Grammar Implementation with TAG

TuLiPA and the lexion

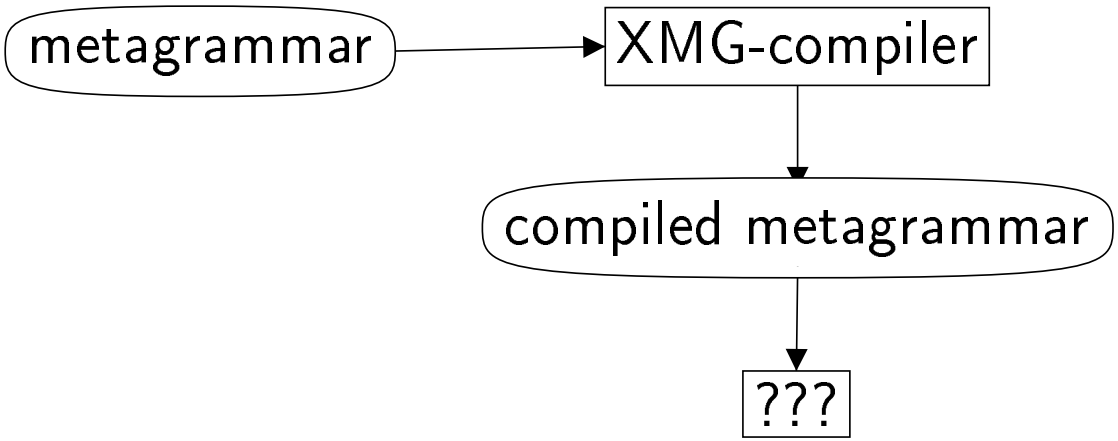
Timm Lichte

HHU Düsseldorf

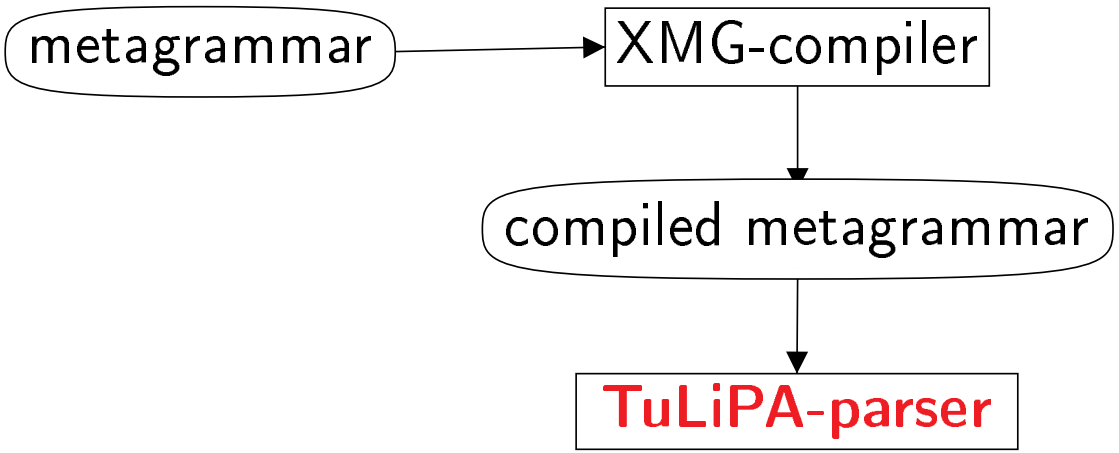
SS 2011

29.06.2011

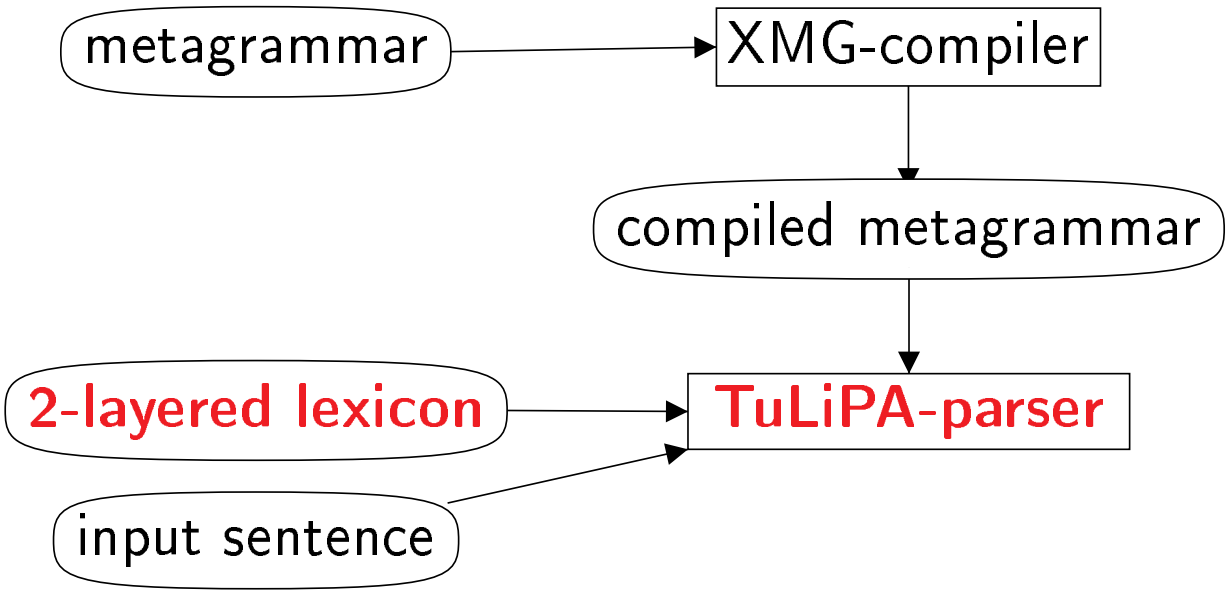
The situation



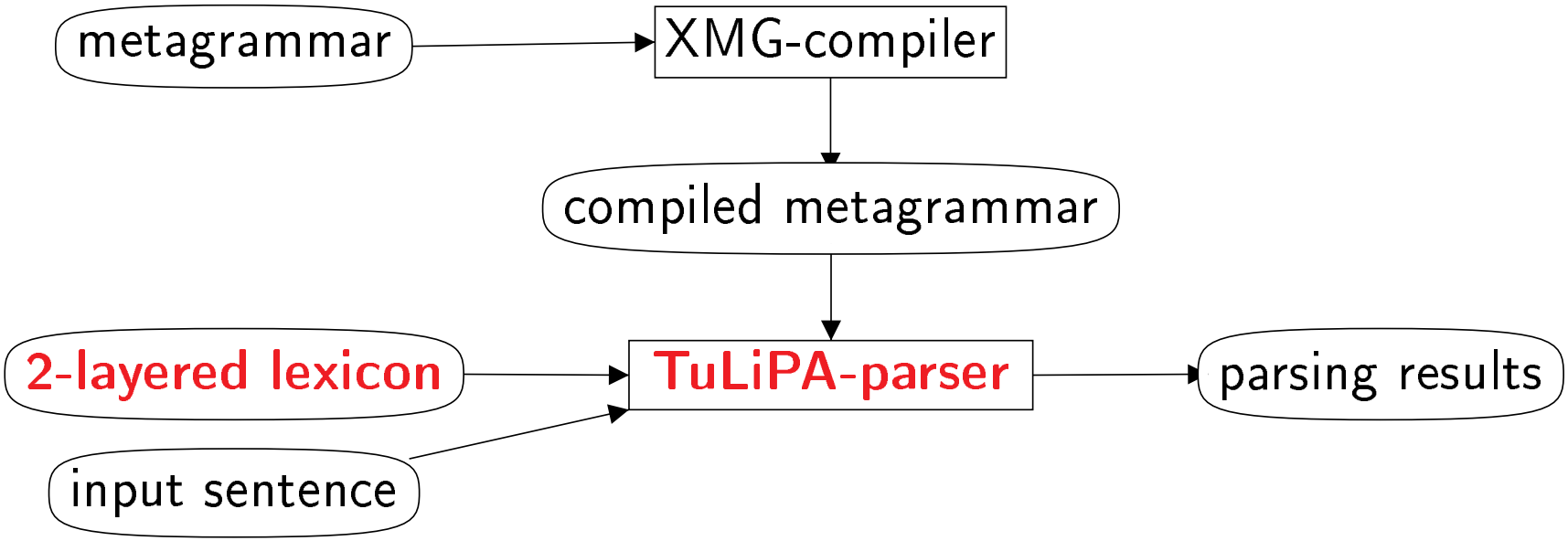
The missing pieces



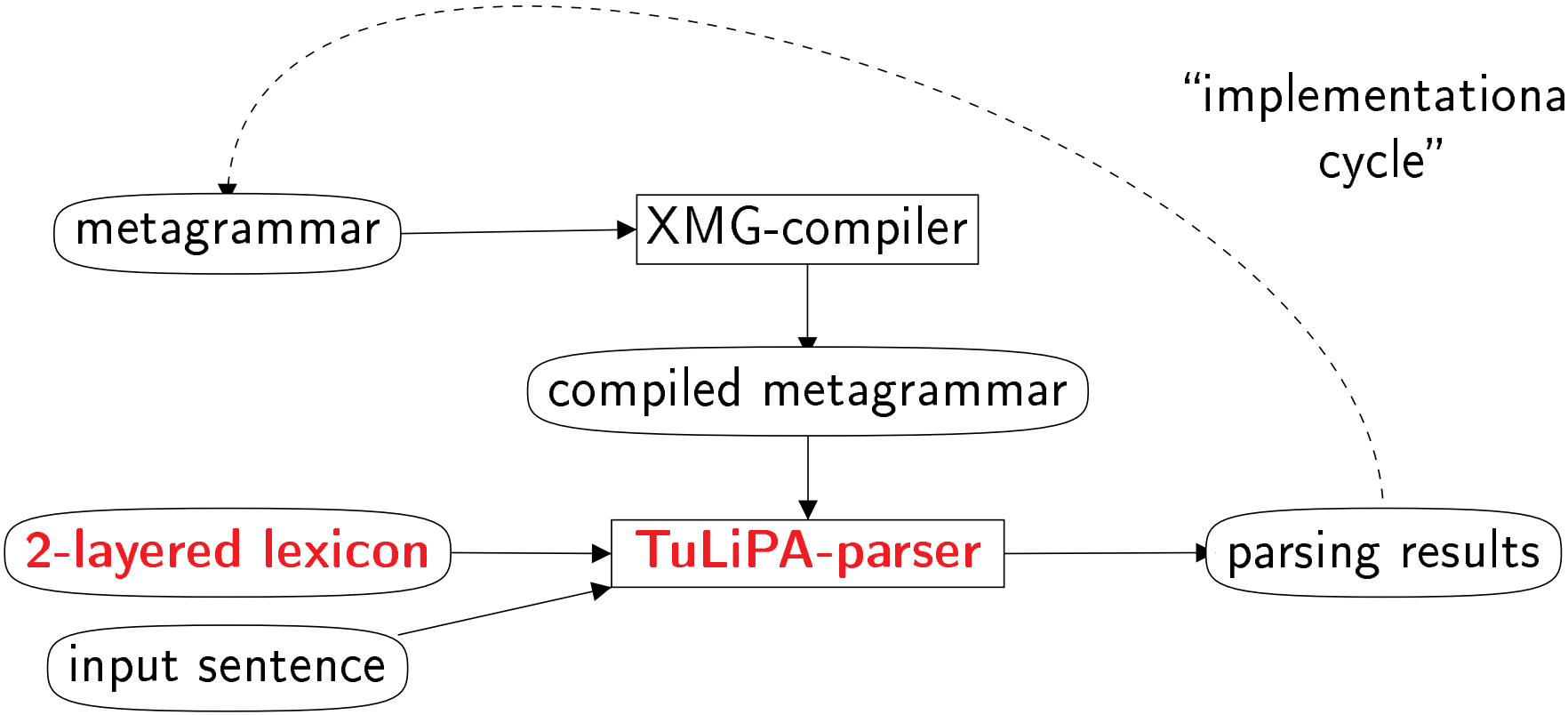
The missing pieces



The missing pieces



The missing pieces



Tübingen Linguistic Parsing Architecture (TuLiPA)

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/`)

Range Concatenation Grammar (RCG)

$$A(X_1X_2, X_3) \rightarrow B(X_2)\ C(X_1, X_3)$$

A, B, C: predicates
X_1, X_2, X_3: range variables (instantiated with substrings of the string)

Example:

$S(X_1X_2) \rightarrow NP(X_1)\ VP(X_2)$
$VP(X_3X_4) \rightarrow V(X_3)\ NP(X_4)$
$NP(Peter) \rightarrow \epsilon$
$NP(Susan) \rightarrow \epsilon$
$V(loves) \rightarrow \epsilon$

Simple RCG

- **non-combinatorial:** each argument on the RHS of a clause consists of a unique variable!

  NOT: $A(XY, Z) \rightarrow B(Y)\ C(XZ)$

- **linear:** each variable appears at most once in the LHS or RHS of a clause!

  NOT: $A(XYX, Z) \rightarrow B(Y)\ C(X, Z)$

- **non-erasing:** every variable in the LHS of a clause is also in its RHS!

  NOT: $A(XY, Z) \rightarrow B(Y)\ C(X)$

TuLiPA - Why RCG?

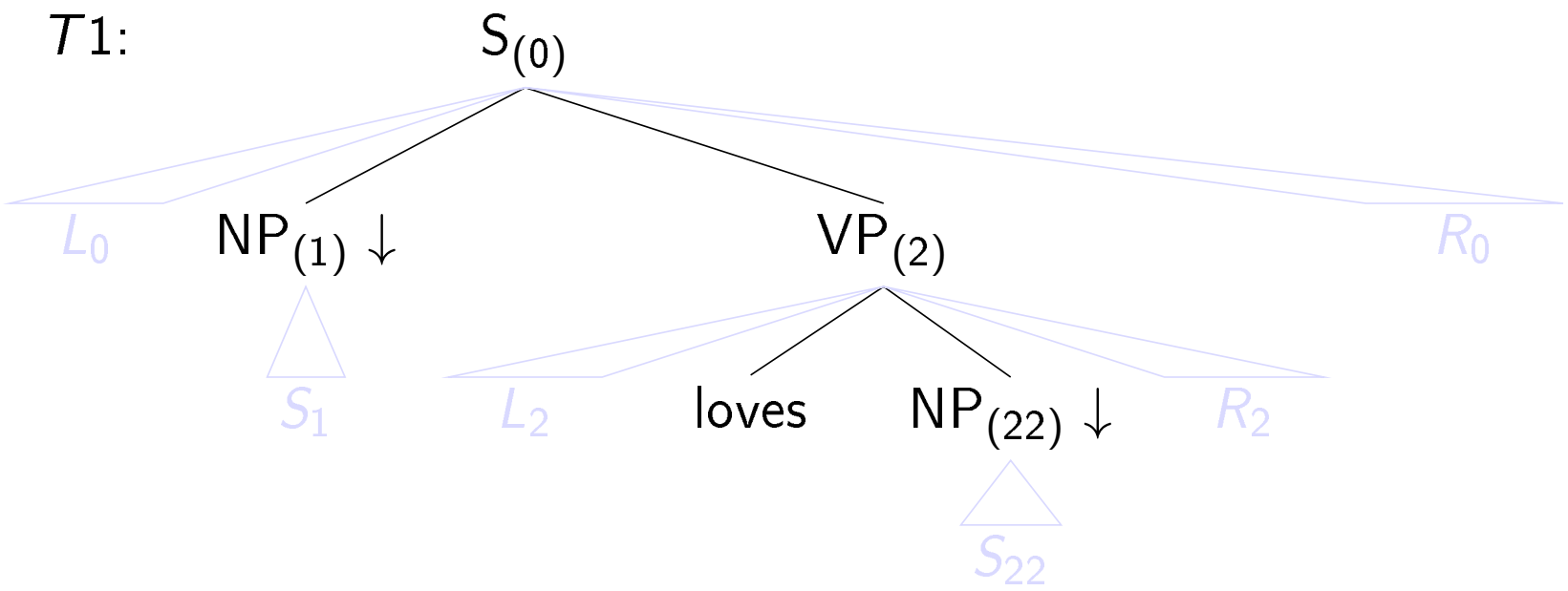
RCG is useful, because:

- it has attractive formal properties (polynomially parsable, full expressive power of MCS-languages);
- there exist parsing algorithms.

$\Rightarrow$ Parser can be reused for other mildly context-sensitive formalisms!

NB: RCG properly includes MCS. We use a restricted RCG, called *simple RCG*, that is included in MCS.

TuLiPA - From TAG to RCG

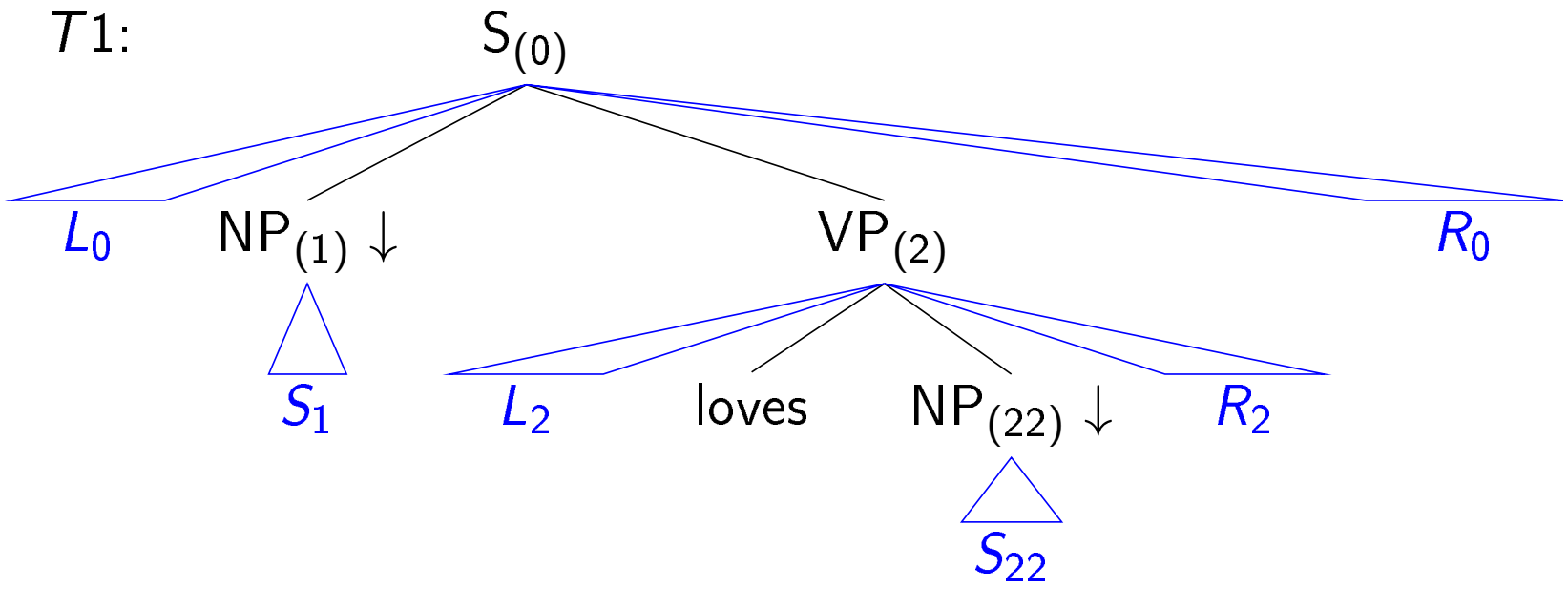


$$T1(L_0\ S_1\ L_2\ loves\ S_{22}\ R_2\ R_0) \rightarrow$$
$$adj_{S_0}(L_0, R_0)\ \ sub_{NP_1}(S_1)\ \ adj_{VP_2}(L_2, R_2)\ \ sub_{NP_{22}}(S_{22})$$

$$adj_{S_0}(X, Y) \rightarrow T2(X, Y)$$

...

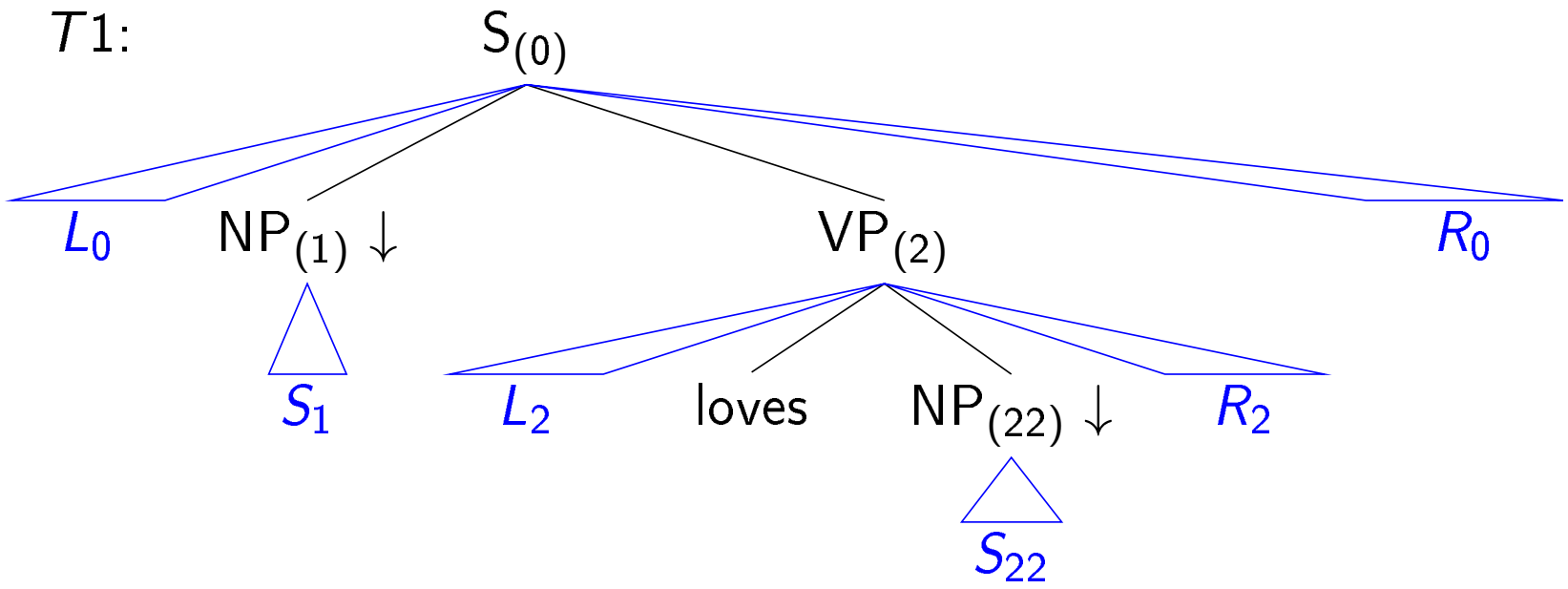
TuLiPA - From TAG to RCG



$$T1(L_0\ S_1\ L_2\ loves\ S_{22}\ R_2\ R_0) \rightarrow$$
$$adj_{S_0}(L_0, R_0)\ \ sub_{NP_1}(S_1)\ \ adj_{VP_2}(L_2, R_2)\ \ sub_{NP_{22}}(S_{22})$$

$$adj_{S_0}(X, Y) \rightarrow T2(X, Y)$$

...

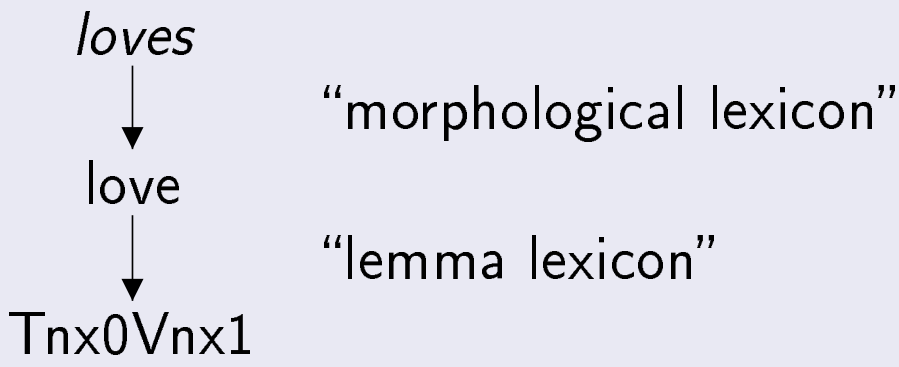


$T1(L_0\ S_1\ L_2\ loves\ S_{22}\ R_2\ R_0) \rightarrow$
$adj_{S_0}(L_0, R_0)\ \ sub_{NP_1}(S_1)\ \ adj_{VP_2}(L_2, R_2)\ \ sub_{NP_{22}}(S_{22})$

$adj_{S_0}(X, Y) \rightarrow T2(X, Y)$

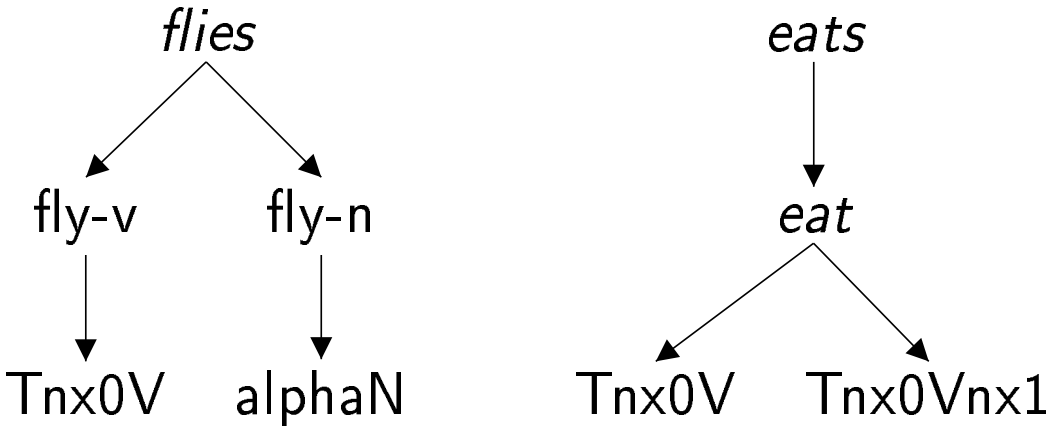
...

A 2-layered lexicon



The morphological lexicon and the lemma lexicon are compiled using `lexConverter` (as a part of LEX2ALL).

A 2-layered lexicon - 2 sources for ambiguity



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

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 $\rightarrow$ nodes of tree templates (via name property)

FILTERS $\rightarrow$ tree templates (via interface expressions)