Introduction to Tree Adjoining Grammar
Grammar Implementation with XMG

Timm Lichte

DGfS-CL Fall School 2011

2. week, 4. session

08.09.2011

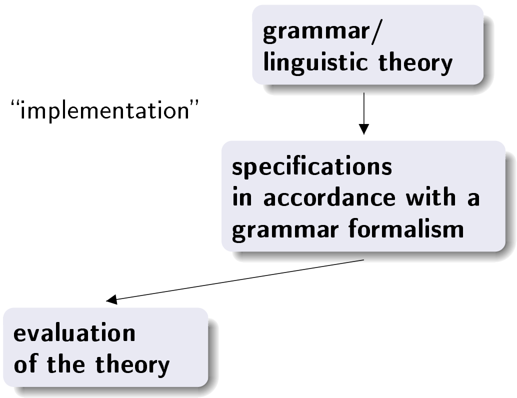
HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

SFB 991

Outline

1. What is grammar implementation?
2. Two ways of tree template implementation:
   - Metarules
   - Metagrammars
3. eXtended Metagrammar (XMG)
4. A case study with XMG

Two kinds of grammar implementation



grammar/ linguistic theory

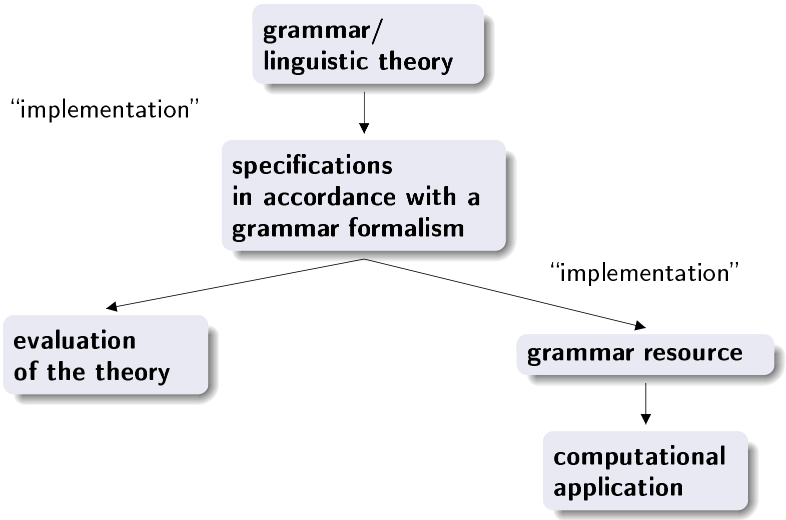
“implementation”

specifications in accordance with a grammar formalism

evaluation of the theory

As is frequently pointed out but cannot be overemphasized, an important goal of formalization in linguistics is to enable subsequent researchers ***to see the defects of an analysis as clearly as its merits****; only then can progress be made efficiently.* [Dowty, 1979, 322]

Two kinds of grammar implementation



grammar/ linguistic theory

“implementation”

specifications in accordance with a grammar formalism

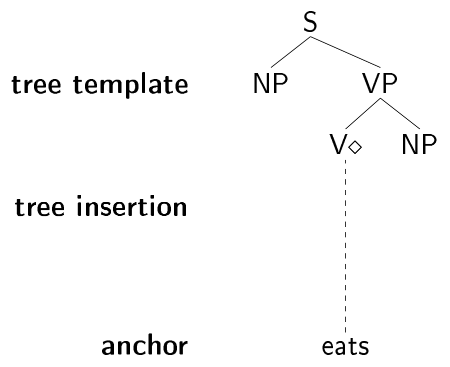
“implementation”

evaluation of the theory

grammar resource

computational application

What kind of grammar resource?



The implementation task for LTAG

General task

Implement a large-coverage LTAG, i.e. based on the XTAG grammar!

Subtasks:

1. Generate unlexicalized trees (= tree templates)!
2. Generate a database of lexical anchors (= the lexicon)!
3. Connect the tree templates with the lexicon (= lexical insertion)!

Two ways of grammar implementation with TAG

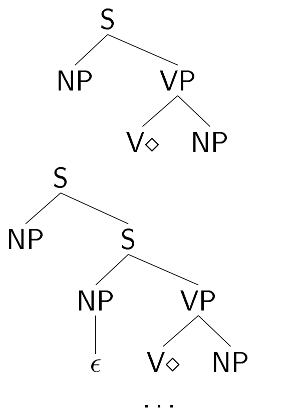
- **XTAG tools** [XTAG Research Group, 2001]
    1. implementation tools (with metarules)
    2. editor/viewer for MorphDB and SynDB
    3. parser
- **XMG + lexConverter + TuLiPA**
    1. XMG: eXtensible MetaGrammar [Duchier et al., 2004]
    2. lexConverter (LEX2ALL)
    3. TuLiPA: Tübingen Linguistic Parsing Architecture [Parmentier et al., 2008]

Outline

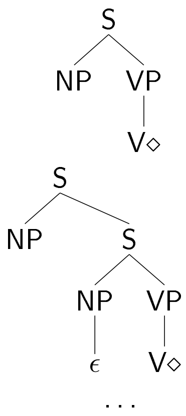
1. What is grammar implementation?
2. Two ways of tree template implementation:
    - Metarules
    - Metagrammars
3. eXtended Metagrammar (XMG)
4. A case study with XMG

The situation

39 templates for transitive verbs



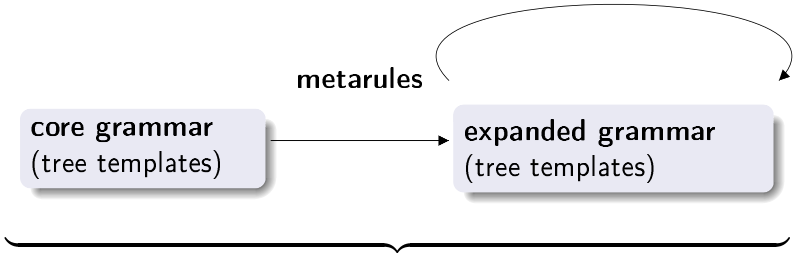
12 tree templates for intransitive verbs



Basically, XTAG defines a set of 221 unrelated tree templates.

Metarules for LTAG

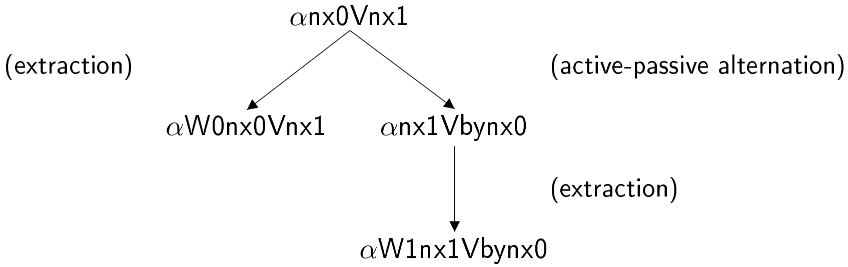
[Becker, 1994], [Becker, 2000], [Prolo, 2002]
Idea from GPSG [Gazdar, 1981]



metarules connect tree templates of a **tree family**

Metarules for LTAG: Example

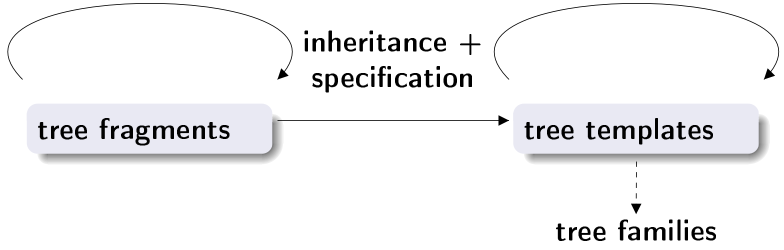
Tnx0nx1:



Metarules do not only add structure, they can also eliminate structure!

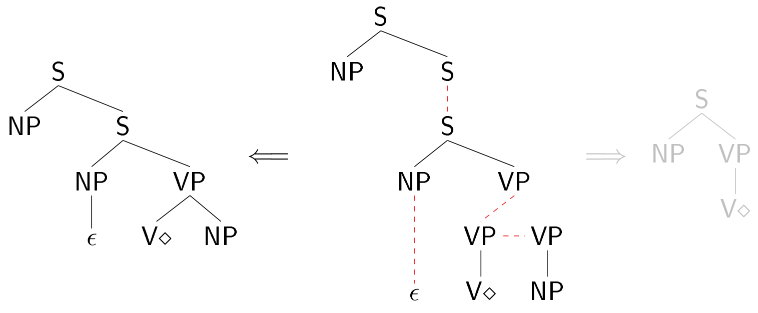
Metagrammars for LTAG

[Candito, 1996], [Xia, 2001], [Crabbé, 2005]

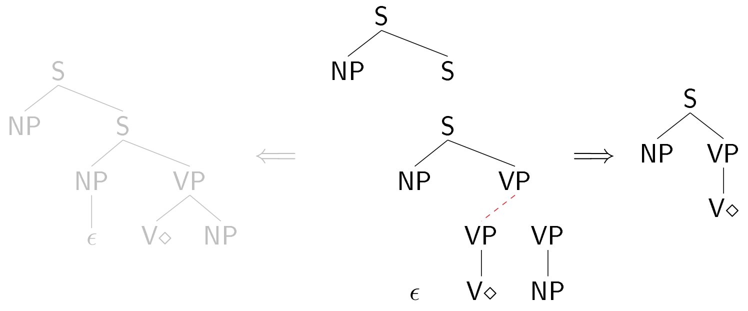


- **tree fragments**: additional layer of abstraction below the level of tree templates
- A tree template is the result of combining and specifying tree fragments and tree templates.
- The notion of **tree families** is independent from the construction of tree templates!

Metagrammars for LTAG: Example



Metagrammars for LTAG: Example



XMG - Background

- name of the metagrammar formalism and of a metagrammar compiler
- developed at LORIA, Nancy, France
- written in Oz/Mozart
- available at `http://sourcesup.cru.fr/xmg`

⇒ Other metagrammar implementations exist, but XMG is the most elaborate one.

Some existing implementations using XMG:

- French: FrenchTAG [Crabbé, 2005]
- English: XTAG with XMG [Alahverdzhieva, 2008]
- German: GerTT [Kallmeyer et al., 2008]

XMG - Description languages

$\mathcal{L}_D$: Description language for tree fragments

Let ?x and ?y be nodes:

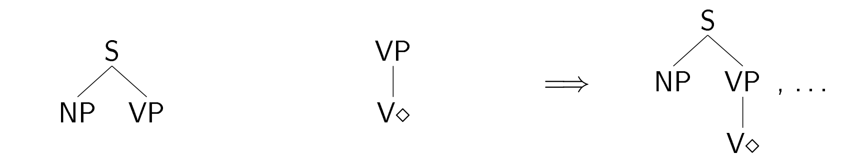
```
Description ::= ( ?x -> ?y | ?x ->+ ?y | ?x ->* ?y |
                  ?x ≫ ?y | ?x ≫+ ?y | ?x ≫* ?y |
                  ?x = ?y |
                  ?x[f:E] | ?x(p:E) |
                  Description ∧ Description )
```

$\mathcal{L}_C$: Description language for the combination of tree fragments

```
Class ::= Name → Content

Content ::= ( Description | Name |
              Content ∨ Content |
              Content ∧ Content )
```

XMG - Description languages - Examples



```
?S -> ?NP,
?S -> ?VP1,
?NP ≫ ?VP1,
?S[cat:s],
?NP[cat:np],
?VP1[cat:vp]
```

```
?VP2 -> ?V,
?VP2[cat:vp],
?V[cat:v],
?V(mark:anchor)
```

```
?S -> ?NP,
?S -> ?VP1,
?NP ≫ ?VP1,
?S[cat:s],
?NP[cat:np],
?VP1[cat:vp],
?VP2 -> ?V,
?VP2[cat:vp],
?V[cat:v],
?V(mark:anchor)
```

XMG - Node variables and compiling

- Node variables have a scope local to the class (= name space).
- Tree descriptions can denote more than one tree fragment! BUT: Each of the tree fragments has to comply with all of the tree descriptions!

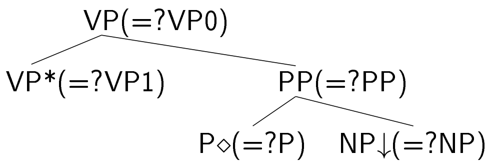
When the class intransitive is compiled:

1. XMG accumulates all tree descriptions involved in intransitive, and
2. XMG identifies tree fragments and tree templates by merging node variables or drawing edges.

E.g., in the previous example, the node variables ?VP1 and ?VP2 can be merged, or ?VP1 can dominate ?VP2.

XMG - The source code - The structure of trees

There are two ways to encode the structure of trees: (1) through tree descriptions, or (2) through brackets and linear order.



```
class betavxPnx
declare ?VP0 ?VP1 ?PP ?P ?NP
{<syn>{
node ?VP0; node ?VP1;
node ?PP; node ?NP;
node ?P;
?VP0 -> ?VP1; ?VP0 -> ?PP;
?PP -> ?P; ?PP -> ?NP;
?VP1 >> ?PP; ?P >> ?NP
}}
```

```
class betavxPnx
declare ?VP0 ?VP1 ?PP ?P ?NP
{<syn>{
node ?VP0 {
   node ?VP1
   node ?PP {
     node ?P
     node ?NP
   }
} }}
```

XMG - The source code - Properties and feature structures

Firstly, the value types of features and properties have to be declared.

```
type MARK = {subst, foot, anchor, coanchor, flex }
type CAT = {np,v,vp,s}
```

Secondly, properties and features must be declared as well.

```
property mark : MARK
feature cat : CAT
```

Finally, properties and features of nodes can be specified.

```
class betavxPnx
{ ...
node ?NP (mark = subst) [cat = np]
... }
```

XMG - The source code - Complex feature structures

How to declare and use complex features?

```
type AGR = [  3rdsing : bool,
              num : NUM,
              pers : PERS,
              gen : GEN        ]
feature agr:AGR
...
node ?NP [agr = [3rdsing = +] ]
...
```

Top-bottom-feature-structures

In XMG, there are predefined complex features top and bot for the specification of top-bottom-feature structures. Otherwise, feature specifications hold for both top and bottom.

Note: Links between features can be established by variables!

XMG - The source code - Reusing classes

General convention: Names of reused classes have [] as a postfix.

First method:

Class instantiations can be assigned to variables in the body. Only exported variables of the class can be used by means of the dot operator.

```
class betavxPnx
{ ...
?VPSpine = VPSpine[];
?VPSpine.?VP0 = ?XP;
... }
```

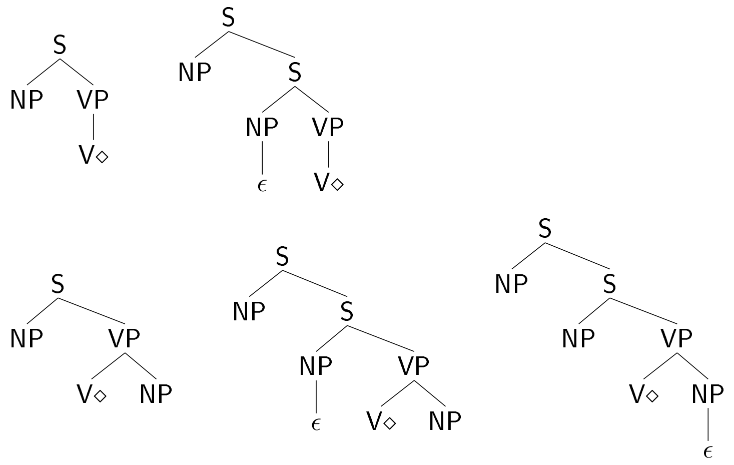
Second method:

Classes can be imported, such that all variables of the imported class, that have been exported, can be used directly.

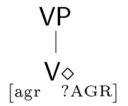
```
class betavxPnx
import VPSpine[]
{...
?VP0 = ?XP;
... }
```

XMG - Case study

How to describe the tree families for intransitive (Tnx0V) and transitive (Tnx0Vnx1) tree templates?

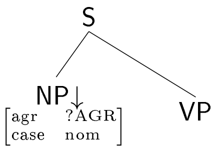


XMG - Case study - The fragments



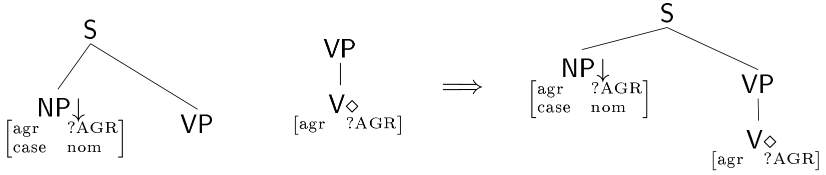
```
class VerbProjection
export ?VP ?V ?AGR
declare ?VP ?V ?AGR
{<syn>{
    node ?VP [cat = vp];
    node ?V (mark = anchor)[cat = v, agr = ?AGR];
    ?VP -> ?V
  }
}
```

XMG - Case study - The fragments



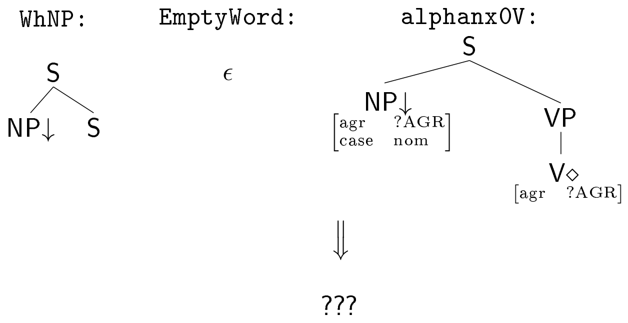
```
class Subject
export ?S ?NP ?VP ?AGR
declare ?S ?NP ?VP ?AGR
{ <syn>{
    node ?S [cat = s]{
        node ?NP (mark = subst)[cat = np, case = nom,
                                agr = ?AGR]
        node ?VP [cat = vp]
        }
    }
}
```

XMG - Case study - Building a tree template



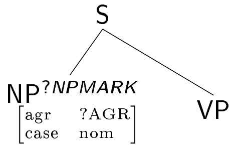
```
class alphanx0V
import VerbProjection[]
export ?S ?NP0
declare ?Subj ?S ?NP0
{
    ?Subj = Subject[];       ?NP0 = ?Subj.?NP;
    ?VP = ?Subj.?VP;         ?S = ?Subj.?S;
    ?AGR = ?Subj.?AGR
}
```

XMG - Case study - Adding fragments for extraction



In order to reuse `alphanx0V` here one has to underspecify the mark property of leaf nodes!

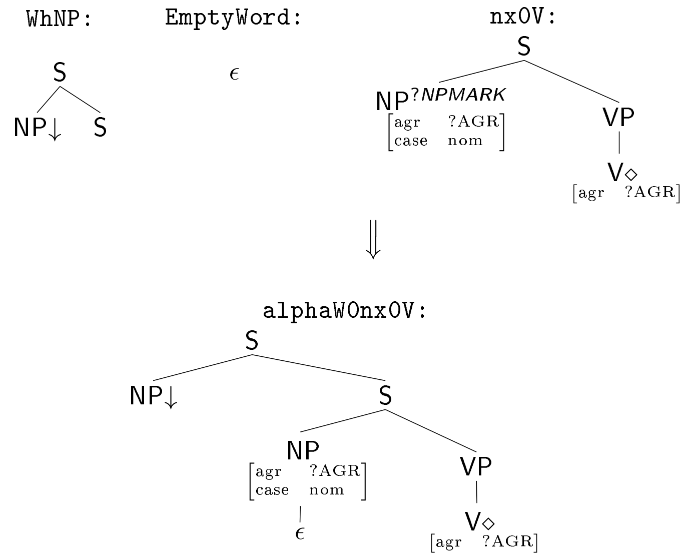
XMG - Case study - A redesigned subject fragment



```
class Subject
export ?S ?NP ?VP ?NPMARK ?AGR
declare ?S ?NP ?VP ?NPMARK ?AGR
{ <syn>{
    node ?S [cat = s]{
        node ?NP (mark = ?NPMARK)[cat = np, case = nom,
                                  agr = ?AGR]
        node ?VP [cat = vp]
        }
    }
}
```

Note: The modified subject class is used to define the class `nx0V`, which can be also reused in `alphanx0V`.

XMG - Case study - Adding fragments for extraction



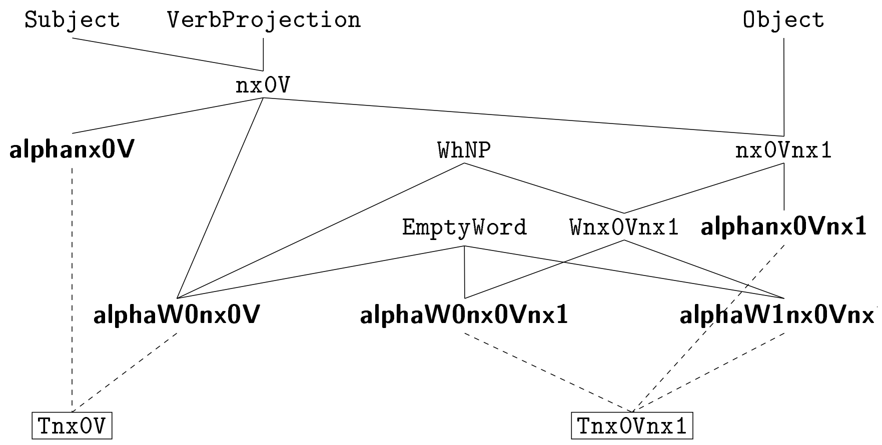
XMG - Case study - Declaring a tree family

```
class Tnx0V
declare ?Tnx0V
{
        ?Tnx0V = ( alphanx0V[] | alphaW0nx0V[] )
}

...

value Tnx0V
```

XMG - Case study - An XMG-hierarchy for Tnx0V and Tnx0Vnx1



Alahverdzhieva, K. (2008).
Reimplementing XTAG using XMG.
Master's thesis, University of Nancy 2 / University of Saarland.

Becker, T. (1994).
HyTAG: A New Type of Tree Adjoining Grammars for Hybrid Syntactic Representations of Free Word Order Languages.
PhD thesis, Universität des Saarlandes.

Becker, T. (2000).
Patterns in metarules for TAG.
In Abeillé, A. and Rambow, O., editors, Tree Adjoining Grammars: Formalisms, Linguistic Analyses and Processing, volume 107 of CSLI Lecture Notes, pages 331–342. CSLI Publications, Stanford.

Candito, M.-H. (1996).
A principle-based hierarchical representation of LTAGs.
In Proceedings of the 16th International Conference on Computational Linguistics (COLING 96), Copenhagen.

Crabbé, B. (2005).
Représentation informatique de grammaires d'arbres fortement lexicalisées: Le cas de la grammaire d'arbres adjoints.
PhD thesis, Université Nancy 2.

Dowty, D. R. (1979).
Word Meaning and Montague Grammar.
D. Reidel Publishing Company, Dordrecht, Boston, London.
Reprinted 1991 by Kluwer Academic Publishers.

Duchier, D., Le Roux, J., and Parmentier, Y. (2004).
The Metagrammar Compiler: An NLP Application with a Multi-paradigm Architecture.
In Second International Mozart/Oz Conference (MOZ'2004).

Gazdar, G. (1981).
Unbounded dependencies and coordinated structure.
Linguistic Inquiry, 12:155–182.

Kallmeyer, L., Lichte, T., Maier, W., Parmentier, Y., and Dellert, J. (2008).
Developing a TT-MCTAG for German with an RCG-based parser.
In (ELRA), E. L. R. A., editor, Proceedings of the Sixth International Language Resources and Evaluation (LREC'08), Marrakech, Morocco.

Parmentier, Y., Kallmeyer, L., Maier, W., Lichte, T., and Dellert, J. (2008).
TuLiPA: A syntax-semantics parsing environment for mildly context-sensitive formalisms.
In Proceedings of the Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+9), pages 121–128, Tübingen, Germany.

Prolo, C. A. (2002).
Generating the XTAG English grammar using metarules.
In Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002), pages 814–820, Taipei. Taiwan.

Xia, F. (2001).
Automatic grammar generation from two different perspectives.
PhD thesis, University of Pennsylvania.

XTAG Research Group (2001).
A Lexicalized Tree Adjoining Grammar for English.
Technical report, Institute for Research in Cognitive Science, University of Pennsylvania, Philadelphia, PA.