Grammar Implementation: Metagrammars & LTAG

Laura Kallmeyer & Benjamin Burkhardt
(Slides partly by Timm Lichte and Simon Petitjean)

Heinrich-Heine-Universität Düsseldorf

WS 2017/2018

Introduction

Large scale Tree Adjoining Grammars are composed of thousands of trees.

- ➔ Manual description by experts is very time consuming (see XTAG).
- ➔ Automatic methods: need an corpus

The Ideal

Precise Resources + Easier Development & Maintenance

Semiautomatic Methods

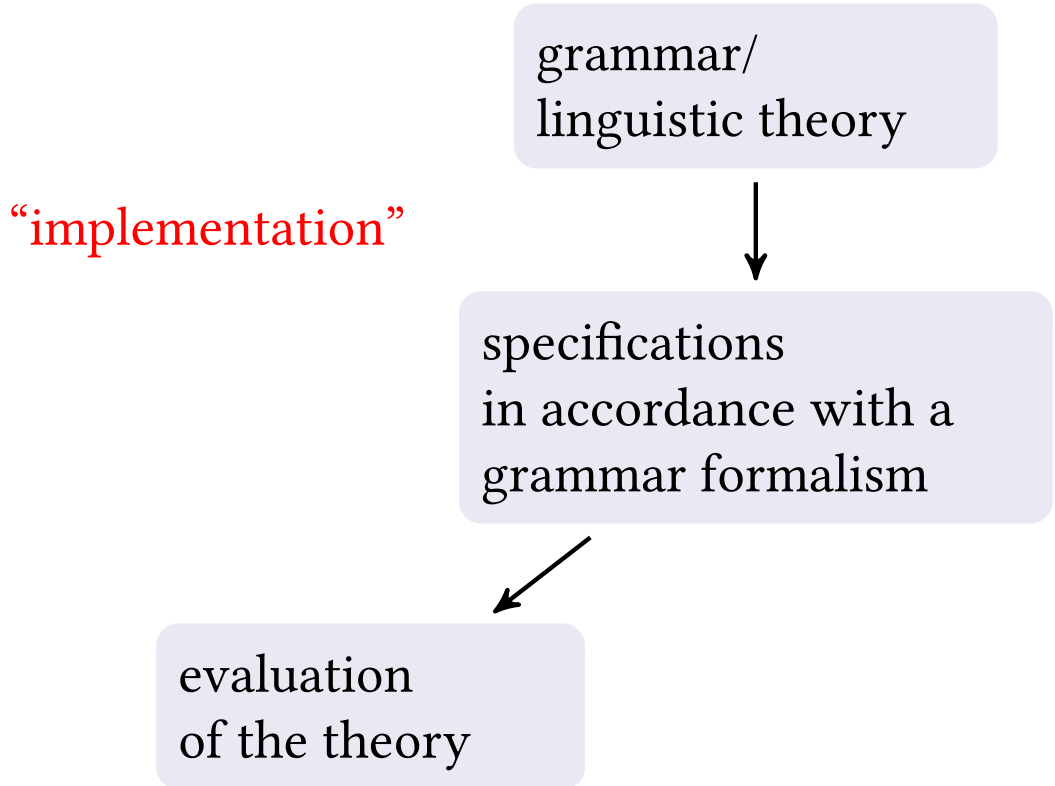
Grammar engineering: the task

grammar sketches, example analyses
(incomplete)
↓
implemented grammars, digital resource
(complete)
↓
grammar in action, parsing
(i.a. usable in NLP)

Grammar engineering: the problem

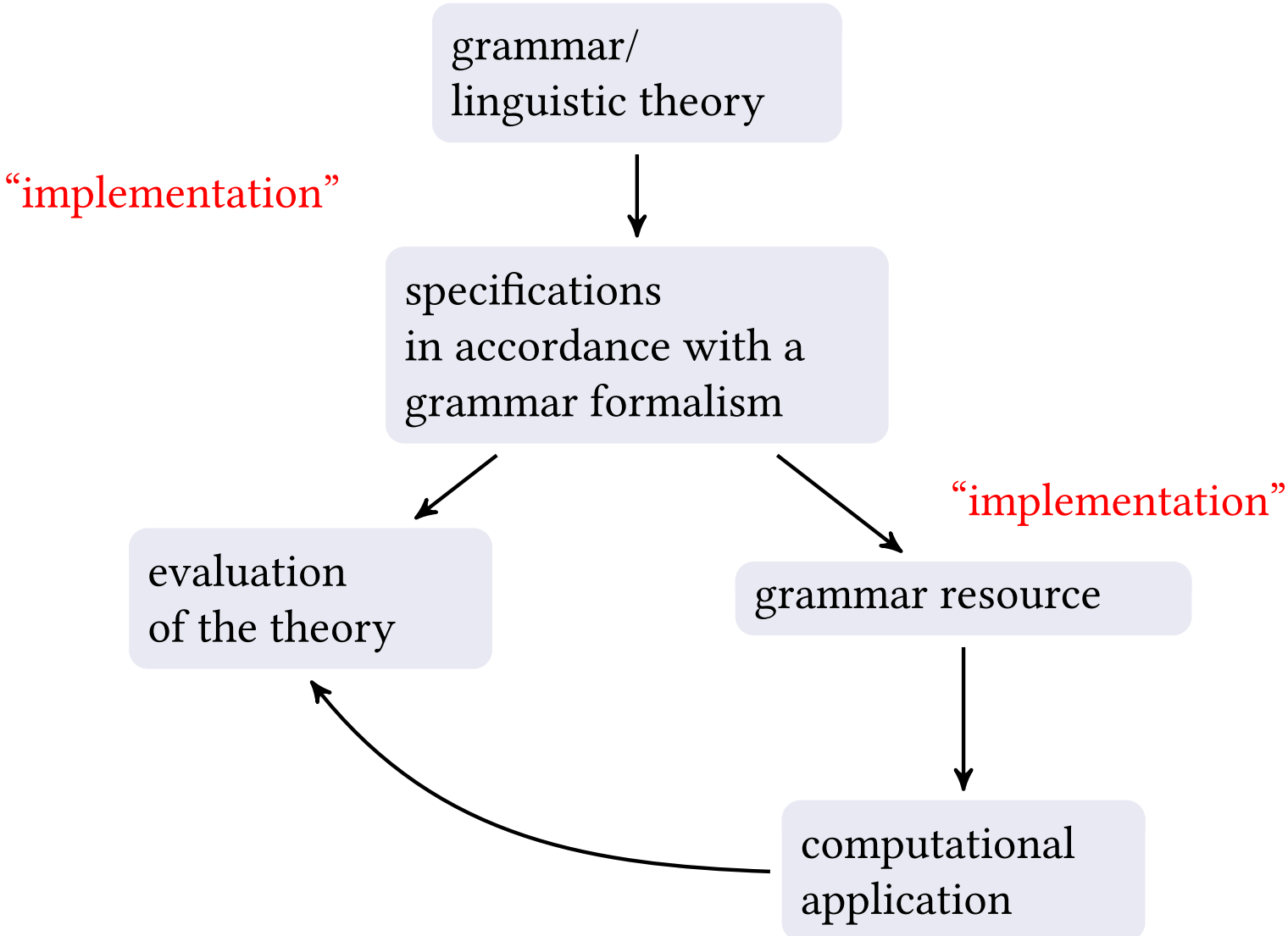
- How to factorize the set of templates?
  - ➔ express lexical generalizations, e.g. active-passive diathesis
  - ➔ define tree families
- How to turn this into an electronic resource?
- How to plug it into a lexicon and use it?

Two kinds of grammar implementation

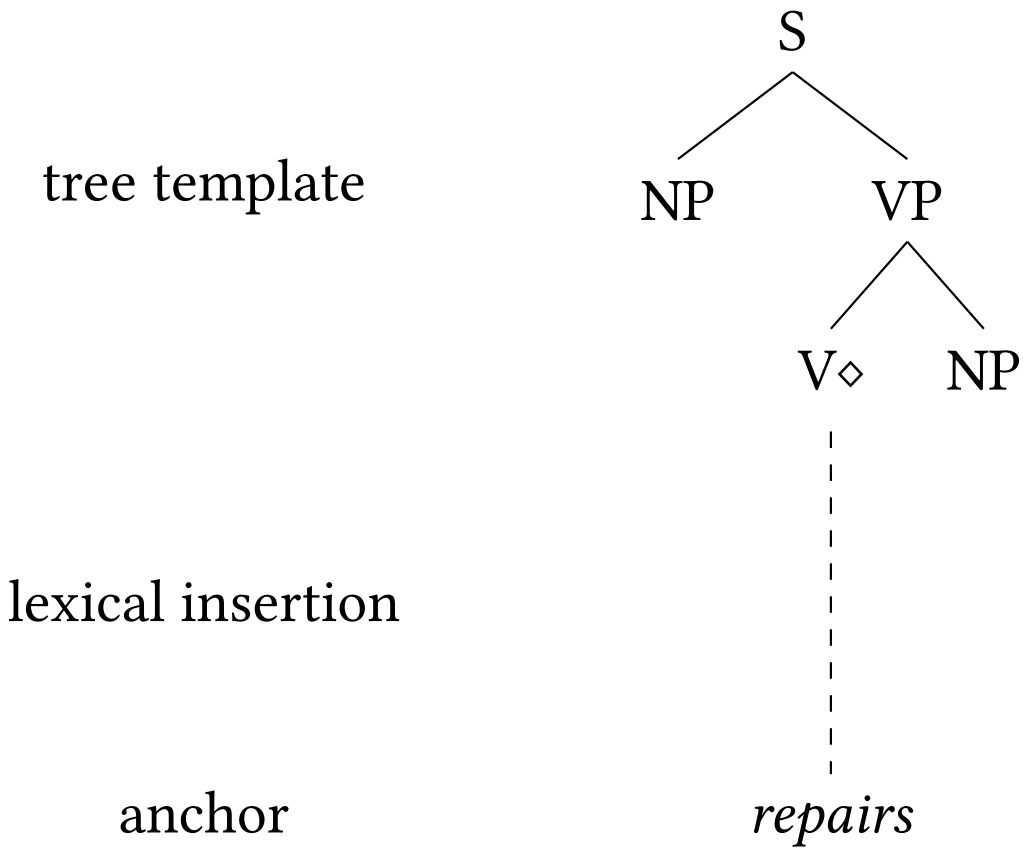


As is frequently pointed out but cannot be overemphasized, an important goal of formalization in linguistics is to enable subsequent researchers to see the defects of an analysis as clearly as its merits; only then can progress be made efficiently. (Dowty 1979: 322)

Two kinds of grammar implementation



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)!

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)!

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

Two existing toolkits:

XTAG tools[16]

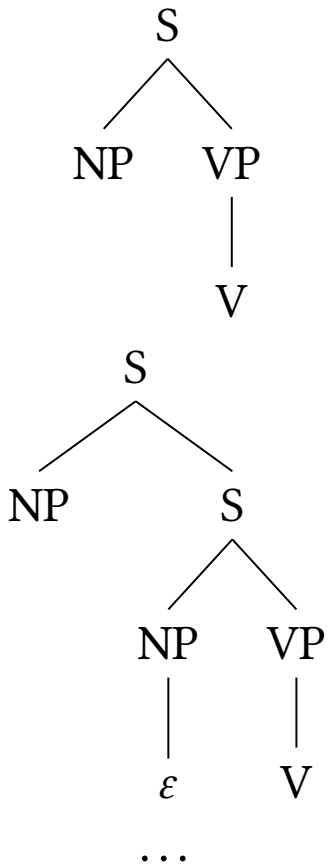
1. implementation tools
   $\Rightarrow$ **metarule approach**
2. editor/viewer for MorphDB and SynDB
3. parser

XMG + lexConverter + TuLiPA

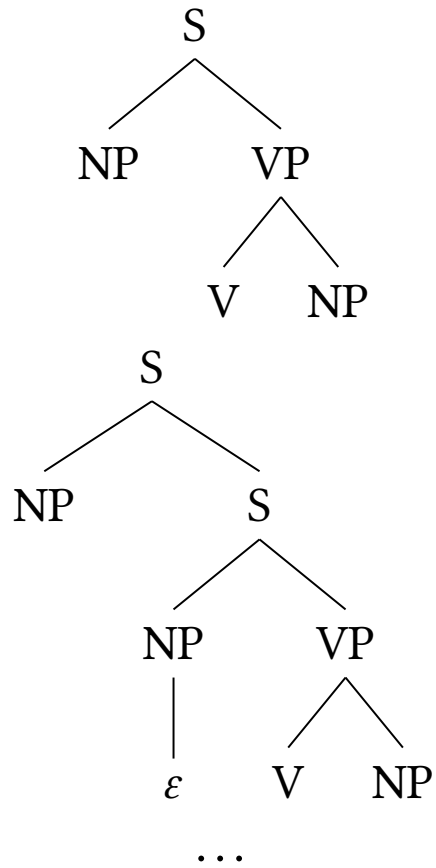
1. XMG: eXtensible MetaGrammar[6]
   $\Rightarrow$ **metagrammar approach**
2. lexConverter (LEX2ALL)
3. TuLiPA: Tübingen Linguistic Parsing Architecture[10]

The situation

12 templates for intransitive verbs



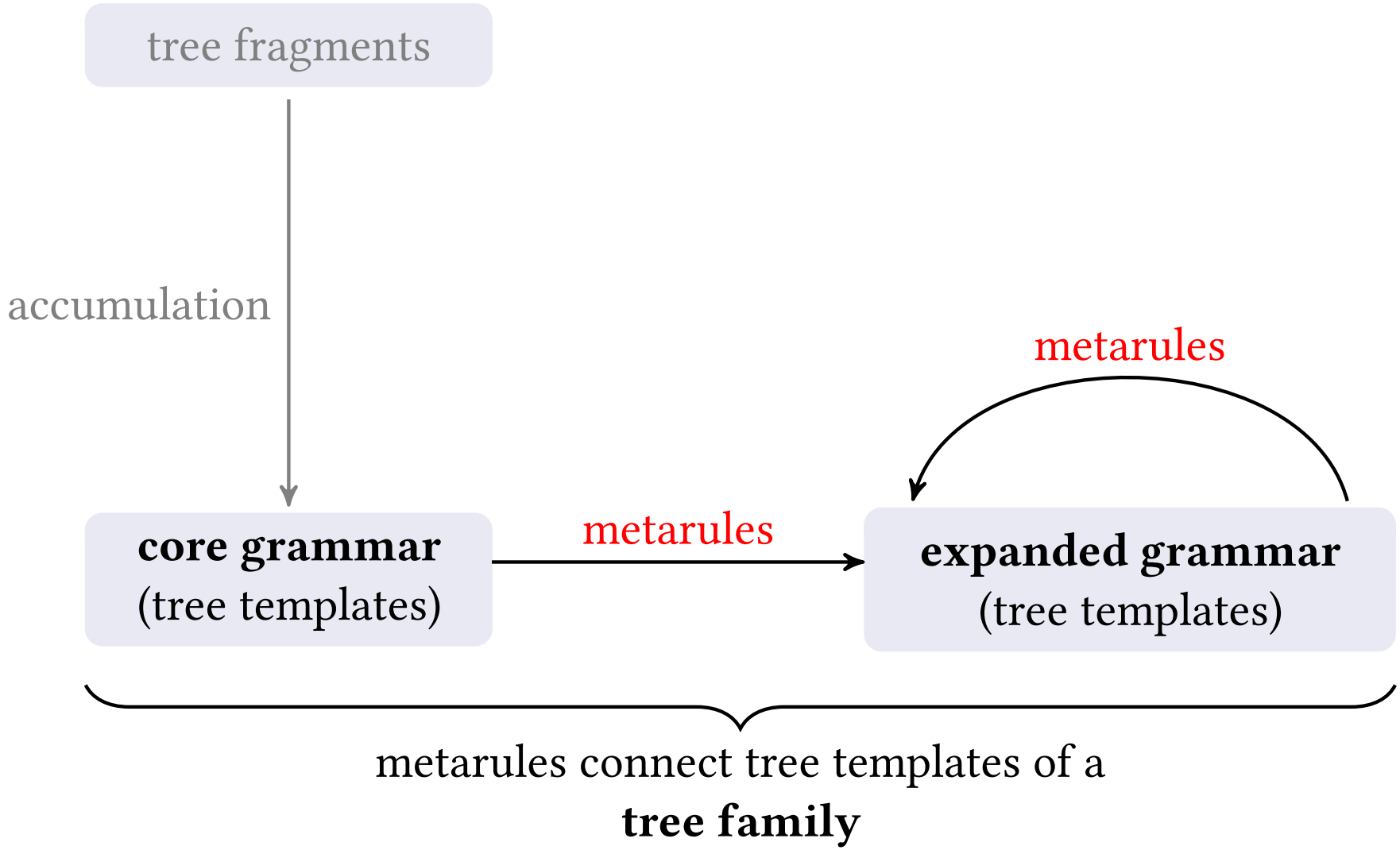
39 tree templates for transitive verbs



Basically, XTAG defines a set of 1008 unrelated tree templates.

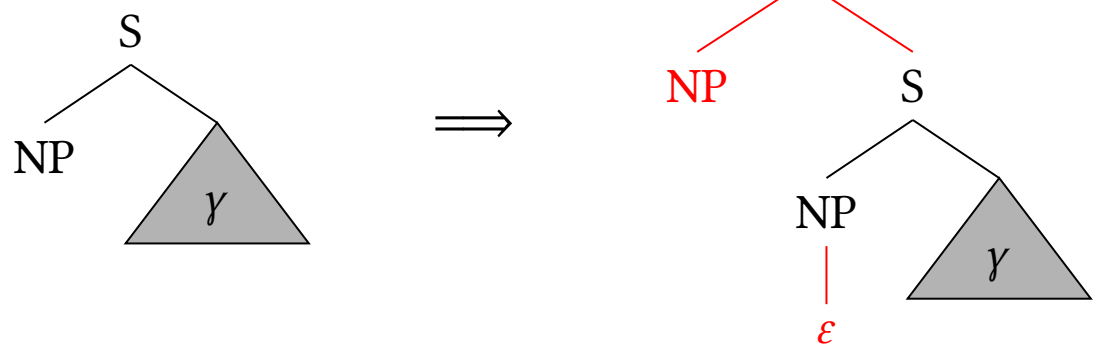
Metarules for LTAG

Idea from GPSG[8], later applied to XTAG[2,3,12]



Metarules for LTAG: Example

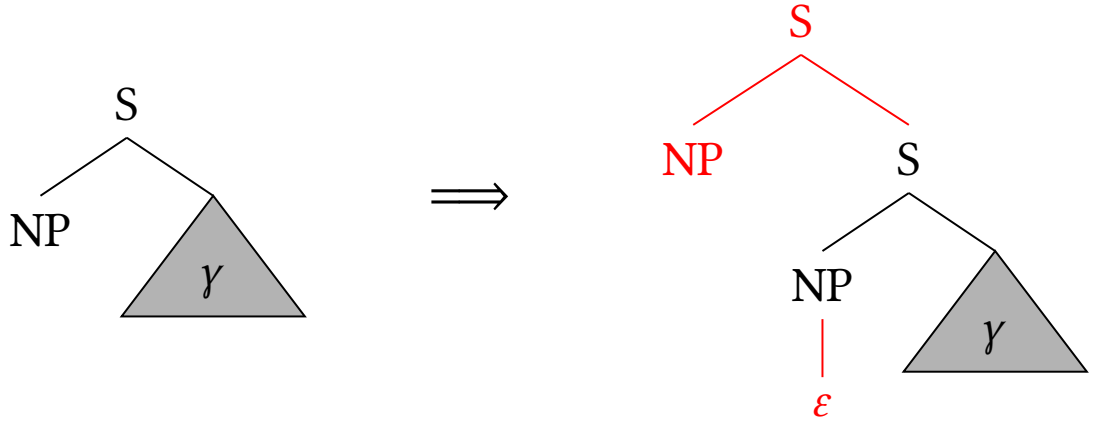
extraction:



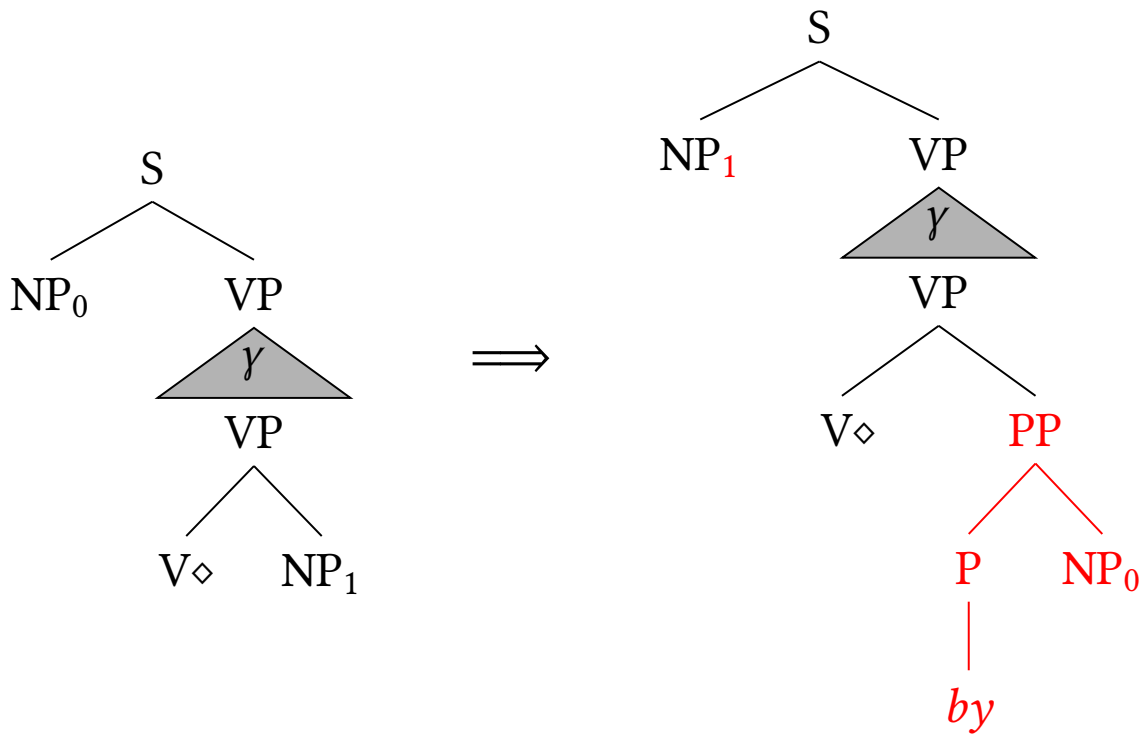
passivization:

Metarules for LTAG: Example

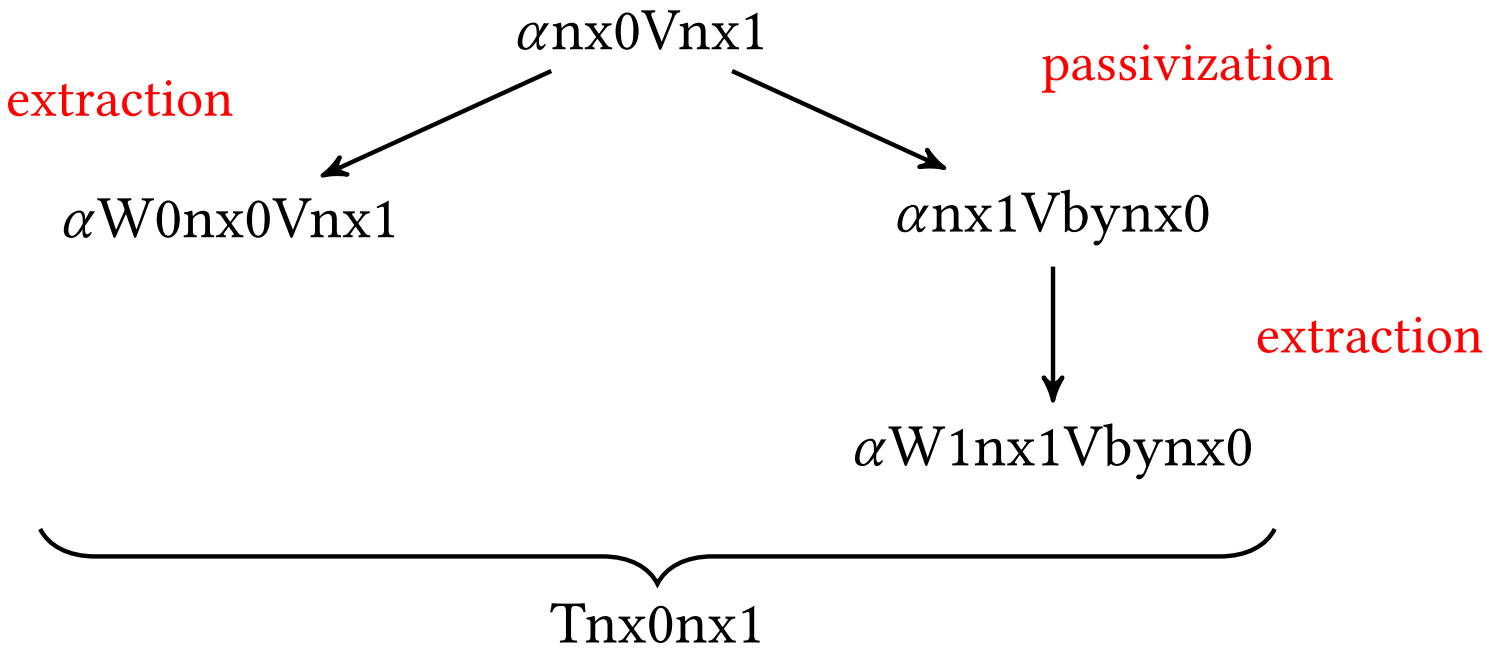
extraction:



passivization:



Metarules for LTAG: Example



Metarules for LTAG: Problems[2]

Metarules are very powerful:

- deletion, copying, recursive application, metavariables over trees
- order sensitive
- in the unrestricted case: undecidable[14]

Restrictions (GPSG):[13]

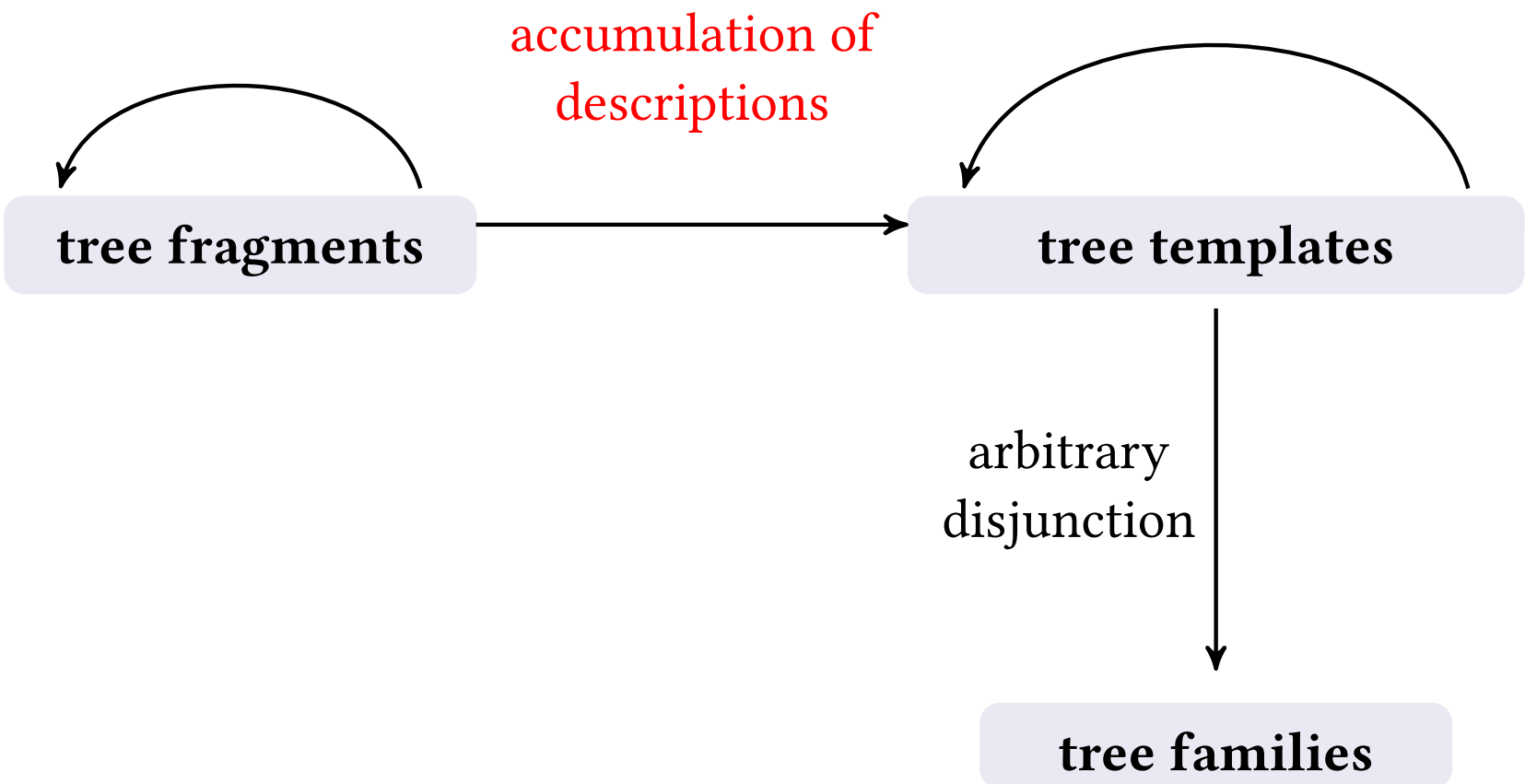
- finite closure: apply every metarule at most once!

  $\Rightarrow$ still NP-complete
- biclosure: apply at most two metarules in a row!

  $\Rightarrow$ insufficient for LTAG metarules[2]
- explicit rule ordering (by means of finite state automata)[12]

Metagrammars for LTAG

Candito (1996)[5,6,15]



Metagrammars for LTAG: Tree descriptions

$\mathcal{L}_D$: Description language for trees

Let n_1 and n_2 be node variables:

$$Description := \left(\begin{array}{lll} n_1 \rightarrow n_2 \mid & n_1 \rightarrow^+ n_2 \mid & n_1 \rightarrow^* n_2 \mid \\ n_1 \prec n_2 \mid & n_1 \prec^+ n_2 \mid & n_1 \prec^* n_2 \mid \\ n_1 = n_2 \mid & & \\ Description \wedge Description & & \end{array} \right)$$

Example:

corresponds to

$$\begin{array}{ll} n_S \rightarrow n_{NP} & \wedge \\ n_S \rightarrow n_{VP} & \wedge \\ n_{NP} \prec n_{VP} & \end{array}$$

corresponds to

$$\begin{array}{ll} n_S \rightarrow n_{NP1} & \wedge \\ n_S \rightarrow^+ n_{NP2} & \wedge \\ n_{NP1} \prec^* n_{NP2} & \end{array}$$

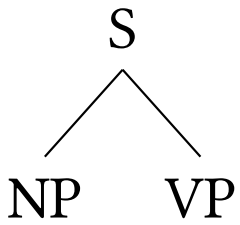
Metagrammars for LTAG: Tree descriptions

$\mathcal{L}_D$: Description language for trees

Let n_1 and n_2 be node variables:

$$Description := \left(\begin{array}{lll} n_1 \rightarrow n_2 \mid & n_1 \rightarrow^+ n_2 \mid & n_1 \rightarrow^* n_2 \mid \\ n_1 \prec n_2 \mid & n_1 \prec^+ n_2 \mid & n_1 \prec^* n_2 \mid \\ n_1 = n_2 \mid & & \\ Description \wedge Description & & \end{array} \right)$$

Example:



corresponds to

$$\begin{array}{ll} n_S \rightarrow n_{NP} & \wedge \\ n_S \rightarrow n_{VP} & \wedge \\ n_{NP} \prec n_{VP} & \end{array}$$

corresponds to

$$\begin{array}{ll} n_S \rightarrow n_{NP1} & \wedge \\ n_S \rightarrow^+ n_{NP2} & \wedge \\ n_{NP1} \prec^* n_{NP2} & \end{array}$$

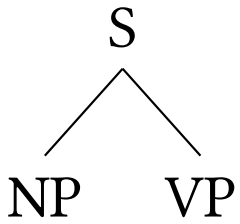
Metagrammars for LTAG: Tree descriptions

$\mathcal{L}_D$: Description language for trees

Let n_1 and n_2 be node variables:

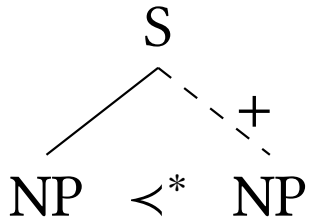
$$Description := \left(\begin{array}{lll} n_1 \rightarrow n_2 \mid & n_1 \rightarrow^+ n_2 \mid & n_1 \rightarrow^* n_2 \mid \\ n_1 \prec n_2 \mid & n_1 \prec^+ n_2 \mid & n_1 \prec^* n_2 \mid \\ n_1 = n_2 \mid & & \\ Description \wedge Description & & \end{array} \right)$$

Example:



corresponds to

$$\begin{array}{ll} n_S \rightarrow n_{NP} & \wedge \\ n_S \rightarrow n_{VP} & \wedge \\ n_{NP} \prec n_{VP} & \end{array}$$



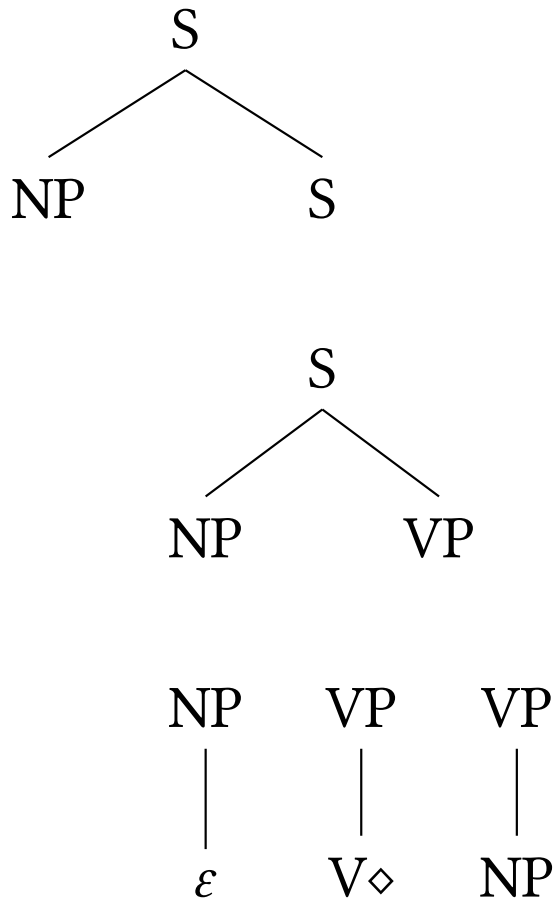
corresponds to

$$\begin{array}{ll} n_S \rightarrow n_{NP1} & \wedge \\ n_S \rightarrow^+ n_{NP2} & \wedge \\ n_{NP1} \prec^* n_{NP2} & \end{array}$$

Metagrammars for LTAG: Example

Minimal model of tree descriptions

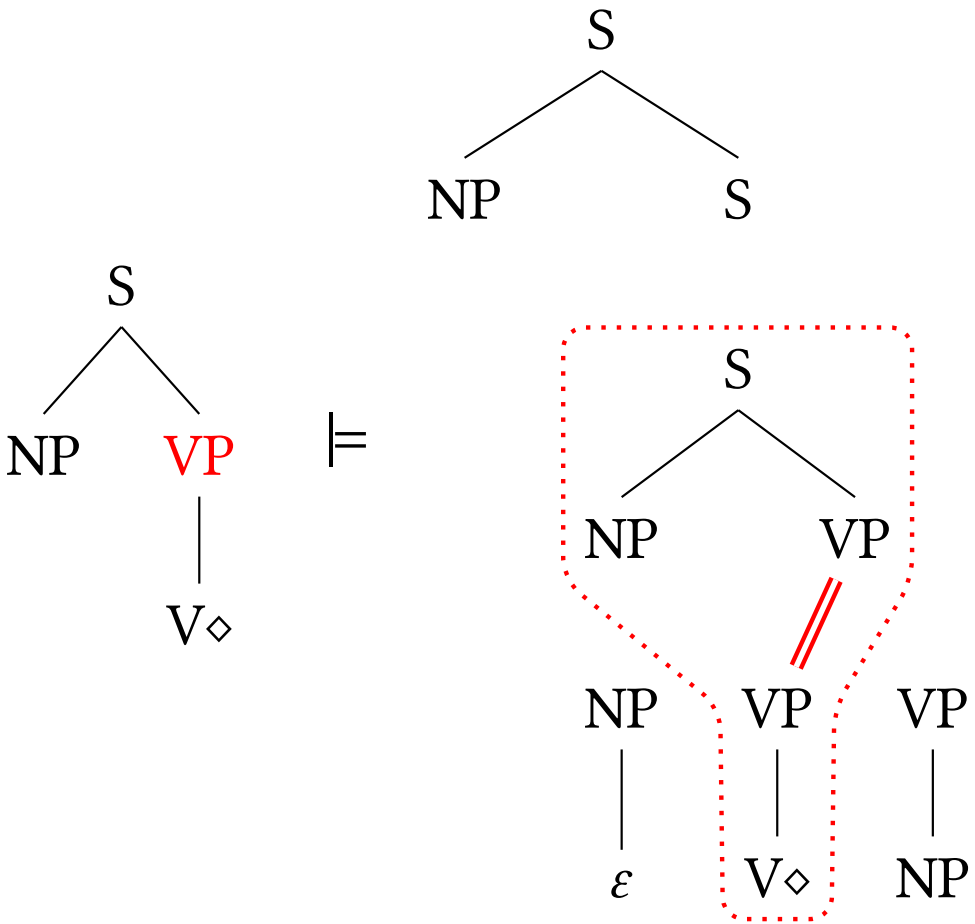
You may add edges but not nodes!



Metagrammars for LTAG: Example

Minimal model of tree descriptions

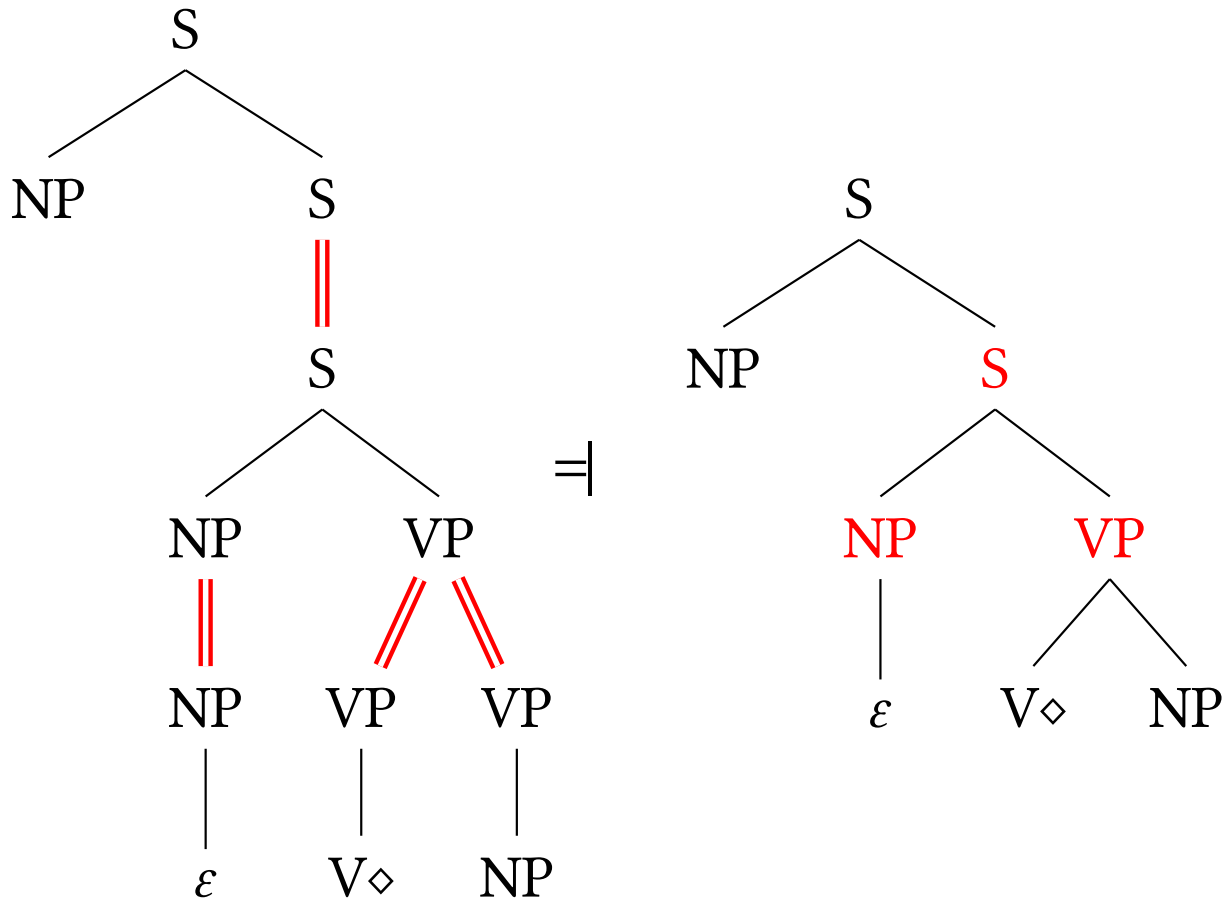
You may add edges but not nodes!



Metagrammars for LTAG: Example

Minimal model of tree descriptions

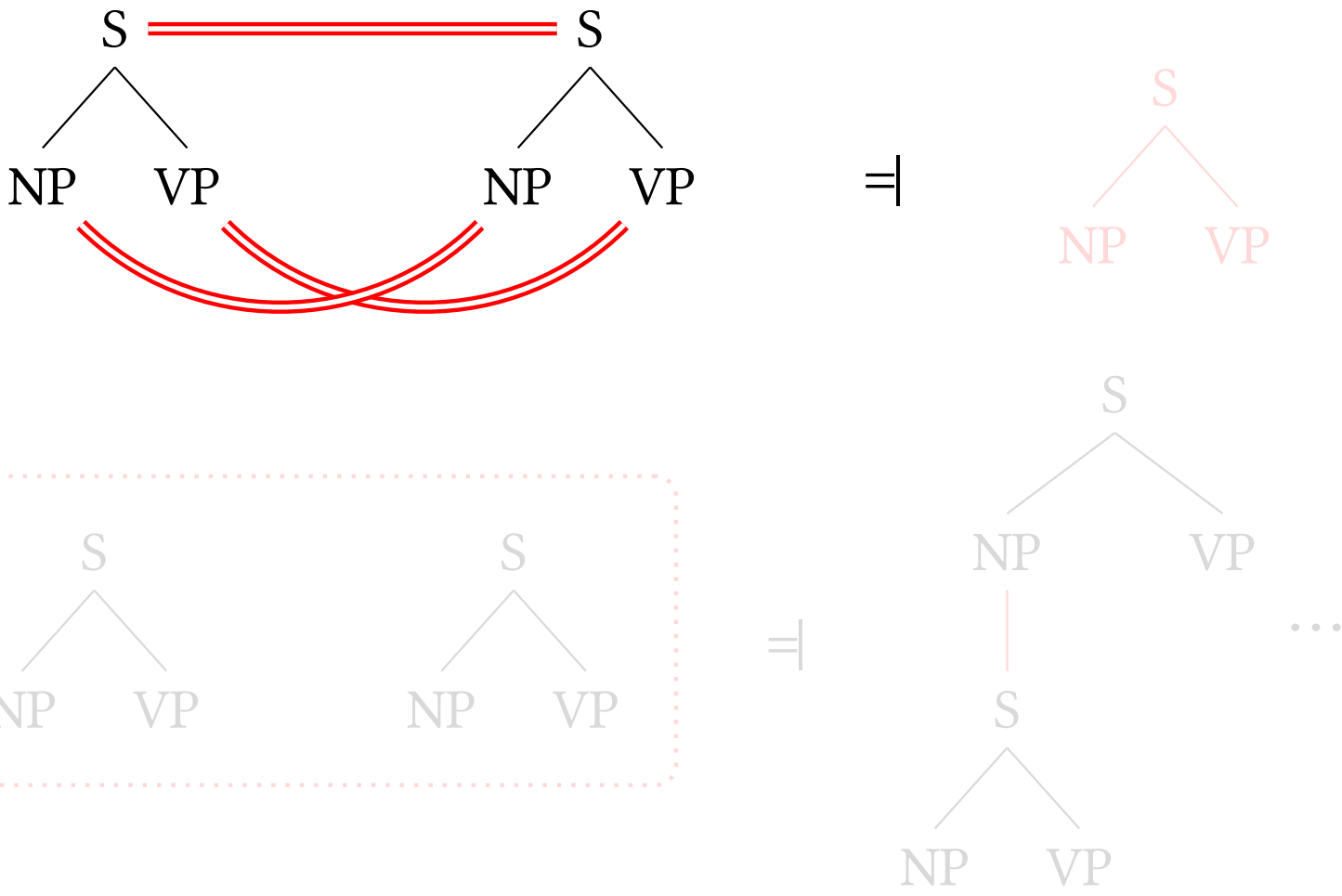
You may add edges but not nodes!



Metagrammars for LTAG: Example

Minimal model of tree descriptions

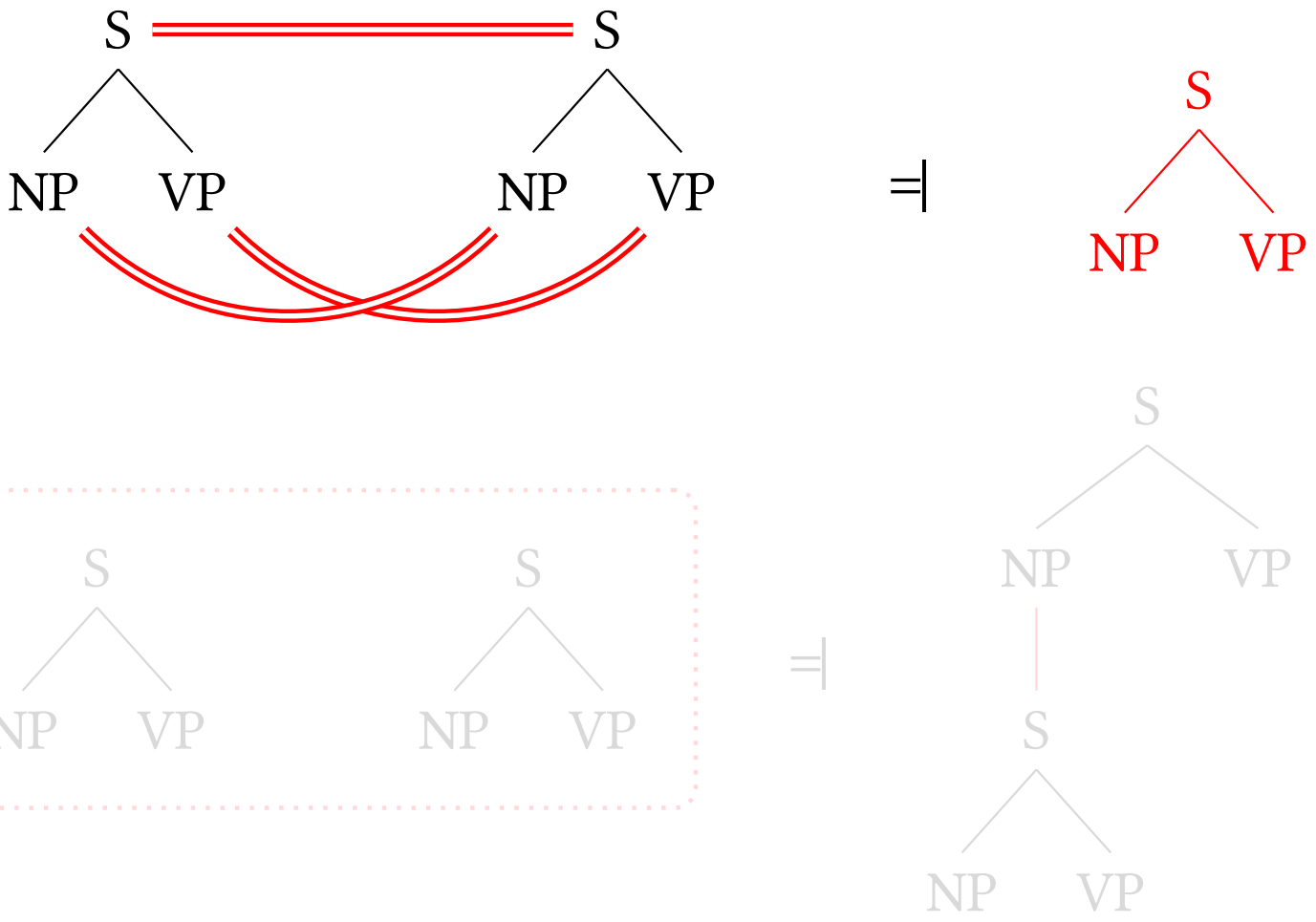
You may add edges but not nodes!



Metagrammars for LTAG: Example

Minimal model of tree descriptions

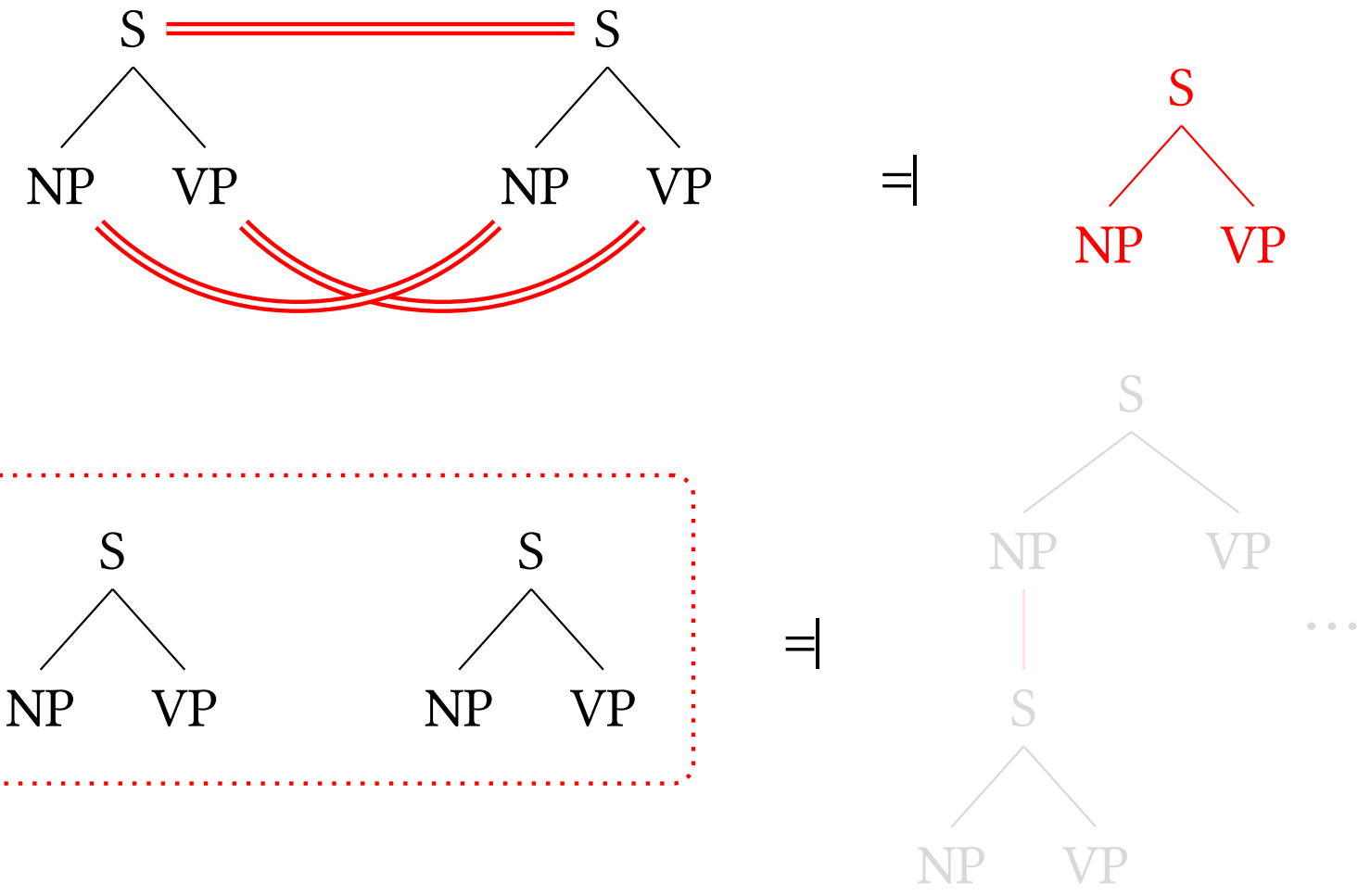
You may add edges but not nodes!



Metagrammars for LTAG: Example

Minimal model of tree descriptions

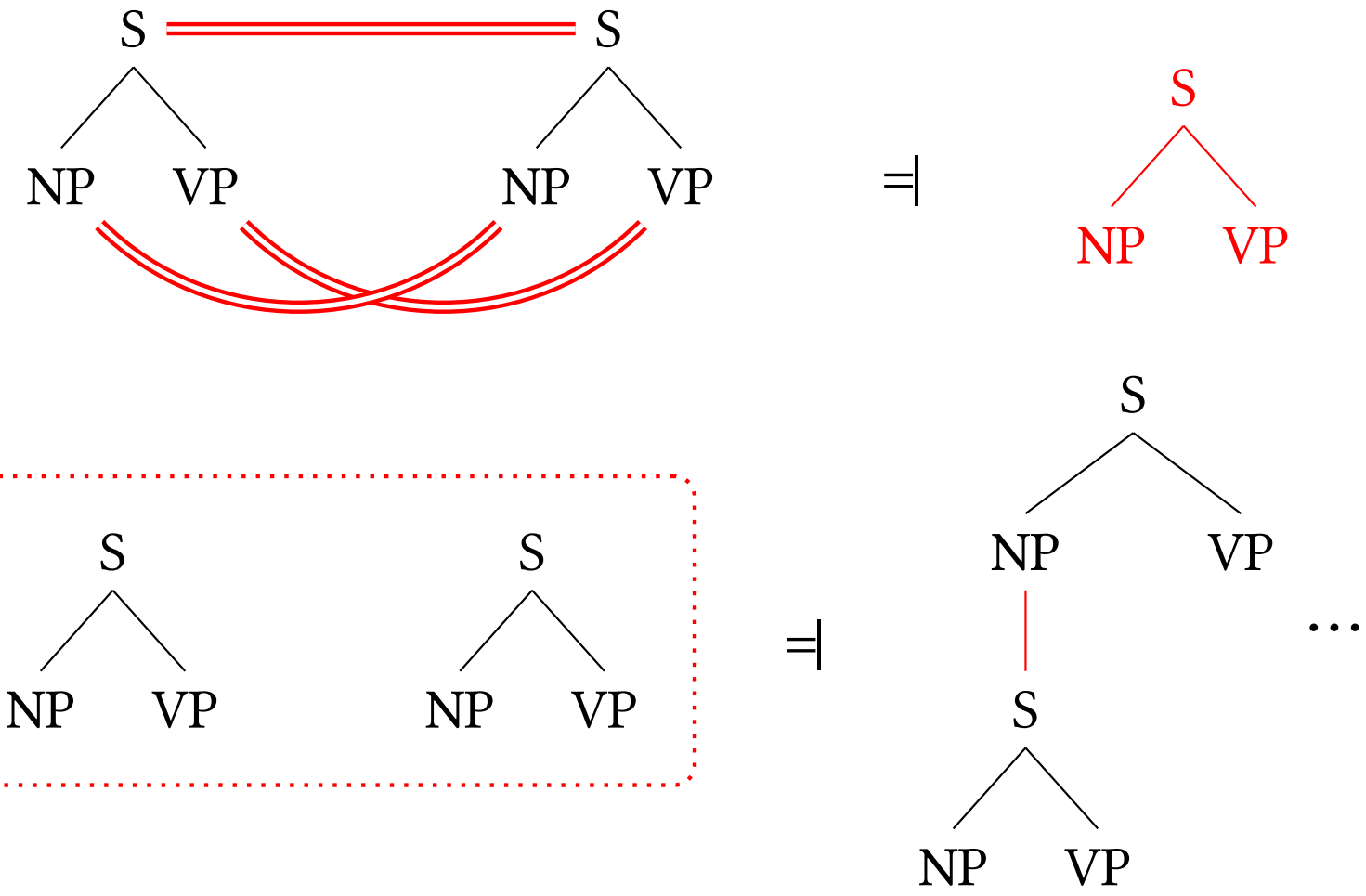
You may add edges but not nodes!



Metagrammars for LTAG: Example

Minimal model of tree descriptions

You may add edges but not nodes!

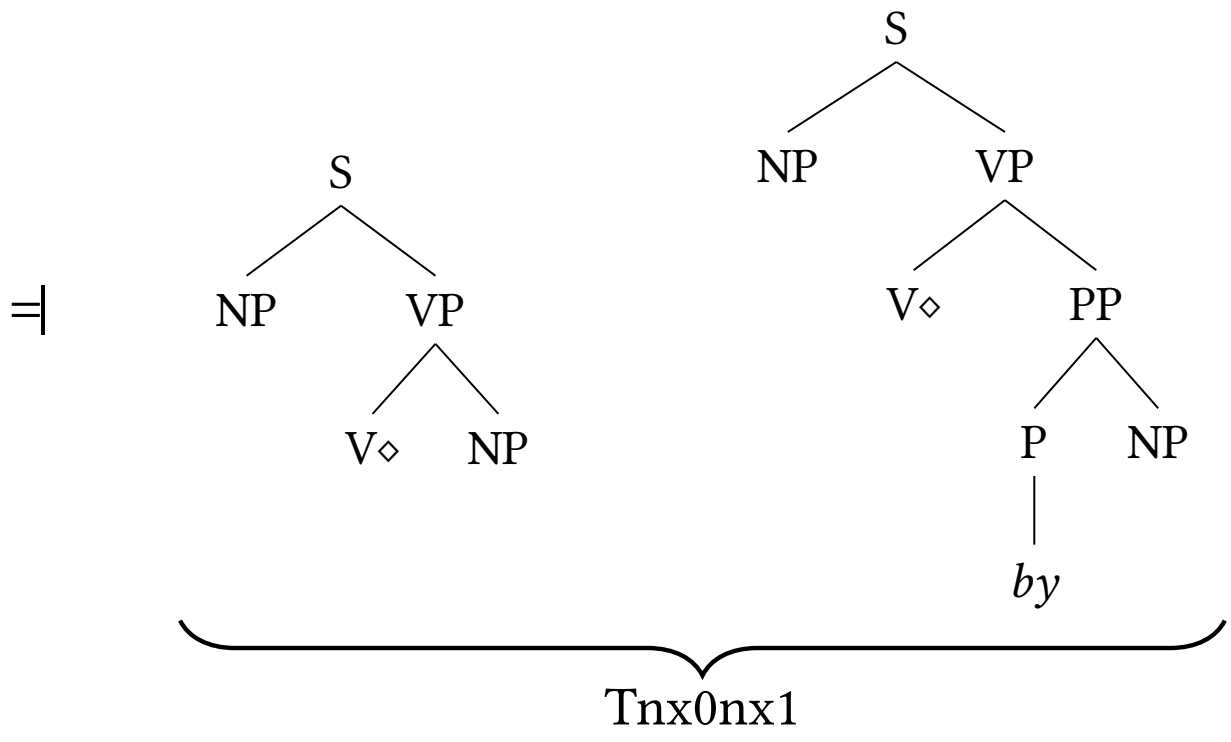


Metagrammars for LTAG: Properties

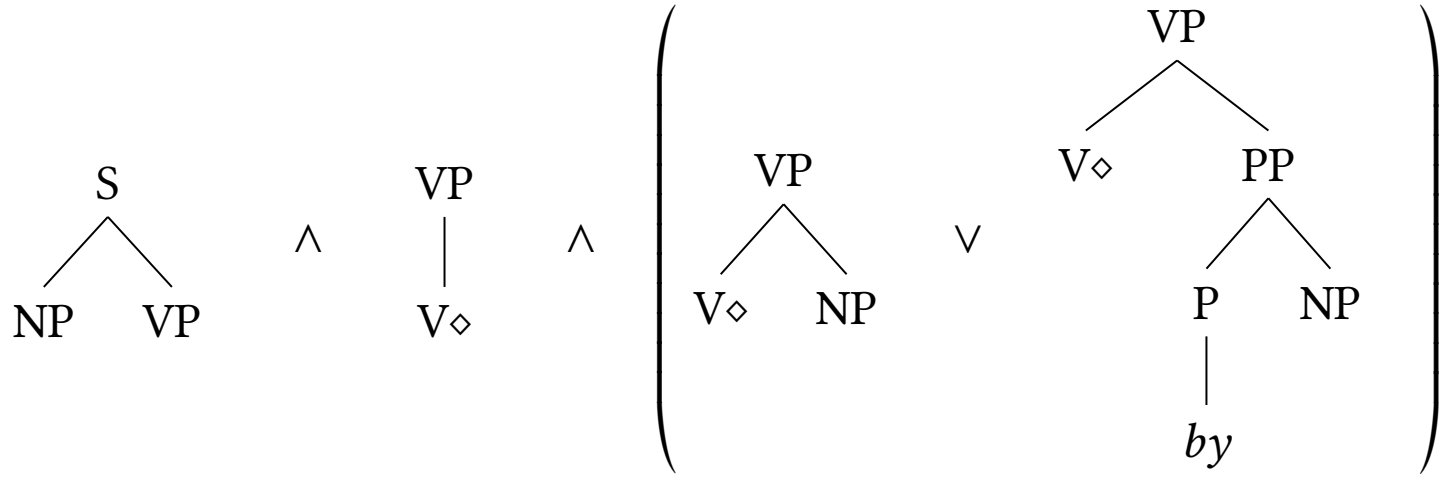
- no deletion, no copying, no recursion
- declarative, order insensitive
- The number of minimal models is finite.
- BUT: the number of minimal models can grow exponentially ($\mathcal{O}(n!)$) in terms of the number of described nodes.

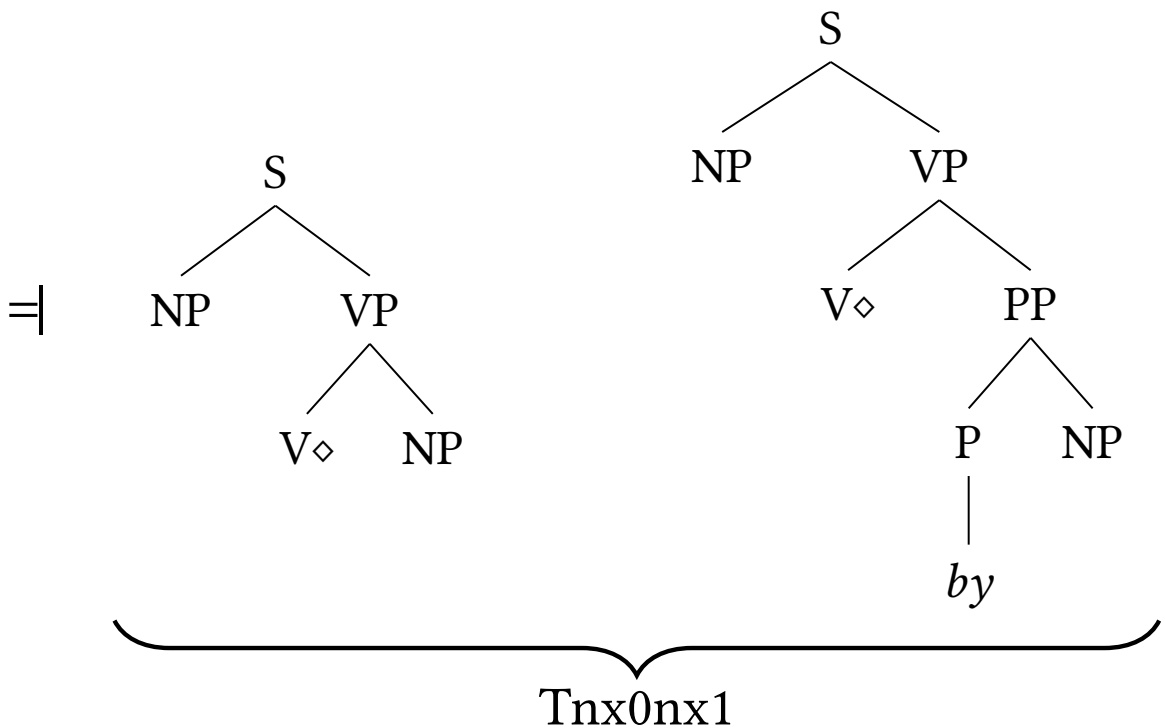
Does it suffice? How to express passivization?

Metagrammars for LTAG: Passivization

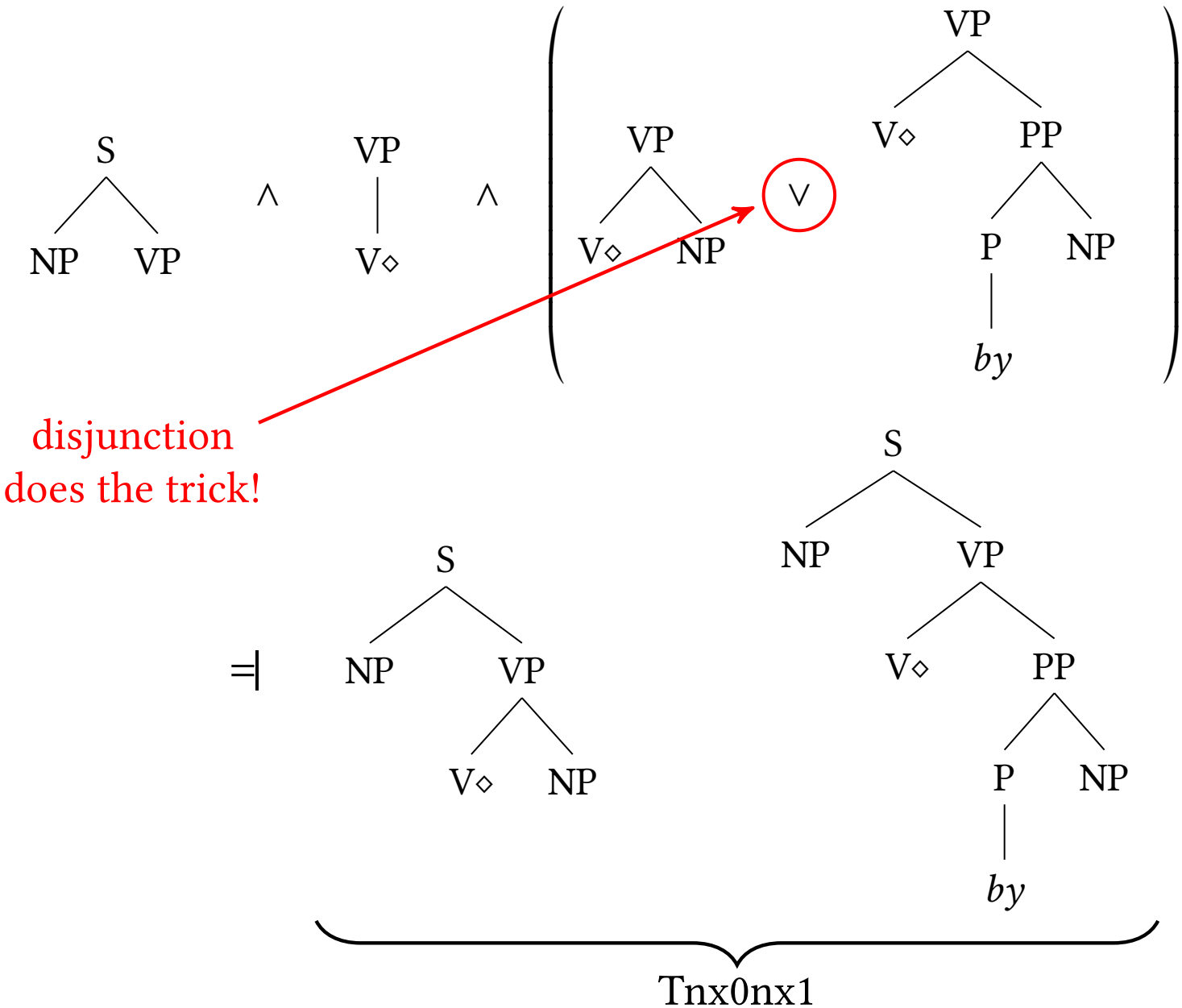


Metagrammars for LTAG: Passivization





Metagrammars for LTAG: Passivization



Metagrammar for LTAG: Classes

Tree descriptions are bundled into so-called **classes**:

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

$$Class := Name : Content$$

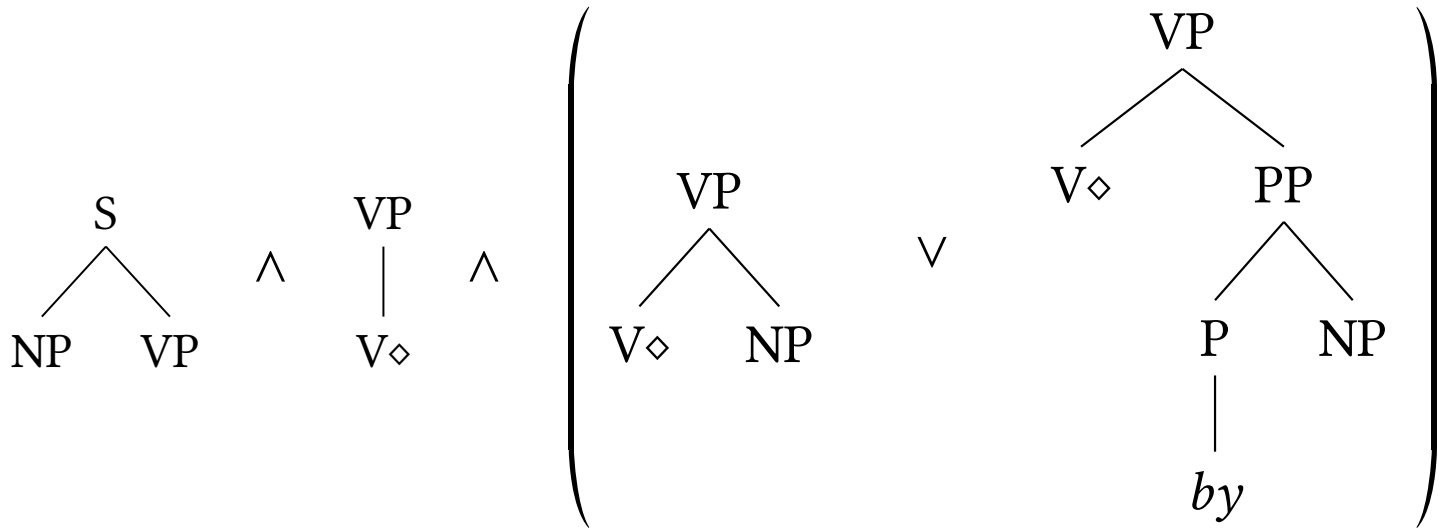
$$Content := \left(\begin{array}{l} Description \mid Name \mid \\ Content \vee Content \mid \\ Content \wedge Content \end{array} \right)$$

Upon instantiating/using a class:

- Node variables are replaced by fresh ones.
- Node variables are known to the instantiating class.
- The class name is replaced by the content in the instantiating class.

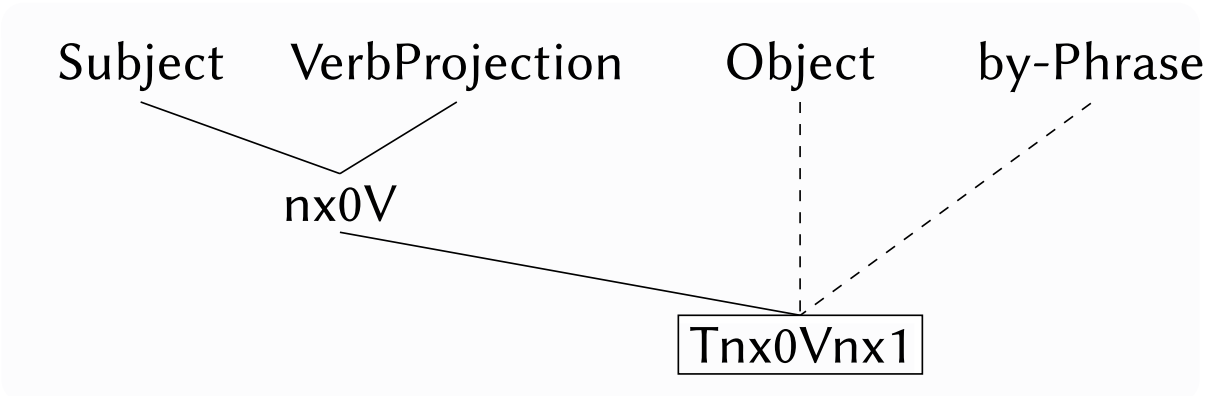
$\Rightarrow$ Classes can be reused!

Metagrammar for LTAG: Classes

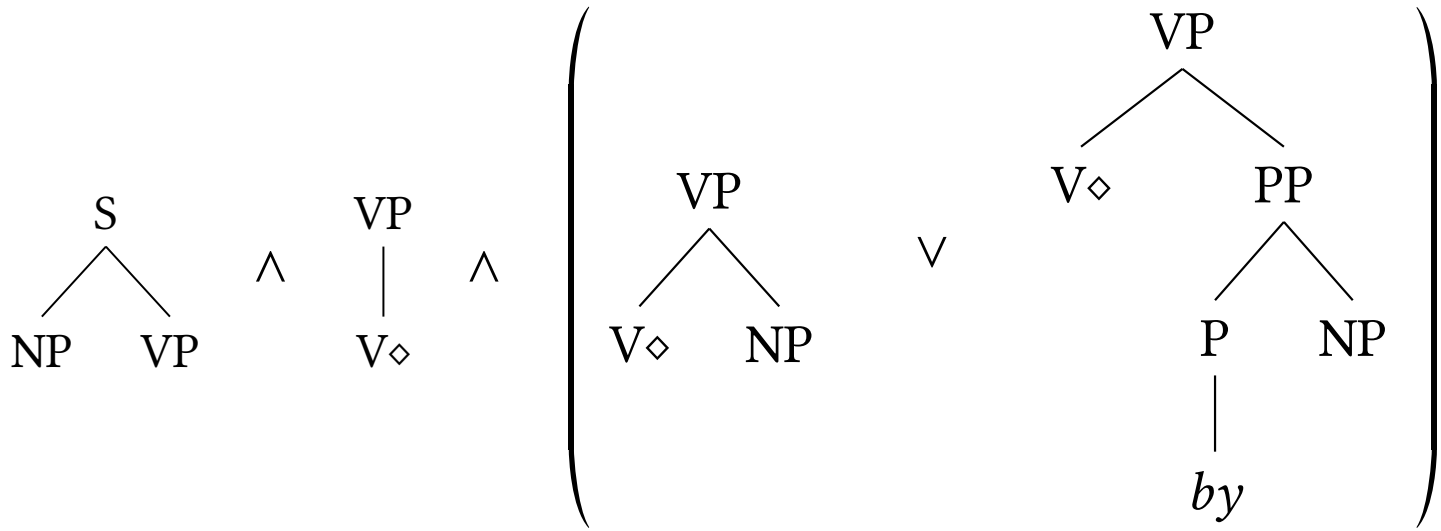
Tnx0Vnx1: 

Tnx0Vnx1: Subject ∧ VerbProjection ∧ (Object ∨ by-Phrase)

Tnx0Vnx1: nx0V ∧ (Object ∨ by-Phrase)

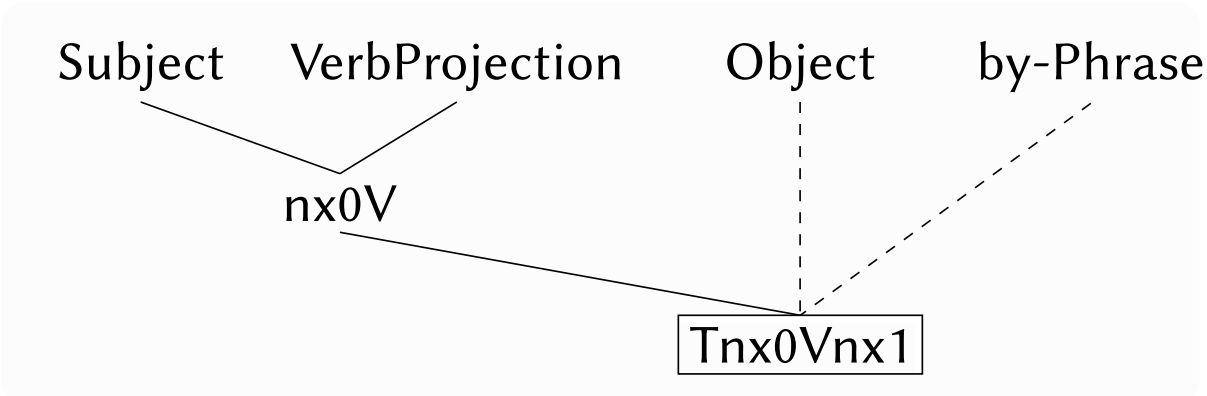


Metagrammar for LTAG: Classes

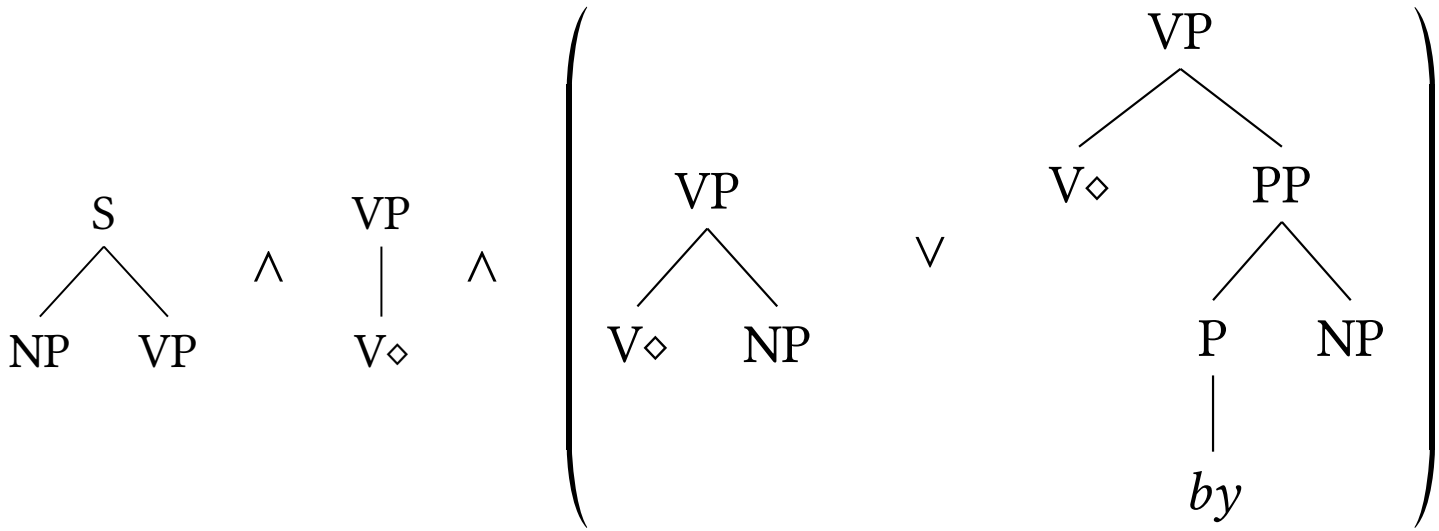
Tnx0Vnx1: 

Tnx0Vnx1: Subject ∧ VerbProjection ∧ (Object ∨ by-Phrase)

Tnx0Vnx1: nx0V ∧ (Object ∨ by-Phrase)

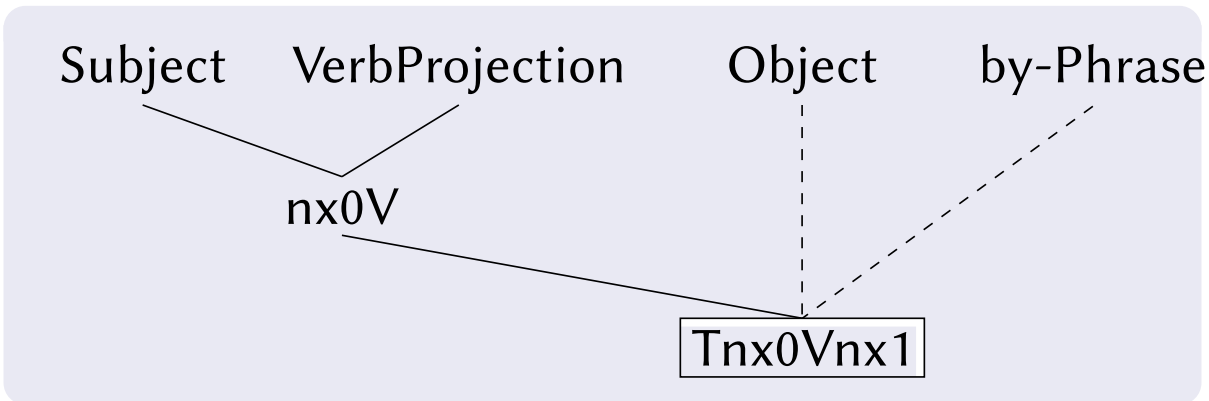


Metagrammar for LTAG: Classes

Tnx0Vnx1: 

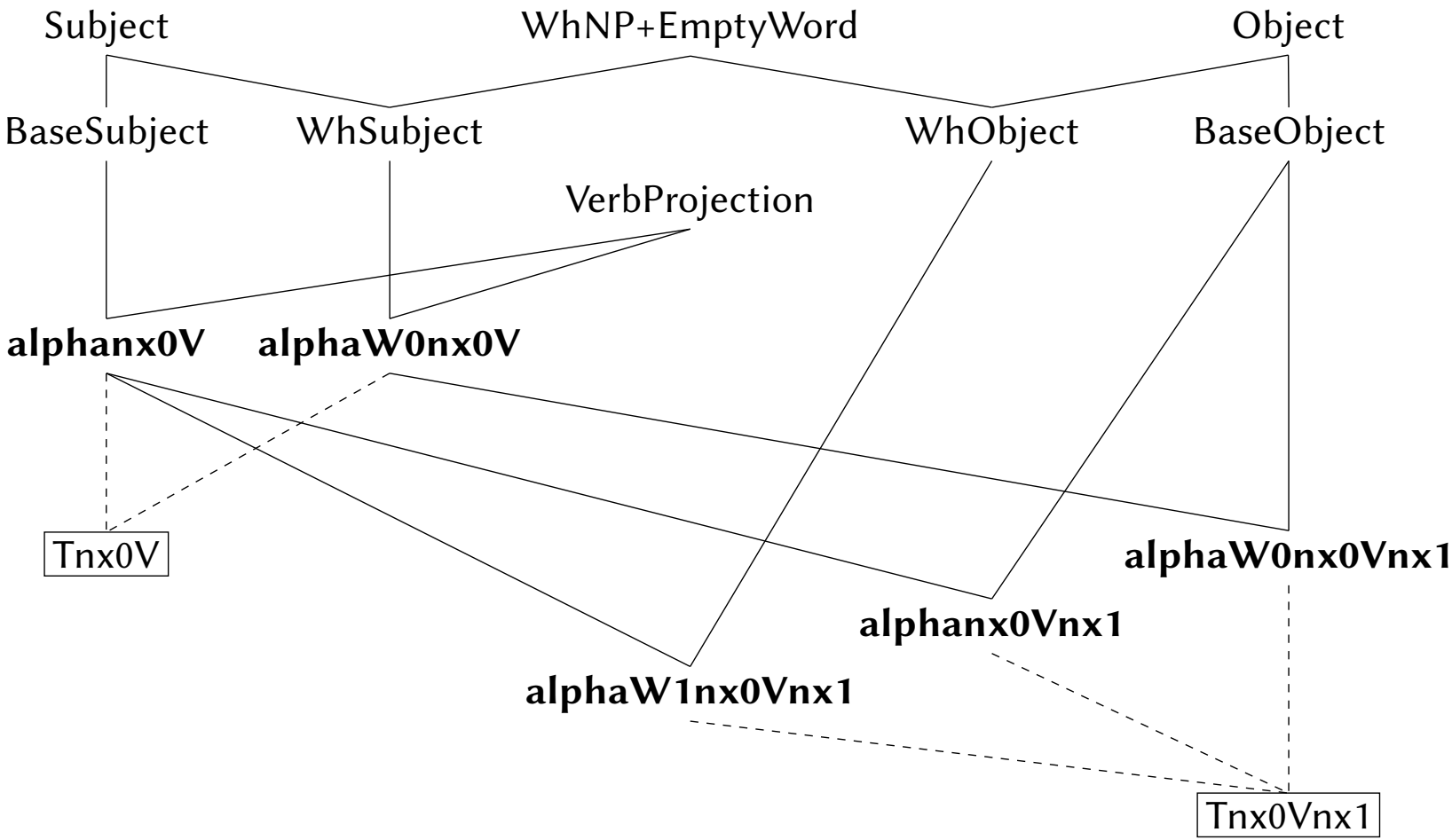
Tnx0Vnx1: Subject ∧ VerbProjection ∧ (Object ∨ by-Phrase)

Tnx0Vnx1: nx0V ∧ (Object ∨ by-Phrase)



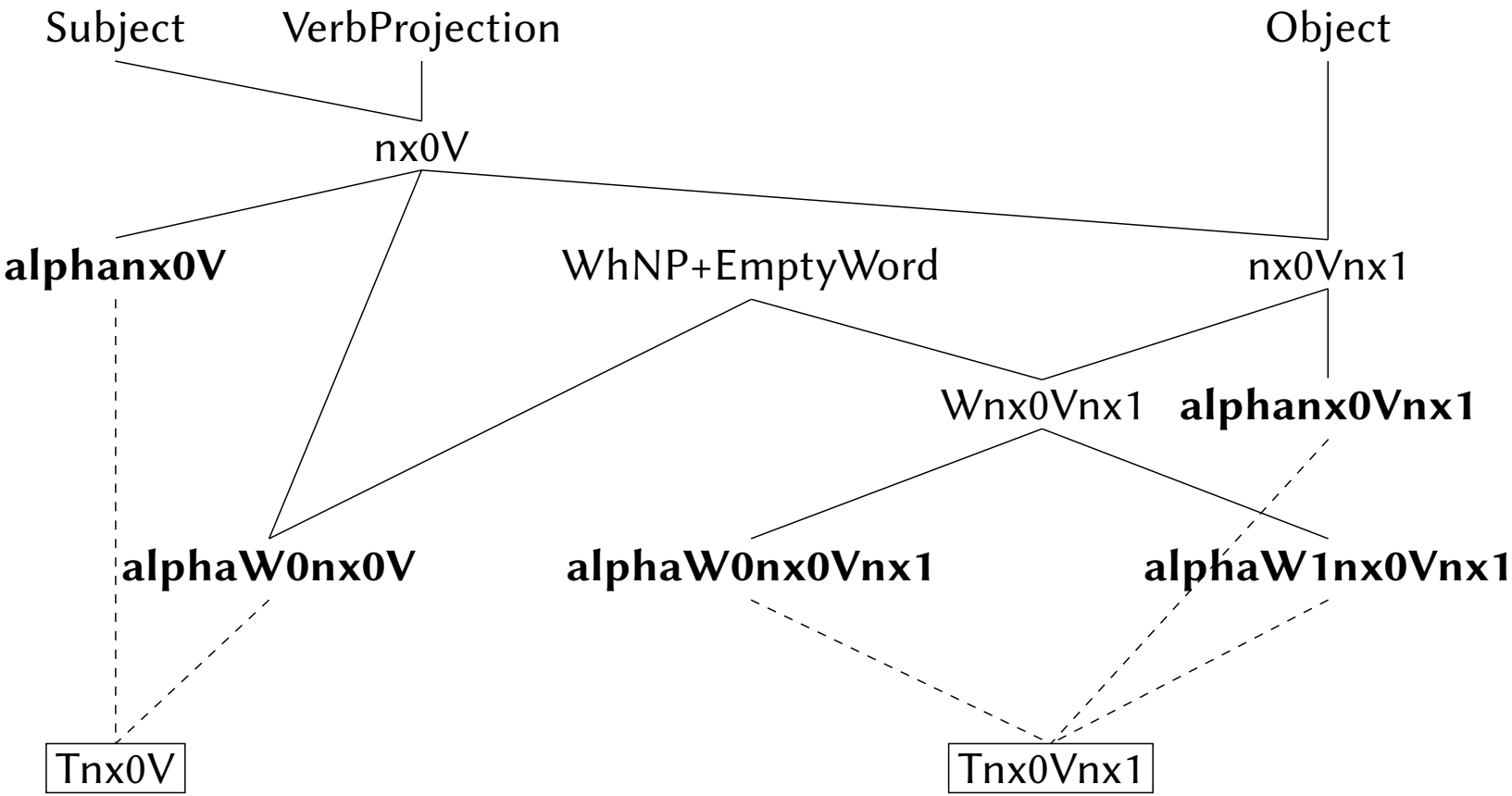
Metagrammar for LTAG: Class hierarchies

There are very many possible class hierarchies …



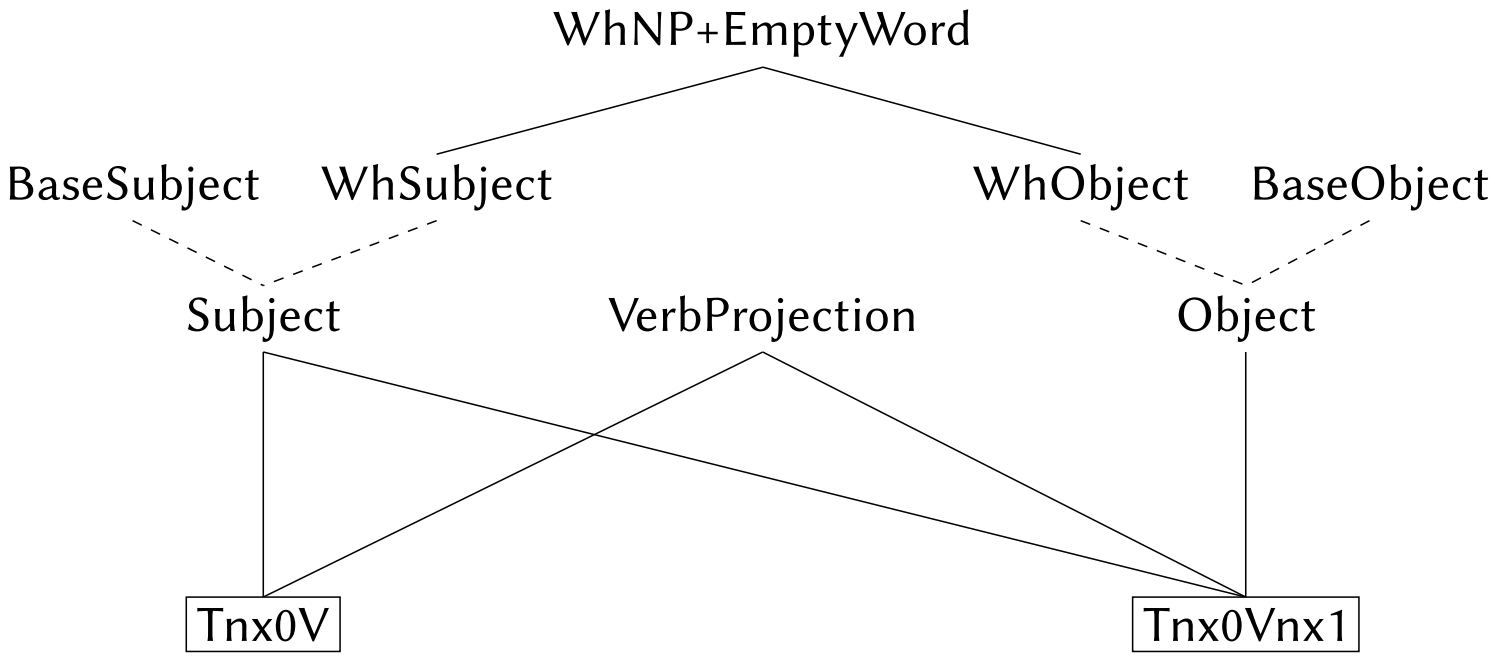
Metagrammar for LTAG: Class hierarchies

There are very many possible class hierarchies …



Metagrammar for LTAG: Class hierarchies

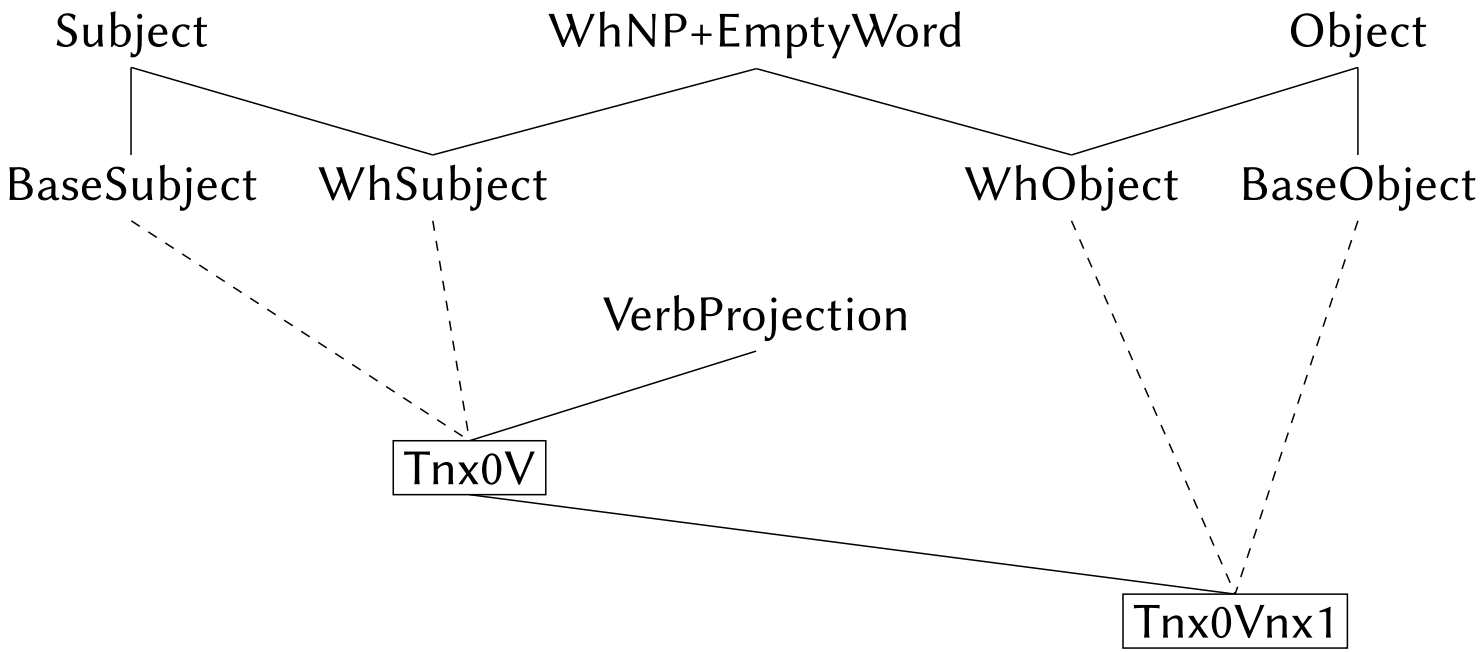
There are very many possible class hierarchies …



[1]

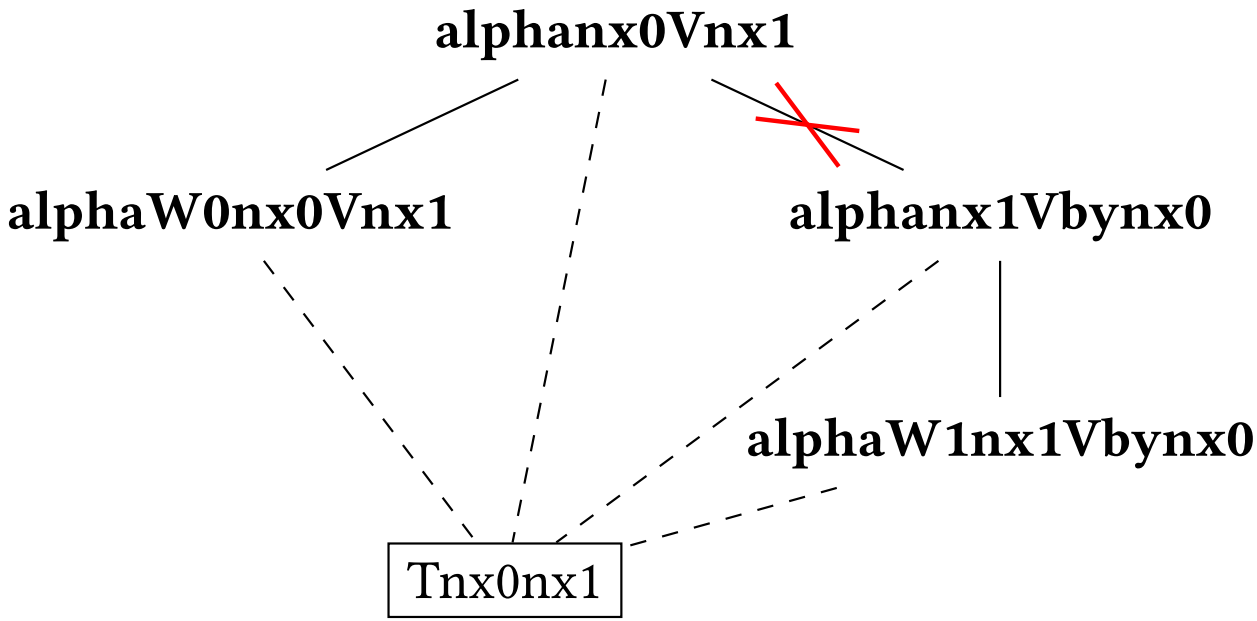
Metagrammar for LTAG: Class hierarchies

There are very many possible class hierarchies …



Metagrammar for LTAG: Class hierarchies

…but not everything is possible:



eXtensible Metagrammar (XMG): Background

- developed at LORIA, Nancy, LIFO, Orléans and HHU, Düsseldorf.[6]
- written in ~~Oz/Mozart~~ YAP and Python (as of XMG2)
- available at `dokufarm.phil.hhu.de/xmg`

Why "eXtensible" ?

- highly modularized[11]
- dimensions with dedicated description languages and compilers (`<syn>`, `<sem>`, `<frame>`, `<morph>`, ...)
- interface using shared variables

Some existing implementations using XMG:

- French: FrenchTAG[5]
- English: XTAG with XMG[1]
- German: GerTT[9]

eXtensible Metagrammar (XMG): Background

- developed at LORIA, Nancy, LIFO, Orléans and HHU, Düsseldorf.[6]
- written in ~~Oz/Mozart~~ YAP and Python (as of XMG2)
- available at `dokufarm.phil.hhu.de/xmg`

Why "eXtensible" ?

- highly modularized[11]
- dimensions with dedicated description languages and compilers (**<syn>**, **<sem>**, **<frame>**, **<morph>**, ...)
- interface using shared variables

Some existing implementations using XMG:

- French: FrenchTAG[5]
- English: XTAG with XMG[1]
- German: GerTT[9]

eXtensible Metagrammar (XMG): Background

- developed at LORIA, Nancy, LIFO, Orléans and HHU, Düsseldorf.[6]
- written in ~~Oz/Mozart~~ YAP and Python (as of XMG2)
- available at `dokufarm.phil.hhu.de/xmg`

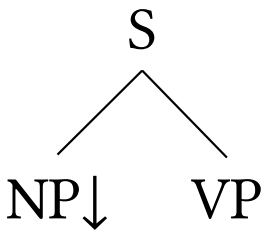
Why "eXtensible" ?

- highly modularized[11]
- dimensions with dedicated description languages and compilers (**<syn>**, **<sem>**, **<frame>**, **<morph>**, ...)
- interface using shared variables

Some existing implementations using XMG:

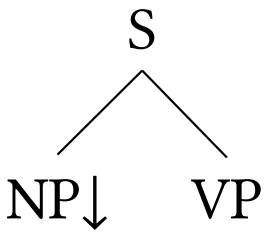
- French: FrenchTAG[5]
- English: XTAG with XMG[1]
- German: GerTT[9]

eXtensible Metagrammar (XMG): Example



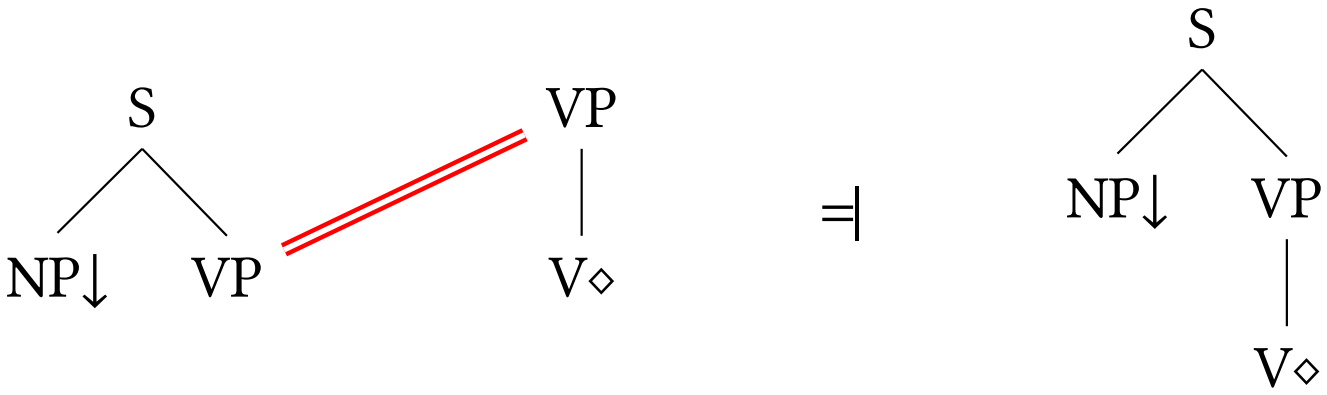

```
1  class Subject
2  export ?S
3  declare ?S ?NP ?VP
4  { <syn>{
5      node ?S [cat=s]{
6          node ?NP (mark=subst) [cat=np]
7          node ?VP [cat=vp]
8          }
9      }
10 }
```

eXtensible Metagrammar (XMG): Example



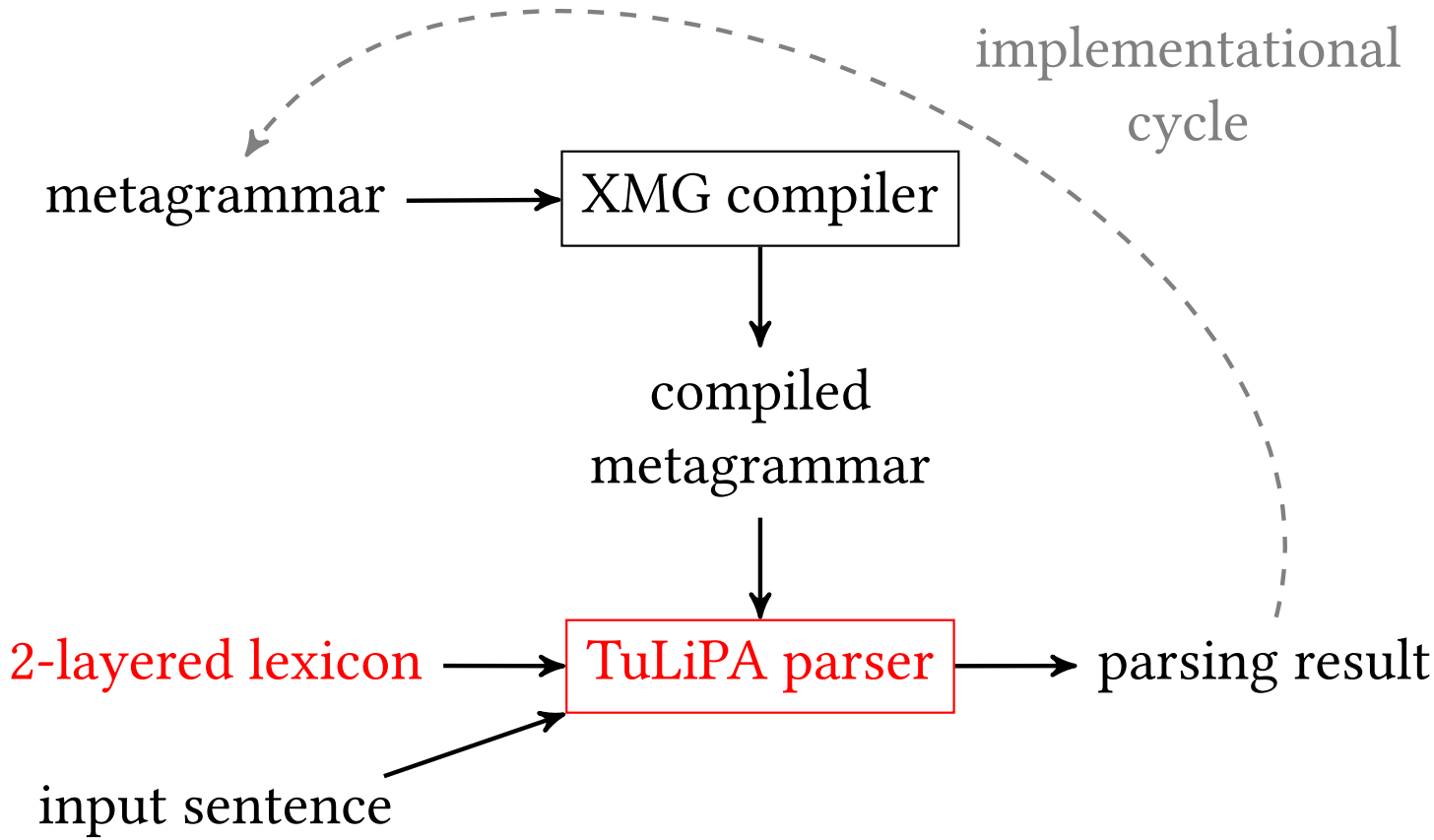

```
1  class Subject
2  export ?S
3  declare ?S ?NP ?VP
4  { <syn>{
5      node ?S [cat=s];
6      node ?NP (mark=subst) [cat=np];
7      node ?VP [cat=vp];
8      ?S -> ?NP;
9      ?S -> ?VP;
10     ?NP >> ?VP
11     }
12 }
```

eXtensible Metagrammar (XMG): Example

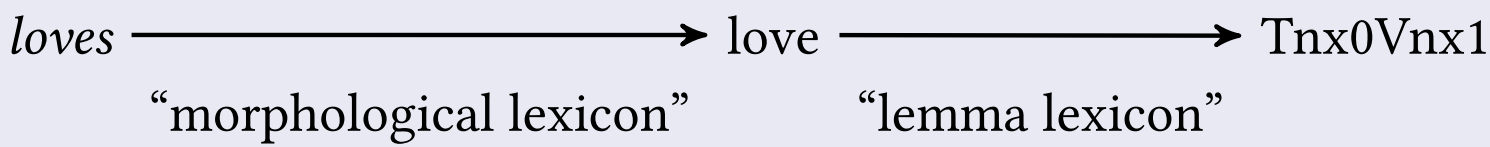



```
1  class alphanx0v
2  import VerbProjection[]
3  declare ?Subj
4  {
5    ?Subj = Subject[];
6    |%\color{red}?Subj.?VP = ?VP %|
7  }
```

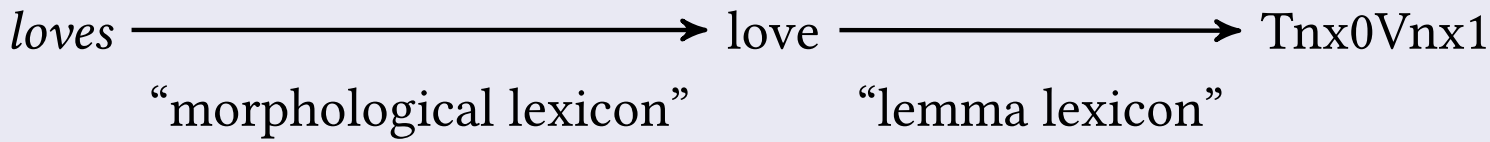
Lexicon and parser



Lexicon and parser: A 2-layered lexicon



Lexicon and parser: A 2-layered lexicon



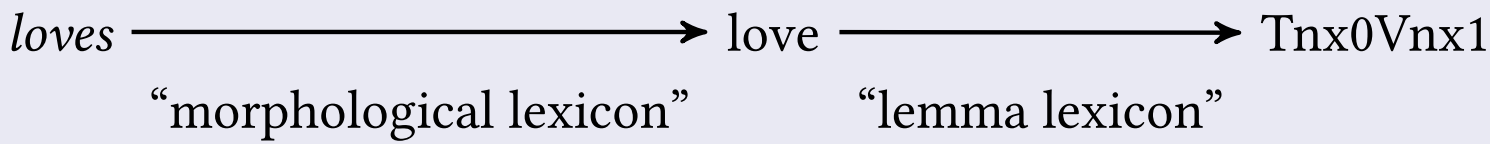
Morphological lexicon

maps an (inflected) token to some base form (= lemma), while preserving morphological information in a feature structure.

loves	love	[pos=v; num=sing; pers=3;]
Peter	Peter	[pos=n; num=sing; pers=3; case=nom|acc;]

Interface with tree templates: Feature unification during lexical insertion

Lexicon and parser: A 2-layered lexicon



Lemma lexicon

maps a lemma onto tree tuple families, while also containing selectional restrictions (e.g., case assignment).

```
*ENTRY: love
*CAT: v
*SEM:
*ACC: 1
*FAM: Tnx0Vnx1
*FILTERS: []
*EX:
*EQUATIONS:
NParg1 -> case = nom
NParg2 -> case = acc
*COANCHORS:
```

Interface with tree templates:
EQUATIONS → nodes of tree templates
FILTERS → selection of tree templates

Lexicon and parser: The TuLiPA parser

(Parmentier et al.)[10]

TuLiPA

- Tübingen Linguistic Parsing Architecture (TuLiPA)
- uses Range Concatenation Grammar (RCG) as a pivot formalism.

Components:

1. TAG-to-RCG converter (on-line)
2. RCG parser → RCG derivation forest → TAG derivation forest
3. Parse viewer (derived tree, derivation tree, dependency view, semantic representation)

Availability of TuLiPA:
written in Java and released under the GNU GPL
(`http://sourcesup.cru.fr/tulipa/`)

[1] Alahverdzhieva, Katya. 2008. *XTAG using XMG. A core Tree-Adjoining Grammar for English.* University of Nancy 2 / University of Saarland Master's Thesis. `http://homepages.inf.ed.ac.uk/s0896251/pubs/msc-sb2008.pdf`.

[2] Becker, Tilman. 1994. *Hytag: a new type of tree adjoining grammars for hybrid syntactic representations of free word order languages.* Universität des Saarlandes dissertation. `http://www.dfki.de/~becker/becker.diss.ps.gz`.

[3] Becker, Tilman. 2000. Patterns in metarules for TAG. In Anne Abeillé & Owen Rambow (eds.), *Tree Adjoining Grammars: Formalisms, linguistic analyses and processing* (CSLI Lecture Notes 107), 331–342. Stanford, CA: CSLI Publications.

[4] Candito, Marie-Hélène. 1996. A principle-based hierarchical representation of LTAGs. In *Proceedings of the 16th international Conference on Computational Linguistics (COLING 96)*. Copenhagen. `http://aclweb.org/anthology-new/C/C96/C96-1034.pdf`.

[5] Crabbé, Benoît. 2005. *Représentation informatique de grammaires d'arbres fortement lexicalisées: Le cas de la grammaire d'arbres adjoints.* Université Nancy 2 dissertation.

[6] Crabbé, Benoit, Denys Duchier, Claire Gardent, Joseph Le Roux & Yannick Parmentier. 2013. XMG: eXtensible MetaGrammar. *Computational Linguistics* 39(3). 1–66. `http://hal.archives-ouvertes.fr/hal-00768224/en/`.

[7] Dowty, David R. 1979. *Word meaning and Montague Grammar.* Reprinted 1991 by Kluwer Academic Publishers. Dordrecht: D. Reidel Publishing Company.

[8] Gazdar, Gerald. 1981. Unbounded dependencies and coordinated structure. *Linguistic Inquiry* 12. 155–182.

[9] Kallmeyer, Laura, Timm Lichte, Wolfgang Maier, Yannick Parmentier & Johannes Dellert. 2008. Developing a TT-MCTAG for German with an RCG-based parser. In European Language Resources Association (ELRA) (ed.), *Proceedings of the sixth international Conference on Language Resources and Evaluation (LREC'08)*, 28–30. Marrakech, Morocco.

[10] Parmentier, Yannick, Laura Kallmeyer, Wolfgang Maier, Timm Lichte & Johannes Dellert. 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)*, 121–128. Tübingen, Germany.

[11] Petitjean, Simon. 2014. *Génération Modulaire de Grammaires Formelles*. Orléans, France: Université d'Orléans Thèse de Doctorat. `https://tel.archives-ouvertes.fr/tel-01163150/`.

[12] Prolo, Carlos A. 2002. Generating the XTAG English grammar using metarules. In *Proceedings of the 19th international Conference on Computational Linguistics (COLING 2002)*, 814–820. Taipei. Taiwan.

[13] Ristad, Eric Sven. 1987. Revised General Phrase Structure Grammar. In *Proceedings of the 25th annual meeting of the Association for Computational Linguistics*, 243–250. Stanford, CA. `http://www.aclweb.org/anthology/P87-1034`.

[14] Uszkoreit, Hans & Stanley Peters. 1987. On some formal properties of metarules. English. In Walter J. Savitch, Emmon Bach, William Marsh & Gila Safran-Naveh (eds.), *The formal complexity of natural language* (Studies in Linguistics and Philosophy 33), 227–250. Dordrecht, The Netherlands: D. Reidel Publishing. `http://dx.doi.org/10.1007/978-94-009-3401-6_9`.

[15] Xia, Fei. 2001. *Automatic grammar generation from two different perspectives*. University of Pennsylvania dissertation. `http://faculty.washington.edu/fxia/papers_from_penn/thesis.pdf`.

[16] XTAG Research Group. 2001. *A Lexicalized Tree Adjoining Grammar for English*. Tech. rep. Philadelphia, PA: Institute for Research in Cognitive Science, University of Pennsylvania.