SemTAG - a platform for Semantic Construction with Tree Adjoining Grammars

Yannick Parmentier
parmenti@loria.fr

Langue Et Dialogue Project

LORIA – Nancy Universities – France

Emmy Noether Project – SFB 441 – Tübingen
18 January 2007

Introduction

- No clear consensus about a syntax / semantics interface for TAG
- No wide-coverage TAG for French including syntactic and semantic information
- Goals of the SemTAG system:
  1. to provide with an environment to implement real-size TAGs equipped with a syntax / semantics interface,
  2. to build underspecified semantic representations of sentences using such a grammar.

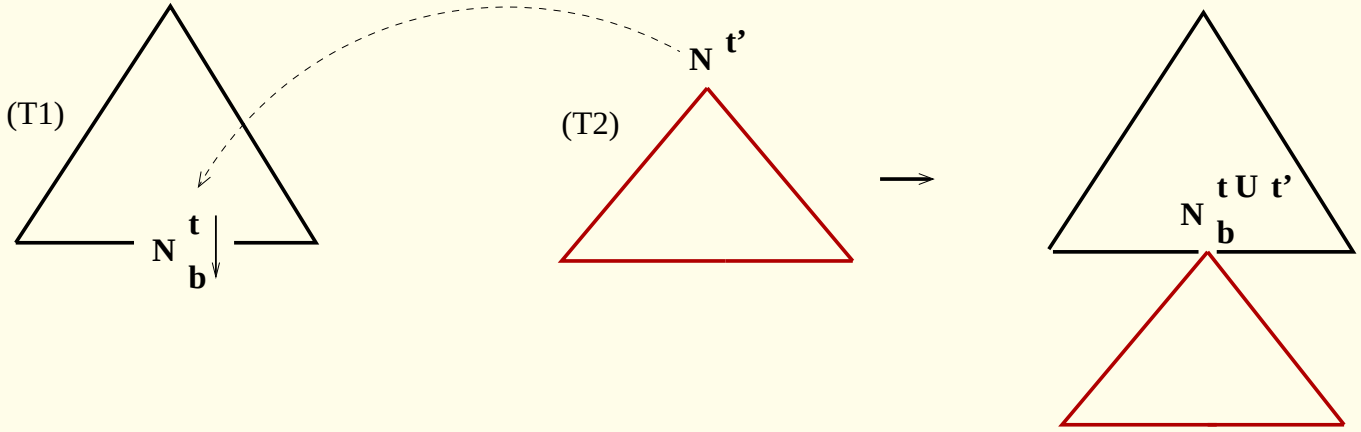
Outline

Part 1. Grammar development

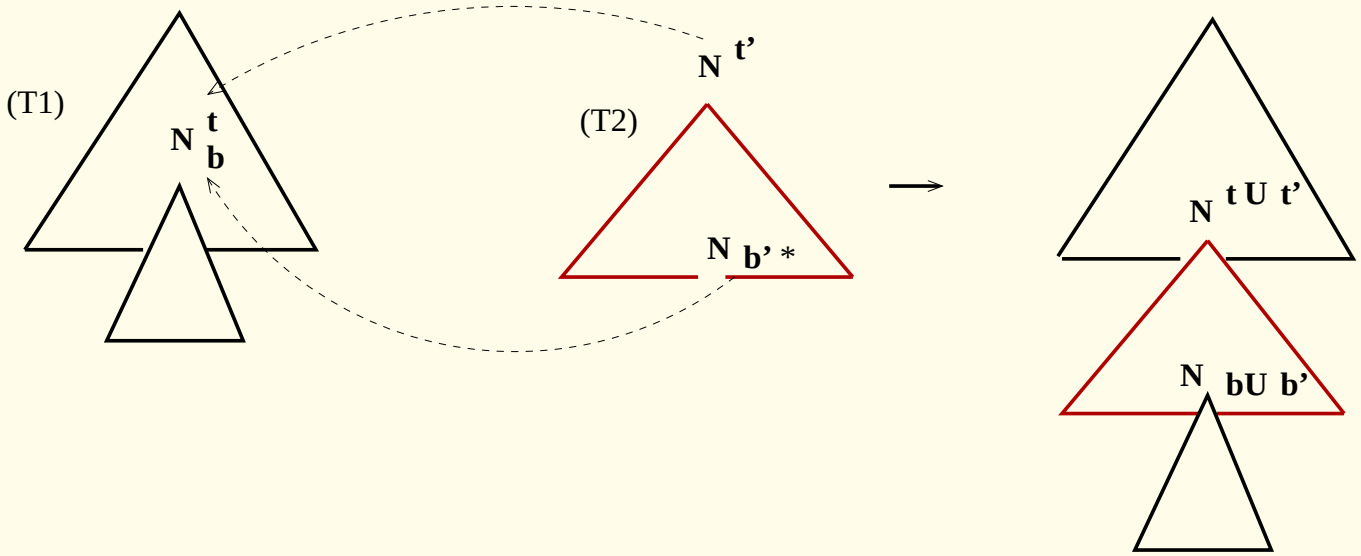
Lexicalised Tree Adjoining Grammars

- syntactic support : **Feature-Based Lexicalised TAG**
  - a set of elementary trees where nodes are labelled with *feature structures (FS)*
  - each elementary tree is associated with at least one *lexical anchor (Lexicalisation)*
  - two operations for combining trees: *adjunction* and *substitution* including *unification* on the FS

1) substitution



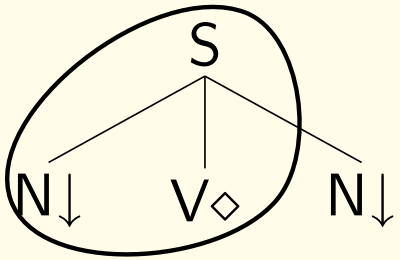
2) adjunction



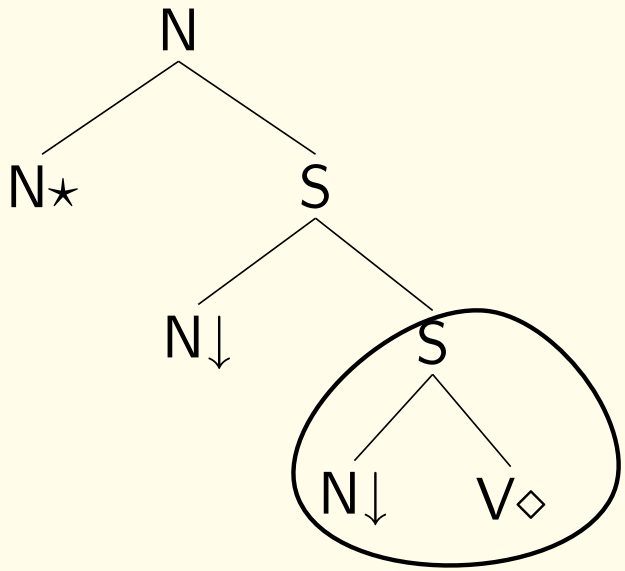
NB: at the end of the derivation, unification of *top* and *bot* FS at each node

LTAG redundancy

- TAG formalism is used in its lexicalised version for parsing.
- Remarks: many trees share common fragments + a given tree is associated with many lexical items (cf lexicalisation)



Jean *mange une pomme*



La pomme que **Jean** *mange*

Metagrammars for LTAG

- Related problems: huge redundancy making the design and maintnance of the grammar difficult.

  e.g. what happens if the agreement representation is modified during grammar development ?

- The MetaGrammar approach:
  - describing the trees of a grammar as combinations of elementary tree fragments,
  - capturing linguistic generalisations through these abstractions.

eXtensible MetaGrammar (1 / 2)

- Description of the grammar trees using an expressive and relatively intuitive language.
- MetaGrammar $\equiv$ manipulation of elementary tree fragments using a control language.
- These elementary tree fragments are defined using a tree description logic.
- The control language can be compared with a *Definite Clause Grammar*, *i.e.* combination rules using **disjunction** and **conjunction**.
- Two methodological axes of description (Crabbé, 05):
  1. *structure sharing* (*i.e.* reusable elementary tree fragments).
  2. *alternatives* (*i.e.* disjunctions referring to alternative forms of a given grammatical function, etc).

eXtensible MetaGrammar (2 / 2)

- A language to describe tree fragments:

$$\begin{aligned} \mathit{Description} \ ::= \ & x \rightarrow y \mid x \rightarrow^{+} y \mid x \rightarrow^{*} y \mid \\ & x \prec y \mid x \prec^{+} y \mid x \prec^{*} y \mid \\ & x = y \mid x[f{:}E] \mid x(p{:}E) \end{aligned} \tag{1}$$

- A language to combine tree fragments:

$$\mathit{Class} \ ::= \ \mathit{Name} \rightarrow \mathit{Content} \tag{2}$$

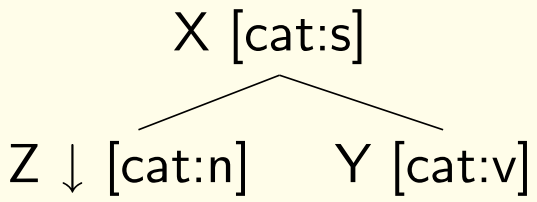
$$\begin{aligned} \mathit{Content} \ ::= \ & \mathit{Description} \mid \mathit{Name} \mid \\ & \mathit{Name} \vee \mathit{Name} \mid \mathit{Name} \wedge \mathit{Name} \end{aligned} \tag{3}$$

Example (1 / 2)

- Tree fragment #1:

$$\begin{aligned} \textit{SubjectCan} \quad \rightarrow \quad & (X\ [\textit{cat} : s] \ \rightarrow Y\ [\textit{cat} : v]\ ) \quad \wedge \\ & (X \rightarrow Z\ (\textit{mark} : \textit{subst})\ [\textit{cat} : n]\ ) \quad \wedge \\ & (Z \prec Y) \end{aligned}$$

$\textit{SubjectCan} \quad \rightarrow$

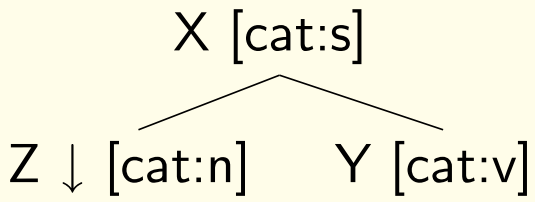


Example (1 / 2)

- Tree fragment #1:

$$\begin{aligned} \mathit{SubjectCan} \quad \rightarrow \quad & (X\ [\mathit{cat} : s] \ \rightarrow Y\ [\mathit{cat} : v]\ ) \quad \wedge \\ & (X \rightarrow Z\ (\mathit{mark} : \mathit{subst})\ [\mathit{cat} : n]\ ) \quad \wedge \\ & (Z \prec Y) \end{aligned}$$

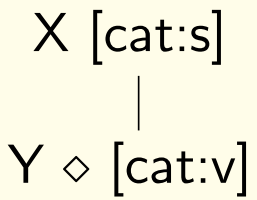
$\mathit{SubjectCan} \quad \rightarrow$



- Tree fragment #2:

$$\begin{aligned} \mathit{Active} \quad \rightarrow \quad & (X\ [\mathit{cat} : s] \quad \wedge \\ & Y\ (\mathit{mark} : \mathit{anchor})\ [\mathit{cat} : v]\ ) \quad \wedge \\ & X \ \rightarrow \ Y\ ) \end{aligned}$$

$\mathit{Active} \quad \rightarrow$

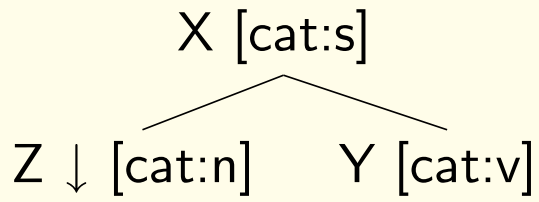


Example (1 / 2)

- Tree fragment #1:

$$\begin{aligned} \textit{SubjectCan} \quad \rightarrow \quad & (X\ [cat : s] \rightarrow Y\ [cat : v]\ ) \quad \wedge \\ & (X \rightarrow Z\ (mark : subst)\ [cat : n]\ ) \quad \wedge \\ & (Z \prec Y) \end{aligned}$$

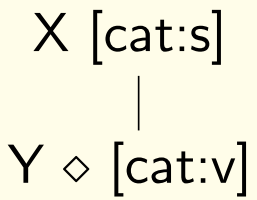
$\textit{SubjectCan} \quad \rightarrow$



- Tree fragment #2:

$$\begin{aligned} \textit{Active} \quad \rightarrow \quad & (X\ [cat : s] \quad \wedge \\ & Y\ (mark : anchor)\ [cat : v]\ ) \quad \wedge \\ & X \rightarrow Y\ ) \end{aligned}$$

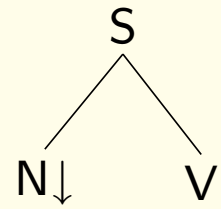
$\textit{Active} \quad \rightarrow$



- Combination rule: $\textit{Intransitive} \rightarrow \textit{SubjectCan} \wedge \textit{Active}\ (*)$

Example (2 / 2)

Some trees for intransitive verbs (*e.g.*, the lexical item *sleeps*)

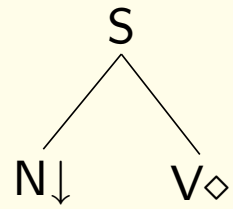


(Canonical Subject)

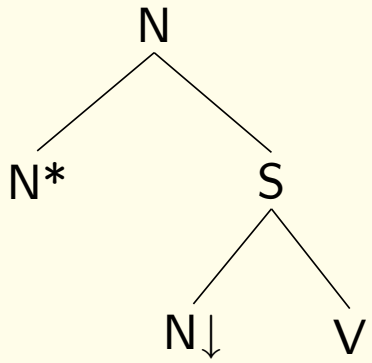
$\wedge$

(Active verb morph)

$\Rightarrow$



(e.g. the boy sleeps)

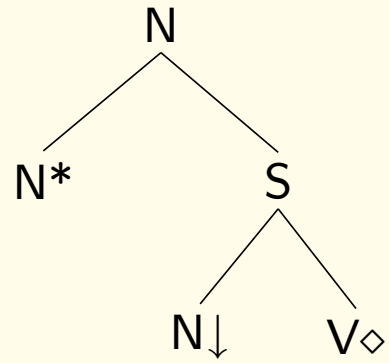


(Extracted Subject)

$\wedge$

(Active verb morph)

$\Rightarrow$



(e.g. the boy who sleeps)

$$\textit{Subject} \rightarrow \textit{SubjectCan} \vee \textit{SubjectExt}$$

$$\textit{Intransitive} \rightarrow \textit{Subject} \wedge \textit{Active}$$

Some features of the XMG formalism

- Flexible management of variables (local scope by default + export declarations) $\Rightarrow$ no name conflicts

$$\begin{array}{lllll} \mathit{Class\ A} & ::= & \langle X, Y \rangle \Leftarrow A & \rightarrow & \{ \ldots \} \\ & \mathit{Class\ B} & ::= \quad B & \rightarrow & \{ Z = A \ \ldots \ Z.X \ \} \end{array}$$

- Possibility to factorise class contents via inheritence.
- Each class of the metagrammar may be equipped with a **interface** (a feature structure used to share information between classes, *e.g.* coindexation of semantic indices)
- The tree description language has been extended to support Interaction Grammars (Perrier, 03).

First extension of the formalism: different levels of description

- Possibility to describe not only tree fragments (*i.e.* syntactic information), but also *flat semantic formulas*.
- Semantic representation based on the *Predicate Logic Unplugged* of (Bos, 95).
- Semantic description language :

$$\begin{aligned} \textit{Description} \ ::= \ & \ell{:}p(E_1, ..., E_n) \mid \ \neg\ell{:}p(E_1, ..., E_n) \mid \\ & E_i \ll E_j \end{aligned} \tag{4}$$

- Each level of description is processed in a specific *dimension*. The control language is then an *Extended Definite Clause Grammar* (Van Roy, 90).

Second extension of the formalism: well-formedness constraints

- Constraints on the structures produced from the metagrammar.
- Interests:
  - to guaranty the validity of the structures (and avoid manual checking).
  - to complete the structures according to linguistic criteria.
- Classification of these constraints into 4 categories:
  1. Formal constraints
  2. Operational constraints
  3. Language-dependent constraints
  4. Theoretical constraints

Formal constraints

- Constraints assuring that the trees generated by the model builder are regular TAG trees.
- On top of being trees, the output structures must respect some specific criteria:
  - each node has a category label,
  - leaf nodes are either marked as *subst*, *foot* or *anchor*,
  - the category of the foot node is identical to that of the root node,
  - etc.

Operational constraint (1 / 3)

- Constraints controlling the combinations of tree fragments (closely linked to the concept of Resources / Needs).
- Constraints based on a colouring of the nodes.
- Each node of the description is labelled either Black, Red or White.
- During minimal model computation, nodes are identified according to the following rules:

$$
\begin{array}{lcl}
\circ_{\mathrm{W}} & + & \circ_{\mathrm{W}} = \circ_{\mathrm{W}} \\
\bullet_{\mathrm{B}} & + & \circ_{\mathrm{W}} = \bullet_{\mathrm{B}} \\
\bullet_{\mathrm{B}} & + & \bullet_{\mathrm{B}} = \bot \\
\bullet_{\mathrm{R}} & + & \{ \circ_{\mathrm{W}} \ ; \ \bullet_{\mathrm{B}} \ ; \ \bullet_{\mathrm{R}} \} = \bot
\end{array}
$$

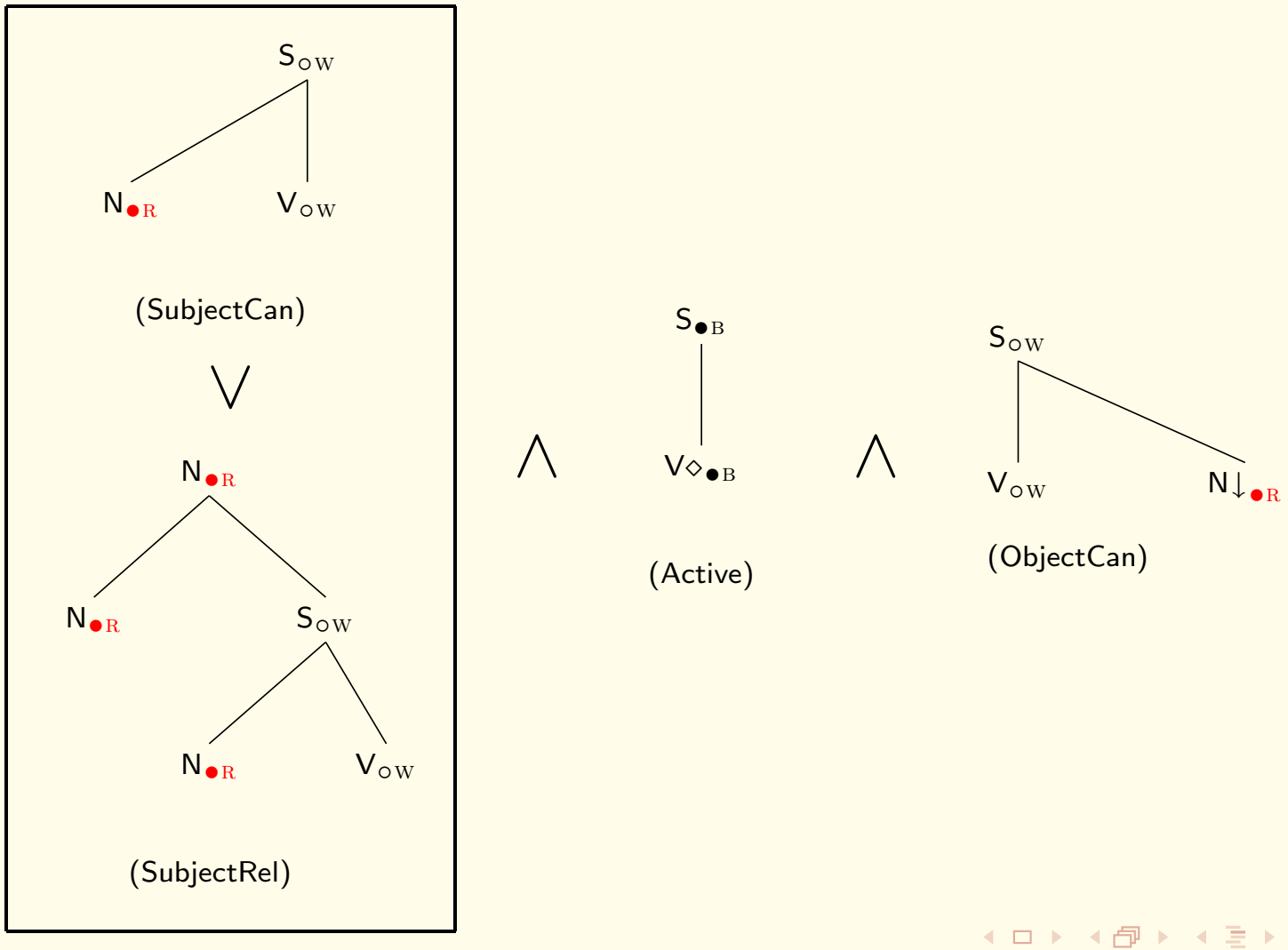
Operational constraint (2 / 3)

Benefits:

- Avoids node naming issues (no global names).
- Allows to reduce the metagrammatical description (node equations are replaced with implicit coloured node identifications).
- Facilitates the reuse of a same tree fragment several times.

Operational constraint (3 / 3)

Example:



Language-dependent constraints (1 / 2)

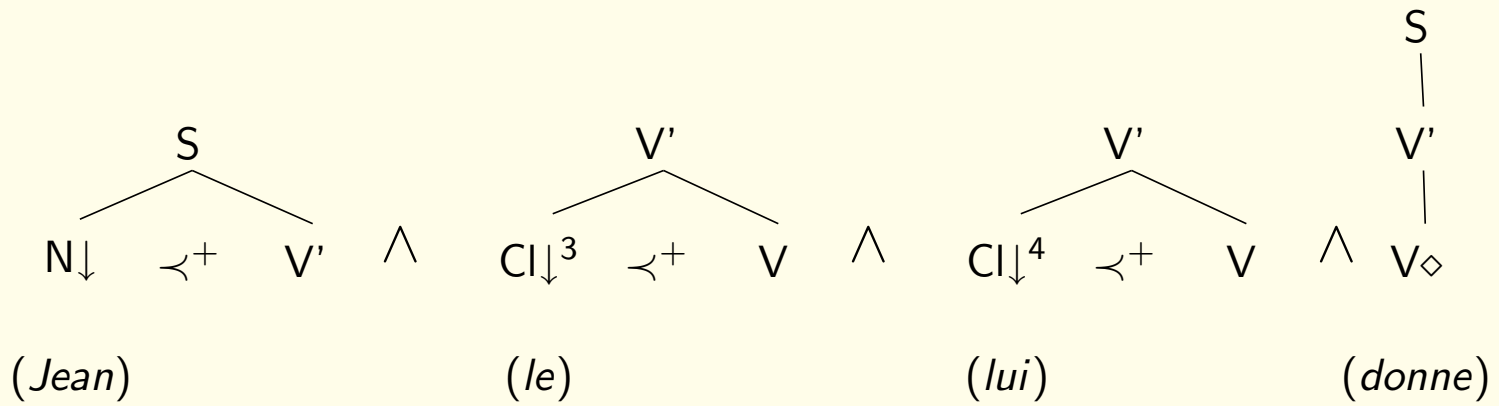
- For French, the ordering and uniqueness of clitics.
- (Perlmutter, 70):

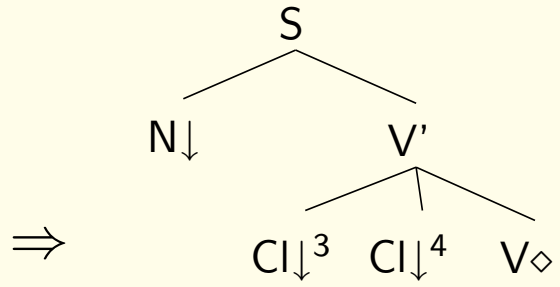
  > *first they appear in front of the verb in a fixed order according to their rank (a-b) and second two different clitics in front of the verb cannot have the same rank (c).*

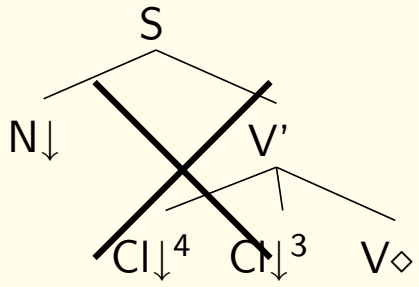
- For instance the clitics *le, la* have the rank 3 and *lui* the rank 4 (rank is a node *property*).

(a) Jean le_3 lui_4 donne
John gives it to him
(b) *Jean lui_4 le_3 donne
*John gives to him it
(c) *Jean le_3 la_3 donne
*John gives it it

Language-dependent constraints (2 / 2)







Theoretical principles

- Language-independent principles related to the grammatical formalism described.
- For TAG, such a principle may be the *Principle of Predicate-Argument Coocurrency*.
- NB: such principles are not yet implemented within the XMG system.

XMG – the implementation

- A 3-step metagrammar compilation:
  1. translation of the metagrammar into intermediate code for a specific virtual machine.
  2. execution of this code and accumulation of partial tree descriptions.
  3. solving of the tree descriptions accumulated in step 2.

The metagrammar is written using an object-oriented concrete syntax:

```
class C1
declare ?X
export X
{
  <syn>
  {node X[cat=s]}
}
```

```
class C2
declare ?Y
export Y
{
  <syn>
  {node Y[cat=v]}
}
```

```
class C
import C1 C2
{
  <syn>
  {X->Y}
}
```

Compilation into specific assembler code:

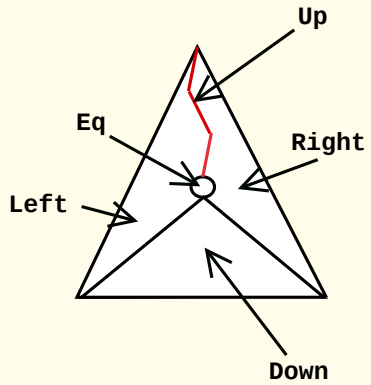
```
call(C1)        % memory allocation + execution of code
node(X)         % unification of X with a node structure
feat(X [(cat s)])
call(C2)
node(Y)
feat(X [(cat v)])
dom(C1.X C2.Y) % accumulation of a dominance relation
```

A Virtual Machine with a logic kernel

- Architecture based on the Warren's Abstract Machine (Stack, Heap, Trail and Choice Point).
- Performs unification with chronological backtracking and multiple accumulation (to distinguish between dimensions).
- NB: Unification may concern specific data structures (*e.g.* polarised features for Interaction Grammars).

A Constraint Satisfaction Tree Description Solver (1 / 3)

1. Setting the constraint framework:
   - Each node in the input description is associated with an integer.
   - Then, we use an asbtract data type to refer to a node of a valid model in terms of the nodes being equals, above, below, or on its side:



$$N_x^i ::= node(\ Eq\colon \{\langle ints\rangle\}\ \ Up\colon \{\langle ints\rangle\}\ \ Down\colon \{\langle ints\rangle\}\ \ Left\colon \{\langle ints\rangle\}\ \ Right\colon \{\langle ints\rangle\})$$

A Constraint Satisfaction Tree Description Solver (2 / 3)

- The input description is converted into relation constraints on node sets.

  For instance, the dominance relation $x \longrightarrow y$ can be translated as:

$$N_x^i \longrightarrow N_y^j \equiv \begin{array}{l}[N_{x.EqUp}^i \subseteq N_{y.Up}^j \wedge N_{x.Down}^i \supseteq N_{y.EqDown}^j \\ \wedge\, N_{x.Left}^i \subseteq N_{y.Left}^j \wedge N_{x.Right}^i \subseteq N_{y.Right}^j]\end{array}$$

A Constraint Satisfaction Tree Description Solver (2 / 3)

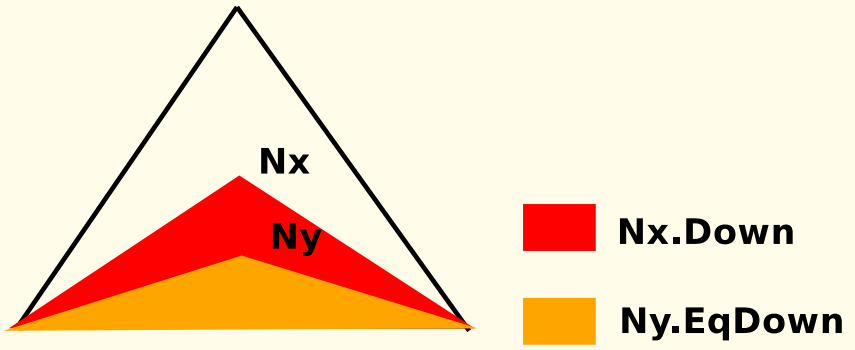
- The input description is converted into relation constraints on node sets.

  For instance, the dominance relation $x \longrightarrow y$ can be translated as:

$$N_x^i \longrightarrow N_y^j \equiv \begin{array}{l}[N_{x.EqUp}^i \subseteq N_{y.Up}^j \wedge N_{x.Down}^i \supseteq N_{y.EqDown}^j \\ \wedge N_{x.Left}^i \subseteq N_{y.Left}^j \wedge N_{x.Right}^i \subseteq N_{y.Right}^j]\end{array}$$

-

$$N_{x.Down}^i \supseteq N_{y.EqDown}^j$$



A Constraint Satisfaction Tree Description Solver (3 / 3)

2. Searching the solutions to the problem:
   - The solutions are the assignments for each of the node sets associated with the nodes of the input description.
   - A distribution strategy is used to explore the consistent assignments for these node sets.
   - The implementation of the solver follows the ideas of (Duchier and Niehren, 2000) and uses the constraint programming support of the Oz/Mozart environment.

Extension to specific constraints (1 / 2)

- This constraint framework can relatively easily be extended to solve specific constraints, such as those introduced previously.
- The idea:
  1. extension of the node representation (tuples whose fields contain sets of nodes),
  2. definition of additional constraints on these fields, reflecting the syntactic constraints we want to express.

Extension to specific constraints (2 / 2)

example: Clitic uniqueness.

- *In a valid model ϕ, there is only one node having a given property p.*

- For each node x *in the description*, we add to its representation a field containing a boolean variable p_x indicating whether the node denoting x *in the model* has this property or not:

$$p_x \equiv (N^i_{x.Eq} \cap \mathcal{V}^\phi_p) \neq \emptyset$$

- Finally, if true value ~ 1 and false ~ 0:

$$\sum_{x \in \phi} p_x \quad \leq \quad 1$$

Some figures

- The XMG system has been used successfully to compute a core TAG for French: 6,000+ trees computed from a description containing 293 classes (Crabbé, 05).
- The compilation of this TAG takes about 15 min with a P4 processor 2.6 GHz and 1 GB RAM.
- This core TAG has been evaluated on the TSNLP and has a coverage rate of 75 % (recognition of grammatical sentences).
- This core TAG has been extended with flat semantics (Gardent, 06) so that it can be used both for *parsing* and *generation* (Kow et al, 06).

Outline

Part 2. Semantic Construction

Syntax / Semantics interface in TAG (1 / 2)

- Syntax / Semantics interface based on (Gardent and Kallmeyer, 03)
- Semantic representation language: flat semantic formulas (Bos, 95) using *unification variables*

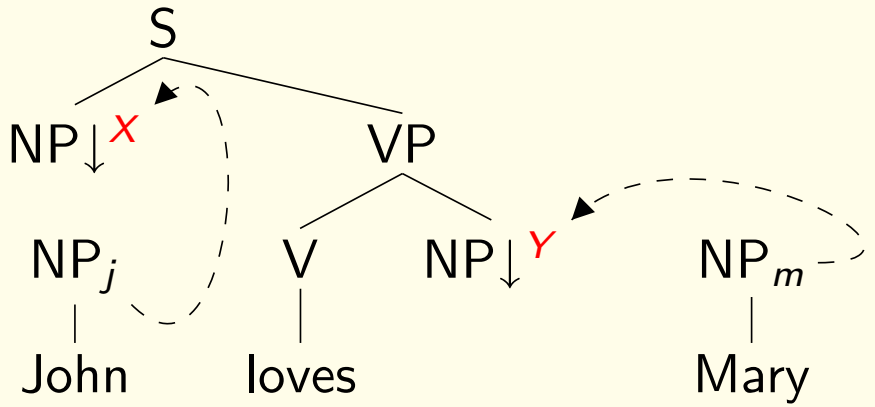
John	$l_0 : john(j)$
Mary	$l_1 : mary(m)$
loves	$l_2 : loves(X, Y)$

- semantic operation: **union modulo unification**

John loves Mary $\quad l_2 : love(j, m), l_0 : john(j), l_1 : mary(m)$

Syntax / Semantics interface in TAG (2 / 2)

- principle:
  - (i) each elementary **tree** is associated with a **semantic formula**
  - (ii) the FS decorating the nodes of the trees **share the unification variables** occuring in the semantic formulas
  - (iii) the **association** between tree nodes and semantic variables specify the **syntax / semantics interface**



l_0:*john*(*j*) l_2:*love*(*X*, *Y*) l_1:*mary*(*m*)

$l_2 : love(j, m)$,
$l_0 : john(j)$,
$l_1 : mary(m)$

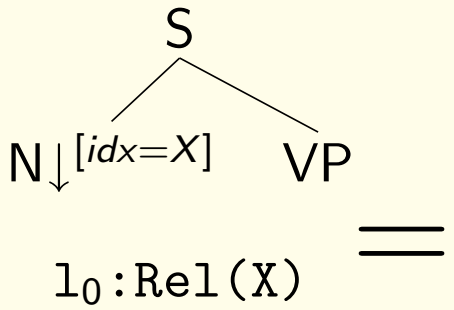
Integration of the semantic interface within the metagrammar (1 / 2)

- Target: share semantic indices with the corresponding node features within tree fragments.
- How: use the class interface to extend the scope of a given node feature (and give it a global name), the same for semantic arguments.
- During class combinations, the interfaces are unified, resulting in the unification of node features and corresponding semantic arguments.

Integration of the semantic interface within the metagrammar (2 / 2)

- Example:

Intransitive:

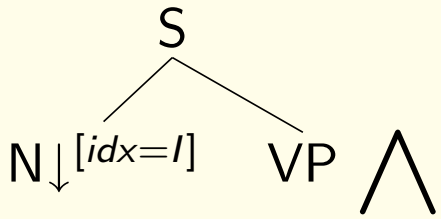


arg1=X
subjIdx=X

(Result)

=

Subject:



∧

subjIdx=I

(Grammatical functions)

Active:

∧

(Verb morph)

UnaryRel:

l_0:Rel(I1)

arg1=I1

(Semantics)

Semantic Construction

- Based on the result of the syntactic parsing, in our case from a **derivation forest** (De La Clergerie, 05)
- Advantage: using the **factorisation** of the parses for computing all the semantic representations
- A 3 steps process:

  (1) **extraction** of the semantic information included in the grammar $\longrightarrow$ **semantic lexicon**

  (2) pure **syntactic** parsing on the basis of the grammar without any semantic feature $\longrightarrow$ **derivation forest**

  (3) **rebuilding of the semantic representation** from (a) the semantic lexicon and (b) the derivation operations

1. Extraction of the semantic lexicon (1 / 2)

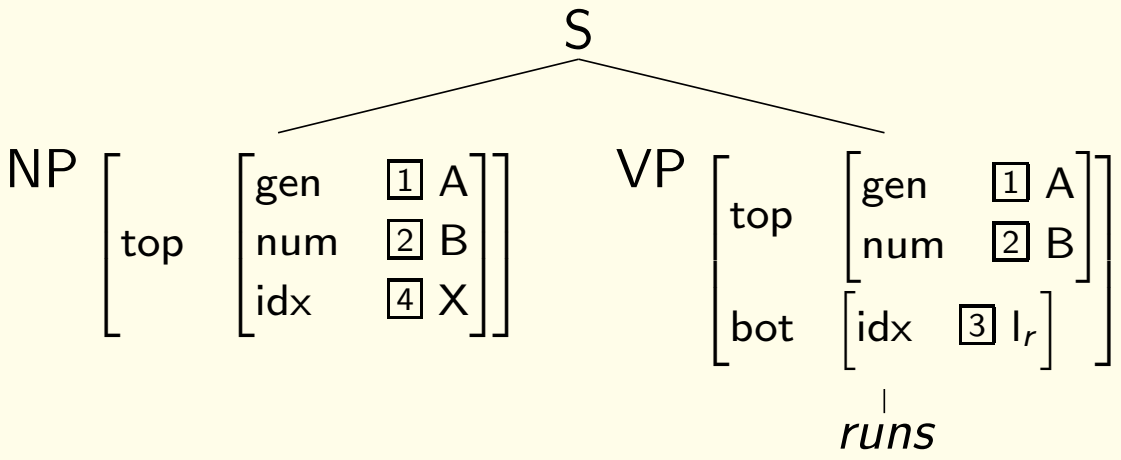
- to be able to **rebuild the derivations** done during parsing, we have to first store the **semantic information** contained in the FB-TAG trees

- Automatic extraction process:

  1. for each tree in the grammar G, **numbering** of the nodes with their gorn address $\longrightarrow$ resulting grammar GC

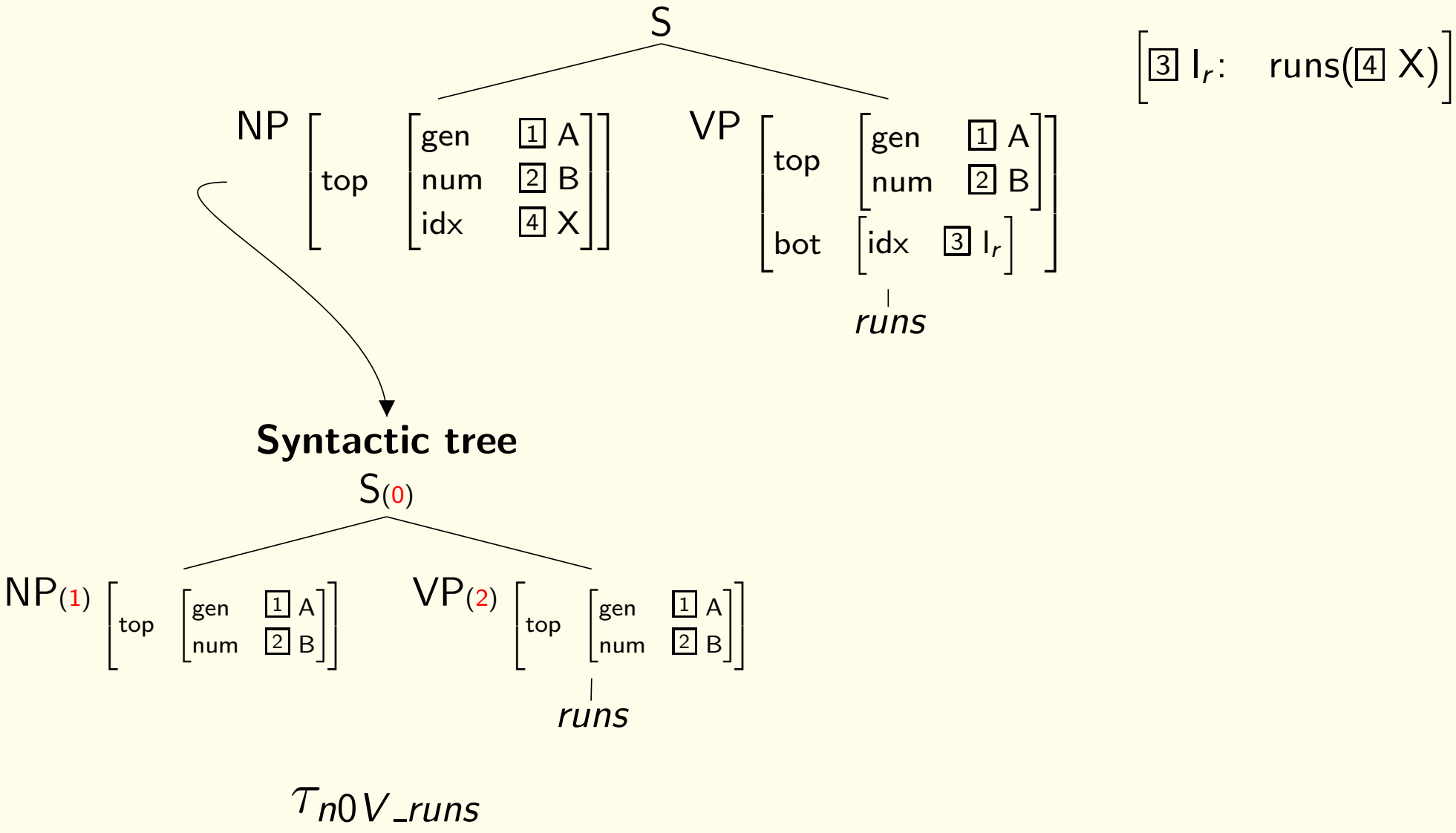
  2. for each tree in the grammar GC:
     (a) create a **purely syntactic** FB-TAG tree
     (b) create an **entry in the semantic lexicon** composed of the tree name, the lemma, the semantic representation, and the semantic features associated with tree nodes (link kept via the gorn address)

1. Extraction of the semantic lexicon (2 / 2)

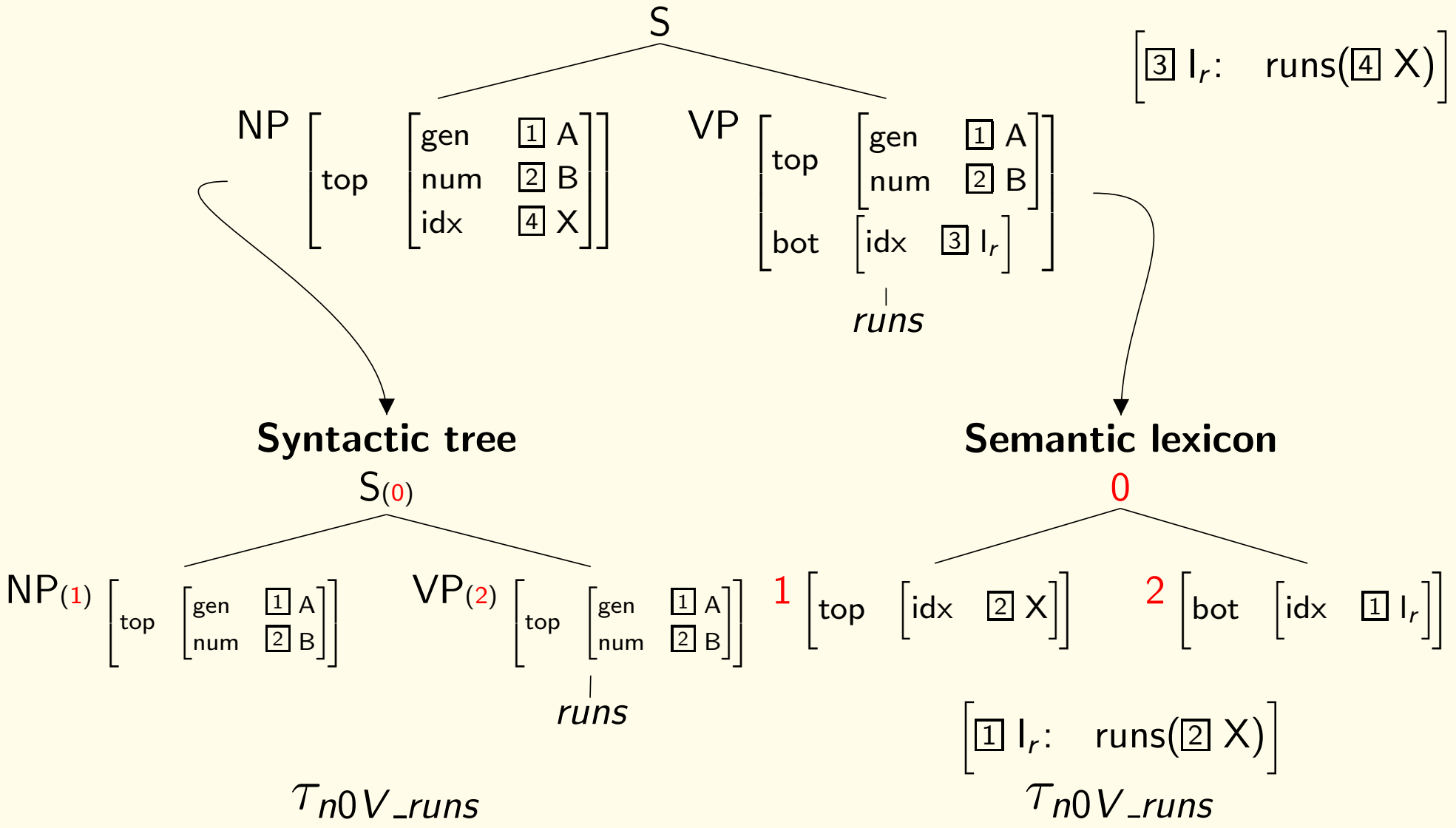


$$\left[\boxed{3}\ l_r\text{:}\quad \text{runs}(\boxed{4}\ X)\right]$$

1. Extraction of the semantic lexicon (2 / 2)



1. Extraction of the semantic lexicon (2 / 2)



2. Pure syntactic parsing

Parsing using the automatically extracted syntactic grammar:

- after extraction of the semantic lexicon, the resulting grammar consists of a *purely syntactic* FB-TAG
- we thus use it to parse sentences with a FB-TAG parser produced automatically by the DyALog system of (De La Clergerie, 05)
- as a result, we obtain a **derivation forest** (*i.e.* a shared representation of all possible derivation trees of the given sentence)

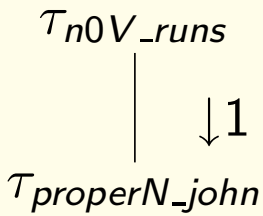
3. Computing semantic representations

- **top-down traversal** of the shared derivation forest
- for each node of the derivation forest (corresponding to an elementary tree), we retrieve its **entry from the semantic lexicon**
- for each derivation edge, we recompute the **unification operations** done on the derived tree: unification on node FS, *i.e.* on semantic features recorded in the semantic lexicon
- we collect the semantic representations of each entry of the semantic lexicon
  N.B. these representations contain unified variables

An example (1 / 2)

- Sentence to parse: *John runs*

Derivation forest



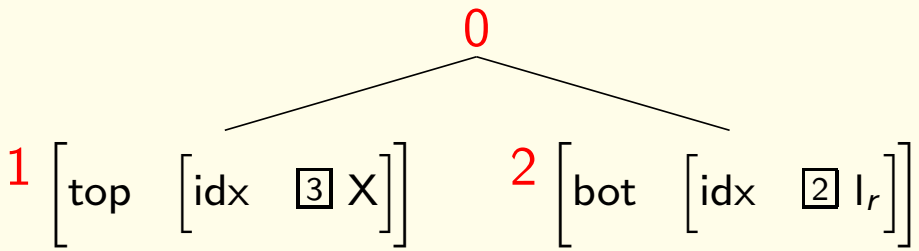
- Content of the semantic lexicon:

$\tau_{properN\_John}$

$$0\left[\begin{array}{ll}\text{bot} & \left[\text{idx} \quad \boxed{1}\ \text{j}\right]\end{array}\right]$$

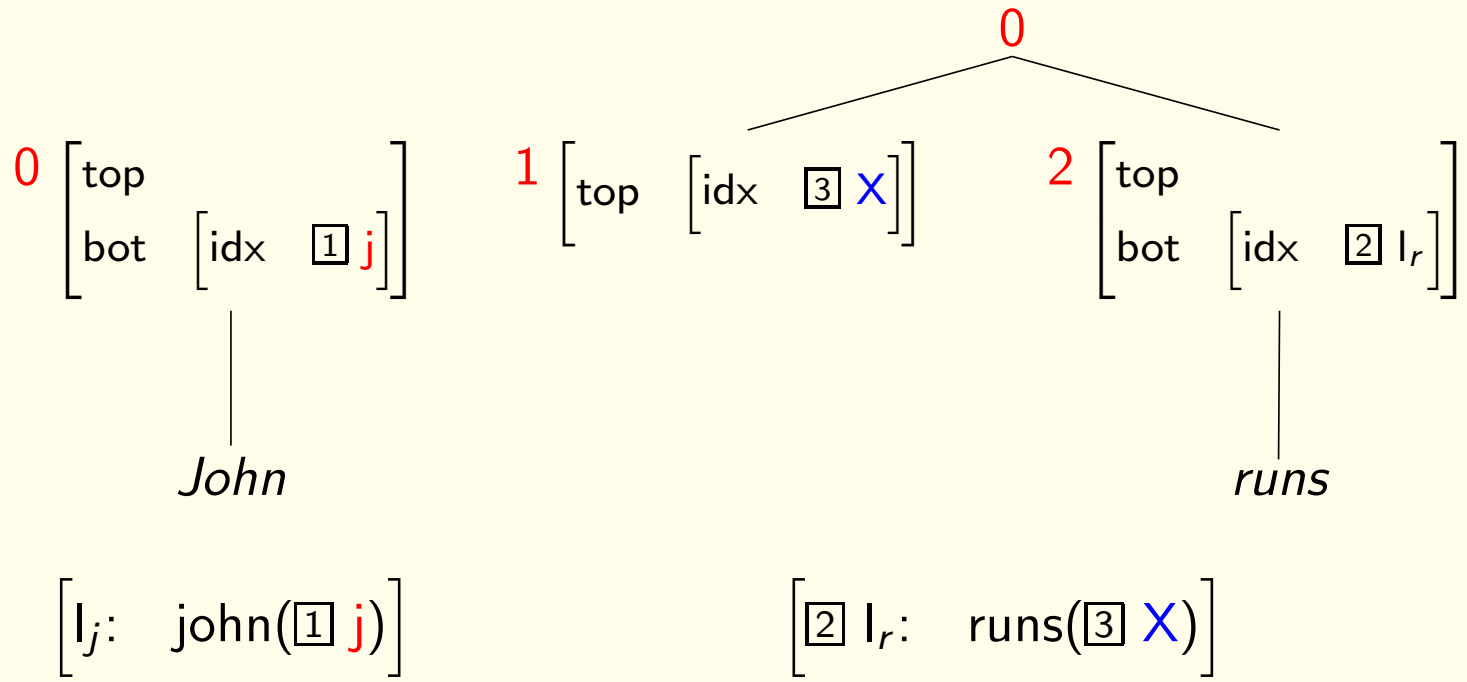
$$\left[\text{l}_j\text{:} \quad \text{john}(\boxed{1}\ \text{j})\right]$$

τ_{n0V_runs}

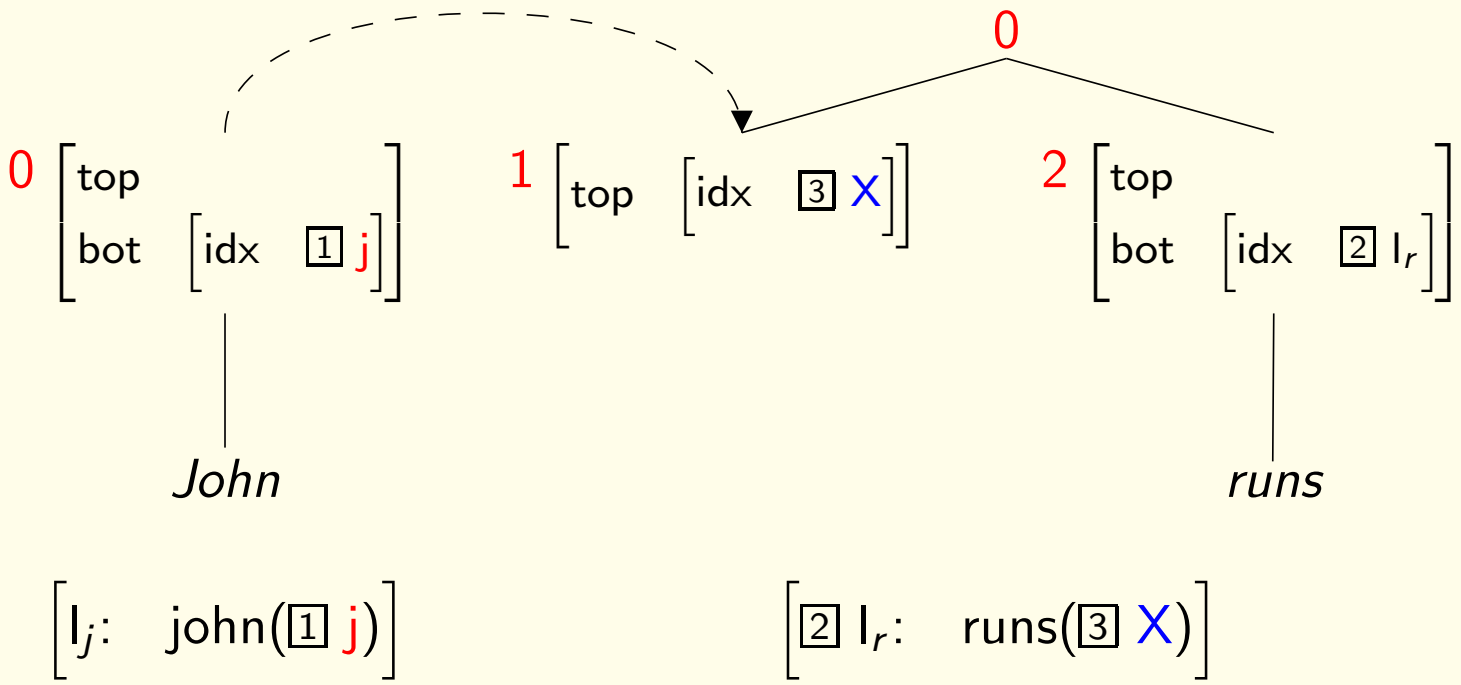


$$\left[\boxed{2}\ \text{l}_r\text{:} \quad \text{runs}(\boxed{3}\ \text{X})\right]$$

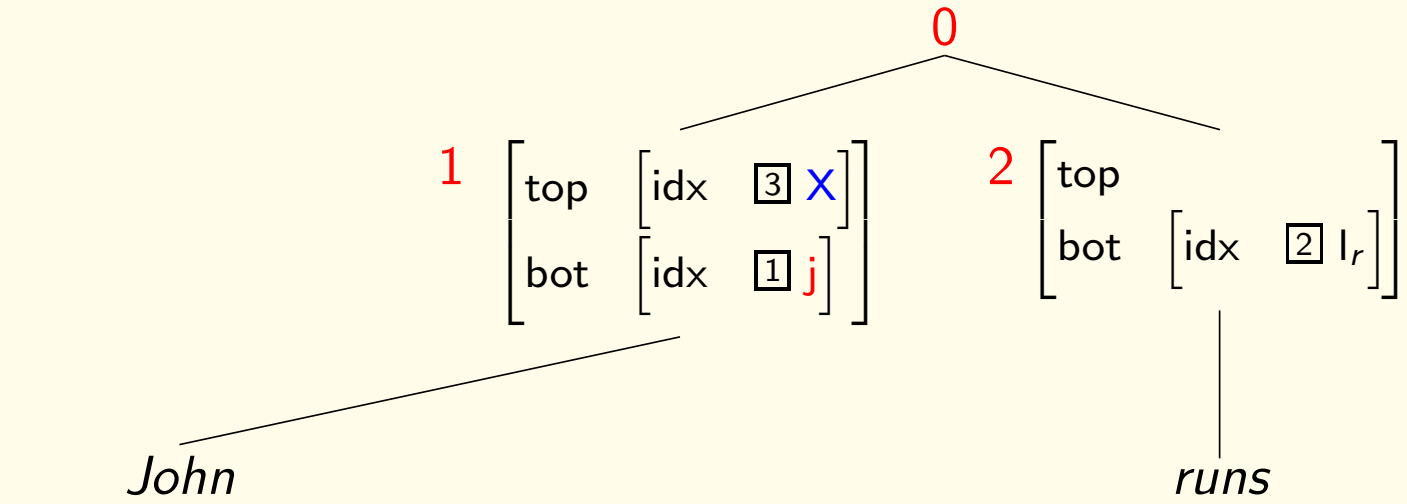
An example (2 / 2)



An example (2 / 2)

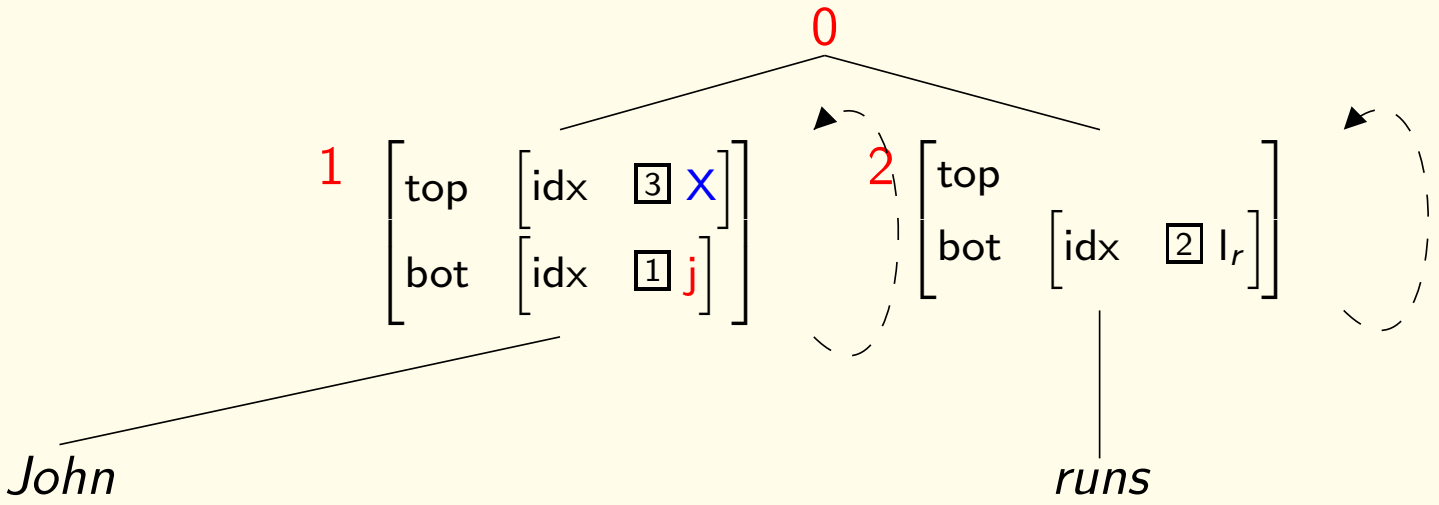


An example (2 / 2)



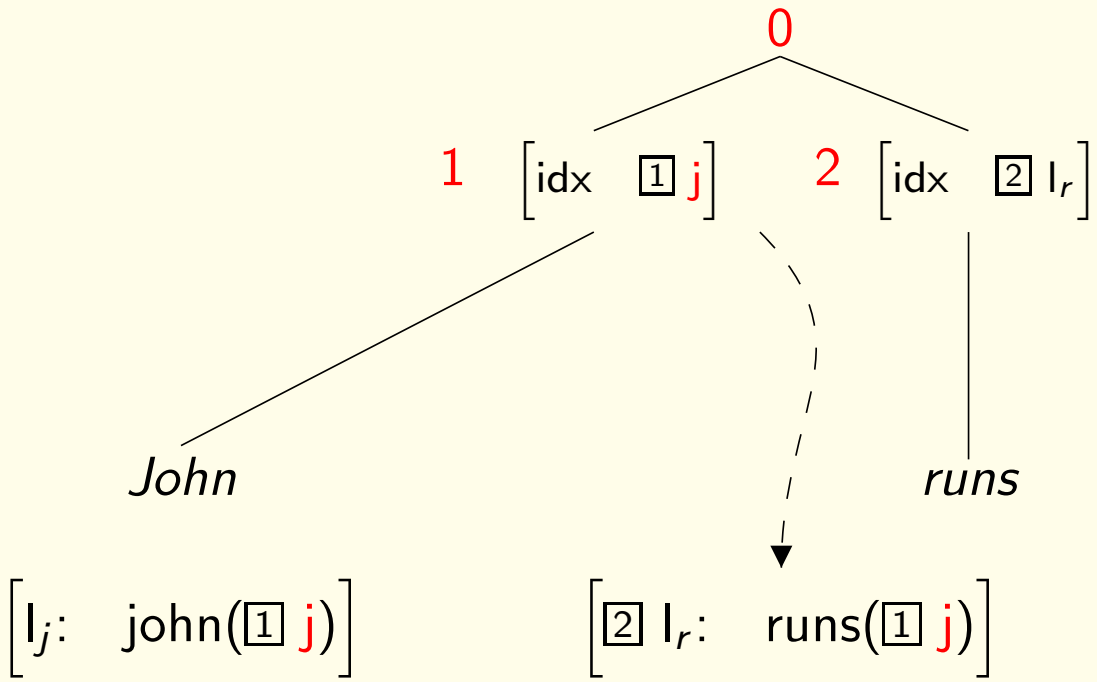
$$\left[l_j : \quad \text{john}(\boxed{1}\, j) \right] \qquad \left[\boxed{2}\, l_r : \quad \text{runs}(\boxed{3}\, X) \right]$$

An example (2 / 2)



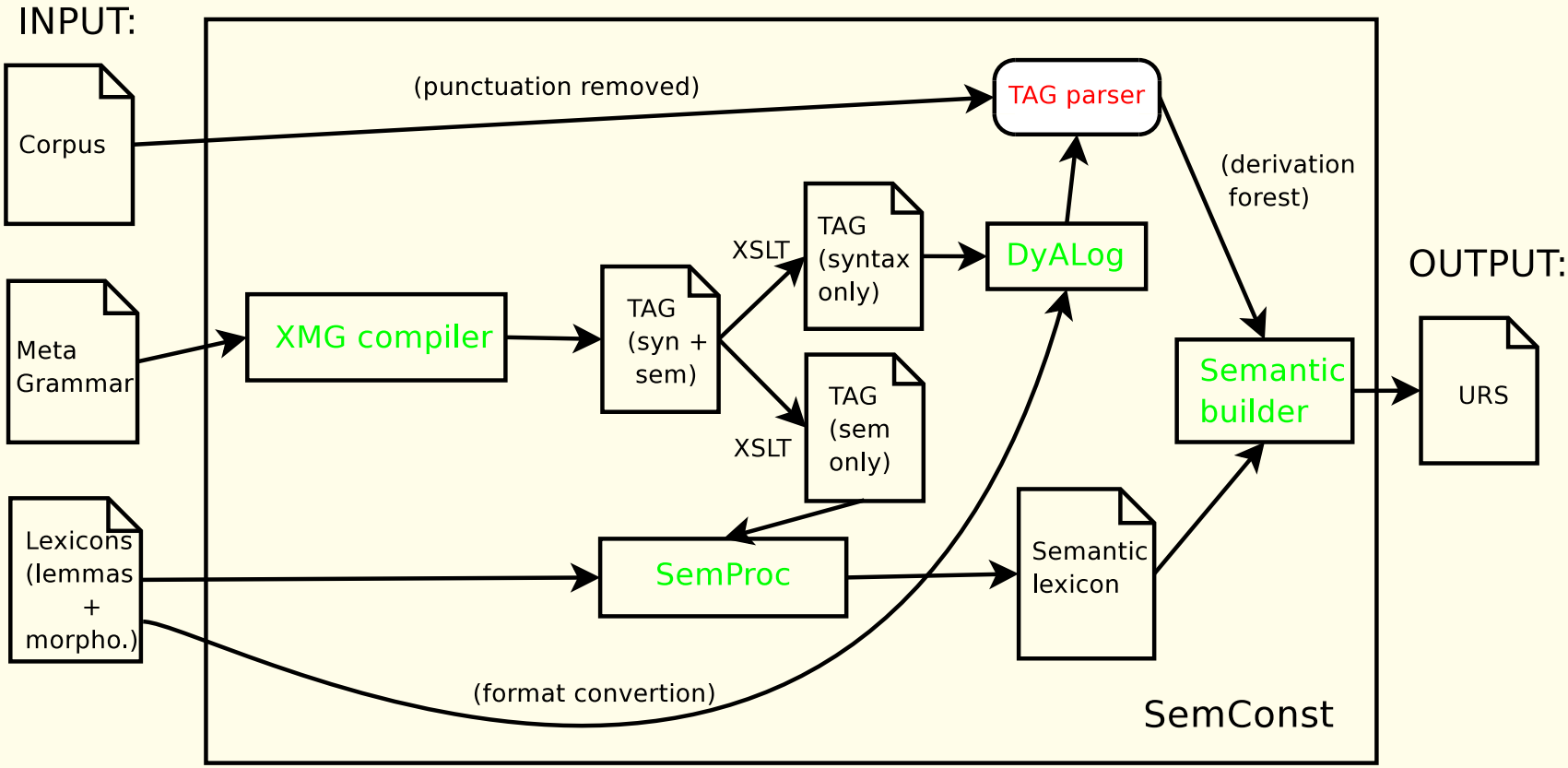
$$\left[l_j : \quad \text{john}(\boxed{1}\, j) \right] \qquad \left[\boxed{2}\, l_r : \quad \text{runs}(\boxed{3}\, X) \right]$$

An example (2 / 2)



SemTAG from the inside

URS: underspecified Semantic Representations



SemTAG ::= SemConst + GenI + XMG

Conclusion (1 / 2)

- The XMG formalism includes a fully declarative language to handle tree fragments.
- On top of describing tree-based grammar, XMG allows (i) to describe other linguistic levels of description (*i.e.* semantics) and (ii) to express constraints on the computed structures.
- The XMG system has been used to produce a core TAG for French (one for syntax and one equipped with semantics).
- Both of these grammars are (being) evaluated using the TSNLP.

Conclusion (2 / 2)

- The Semantic Construction program allows high information sharing by using a metagrammar and a tabular parser.
- The Syntax / Semantic interface chosen presents some interesting features : homogeneous interface using feature structures, unification as a semantic composition operation.
- Future work concerns :
  - the extension of the control language to facilitate metagrammar development: node typing ?
  - the extension of the metagrammatical framework to describe different target formalisms (from a given metagrammar), development of a library of constraints ?
  - the extension of the semantic interface and the semantic construction process to deal with different semantic representations, TAG semantic with λ-terms ?

Acknowledgement

- *eXtensible MetaGrammar* – Benoît Crabbé, Denys Duchier and Joseph Le Roux.
- *Semantic Constructor* – Claire Gardent, Eric Kow.
- For further information about the SemTAG platform :

`http://trac.loria.fr/~semtag`
`http://trac.loria.fr/~semconst`
`http://trac.loria.fr/~geni`
`http://sourcesup.cru.fr/xmg`