsing and generation
**XDG and the 1:1-assumption**

- XDG solving *efficient* at least for our *smaller-scale* handwritten example grammars
- but: good results hinge substantially on the 1:1-*correspondence*
- but for generation, we have no choice: we *must weaken* the 1:1-*correspondence* too
Weakening the 1:1-assumption for XDG

- in this talk, we show how to weaken the 1:1-assumption for XDG, without sacrificing the potential for efficient parsing and generation
- new layer of lexical organization called groups, above the basic XDG lexicon
- groups describe MWEs as tuples of dependency subgraphs
Overview

1. Introduction
2. *Extensible Dependency Grammar (XDG)*
3. Groups
4. Compilation
5. Conclusions
Extensible Dependency Grammar (XDG)

- new grammar formalism (Debusmann et al. 04)
- characterizes linguistic structure along arbitrary many *dimensions*
- all dimensions correspond to dependency graphs, sharing the same set of nodes but having different edges
Well-formedness conditions

- well-formedness conditions determined by principles
- principles can be one-dimensional (applying to a single dimension only), or multi-dimensional (constraining the relation between several dimensions)
- basic one-dimensional principle: valency
- basic multi-dimensional principle: linking (syntactic realization of semantic arguments)
The lexicon

- XDG is *highly lexicalized*
- *lexical entries* serve as the *parameters* for the principles
- since a lexical entry constrains all dimensions simultaneously, it can also help to *synchronize* the various dimensions
Example analysis

- two-dimensional XDG analysis of “He dates her” (syntax left, semantics right):

- used principles:
  1. syntactic valency
  2. semantic valency
  3. linking
1. Syntactic valency

- syntactic analysis:

```
He dates her
```

Groups for Surface Realization with Extensible Dependency Grammar – p.13
2. Semantic valency

- semantic analysis:

  - In: {arg1*}
    - Out: {}
  - In: {arg2*}
    - Out: {}
  - In: {}
    - Out: {arg1!, arg2!}

- He dates her
3. Linking

- semantic and syntactic analyses:

\[
\text{link: \{arg1: \{subj\}, arg2: \{obj\}\}}
\]
**Parsing and generation**

- **XDG solver**
- implements a declarative axiomatization of XDG as a constraint satisfaction problem \((\text{Duchier 03})\)
- XDG solving is \textit{NP-complete} \((\text{Koller and Striegnitz 02})\)
- average-case complexity polynomial for smaller-scale handwritten grammars
- research on XDG solving of large-scale grammars in progress
Overview

1. Introduction
2. Extensible Dependency Grammar (XDG)
3. Groups
4. Compilation
5. Conclusions
MWEs as contiguous substrings?

- **example paraphrase:**
  1. “He dates her.”
  2. “He takes her out.”
- **XDG analysis:**

```
.       .       .
He      takes    her  out
```

- i.e. we **cannot** treat MWEs as _contiguous_ word strings:
  “takes out” interrupted by object “her”
MWEs as dependency subgraphs!

- instead, we implement the *continuity hypothesis* (Kay and Fillmore 99)
- idea: model MWEs as *dependency subgraphs*
- new layer of lexical organization: *groups*
- a group is a *tuple of dependency subgraphs* covering one or more node
- each of the *components* correspond to a *dimension*
Example groups

- group for “dates”:

- group for “takes out”:

- groups can leave out nodes present in the syntax in the semantics (here: “out”)
Support verbs

• more complicated paraphrase:
  1. “He argues with her.”
  2. “He has an argument with her.”

• in 2., “has” is a support verb; the semantic head of the construction is the noun “argument”
XDG analysis

- XDG analysis of the support verb construction:

  

  \[
  \text{He has an argument with her}
  \]

  

  \[
  \text{arg1} \quad \text{arg2}
  \]

  

  \[
  \text{He has an argument with her}
  \]

  

- Interdependencies: “argument” is the object of “has” in the syntax, but the semantic head in the semantics.
Groups

- groups for the support verb construction:
Groups are nice

- groups can capture difficult constructions such as the support verb construction quite elegantly
- key aspect: *multi-dimensionality*, describing tuples of dependency subgraphs over a *shared* set of nodes
- sharing: helps to express *interdependencies* between the different dimensions
- groups can be regarded as a declarative formalization of Mel’čuk’s *paraphrasing rules* (Mel’čuk 96)
- or as a realization of the *extended domain of locality* of TAG (Joshi 87) for DG
Overview

1. Introduction
2. Extensible Dependency Grammar (XDG)
3. Groups
4. Compilation
5. Conclusions
Compilation

- groups are a *conservative* extension to XDG
- can be *compiled* into simple XDG lexical entries for individual words
- benefit: we can *retain* XDG in its entirety, including the XDG solver for *parsing* and *generation*
- three steps:
  1. node deletion
  2. dependency subgraphs
  3. group coherence
1. Node deletion

- on each dimension, *each word* must correspond to precisely *one node* in the dependency graph
- the *groups* shown above clearly *violate* this assumption: nodes present in the syntax were omitted in the semantics
- idea: accommodate deletion of nodes by introducing an *additional root node* in each analysis
**Node deletion example**

- example:

  [Diagram of a tree structure showing node deletion with labels: `arg1`, `arg2`, `He`, `has`, `an`, `argument`, `with`, `hér`, `del`, `root`, `old root`, `deleted nodes`]

  - old root = *root*-daughter of the new root
  - deleted nodes = *del*-daughters
2. **Dependency subgraphs**

- second step: compile dependency subgraphs into lexical entries for individual words
- idea: use valency (*in* and *out* features)
**Dependency subgraphs example (1/2)**

- **syntax:**

  ![Dependency subgraphs diagram](chart)

  - in: \{obj?\} out: \{det!, pmod!\}
  - in: \{pmod?\} out: \{pcomp!\}
  - in: \{root?\} out: \{subj!, obj!\}
  - in: \{det?\} out: \{\}
Dependency subgraphs example (2/2)

- semantics:

```

in: {root?}
out: {arg1! arg2!}

in: {del?}
out: {}
```

Groups for Surface Realization with Extensible Dependency Grammar – p.31
3. **Group coherence**

- ensure that *inner group nodes stay together*
- each node has feature denoting its *group ID*
- group IDs must match for each edge within a group
- expressed by a lexicalized principle
**Group coherence example**

- “an”, “argument” and “with” are inner nodes
- group IDs must match for the endpoints of the *obj*, *det* and *pmod* edges
Parsing

- for parsing, we use the existing XDG solver *unchanged*
Generation

- we can use the same group lexicon as for parsing
- caveat: need to introduce a finite set of extra nodes to fill up the groups
- to realize a semantic literal $s$, introduce as many nodes as the largest group which verbalizes $s$
- assume argue’ can be realized either by “argue with” (2) or “has an argument with” (4): introduce 4 nodes
Overview

1. Introduction
2. Extensible Dependency Grammar (XDG)
3. Groups
4. Compilation
5. Conclusions
Conclusions

- groups allow to weaken the 1:1-correspondence between nodes and words in XDG
- new layer of lexical organization
- powerful enough to handle complicated MWEs (e.g. support verb constructions)

Benefits:
1. conservative extension: we can retain XDG in its entirety, including the XDG solver
2. we can use the same group lexicon for both parsing and generation
Open questions

- integration of groups and the metagrammatical functionality of the XDG lexicon for individual lexical entries
- how does this all scale up to large-scale grammars?