Zen and The Art of Grammar Maintenance

Denys Duchier

LIFO, Université d'Orléans

6 June 2008 / TAG+9 Tutorial

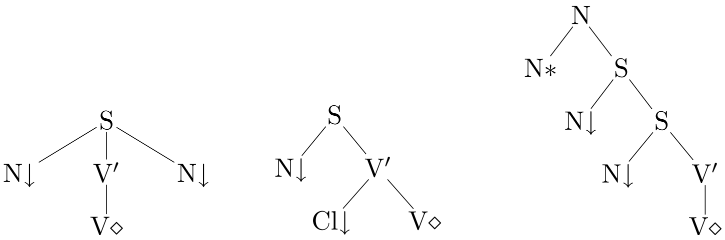
Outline

1 Motivations

2 Metagrammars as Logic Programs

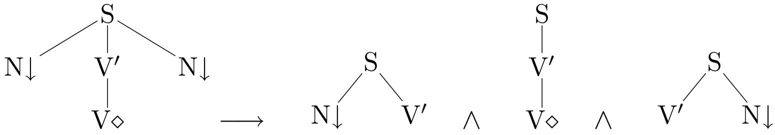
3 XMG Formalism

Factorization and Reuse



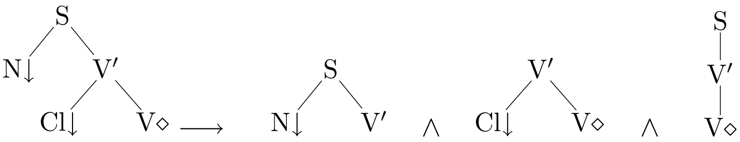
Jean mange une pomme   Jean la mange   la pomme que Jean mange

Jean mange une pomme



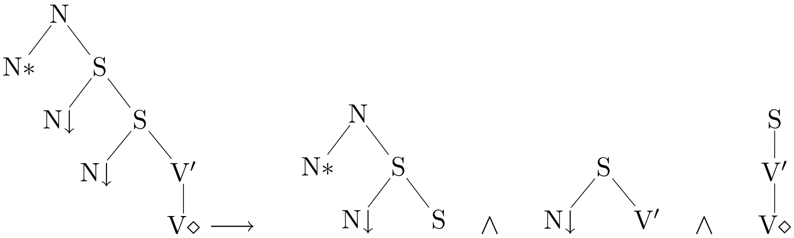
- Canonical Subject
- Active Verb Form
- Canonical Object

Jean la mange



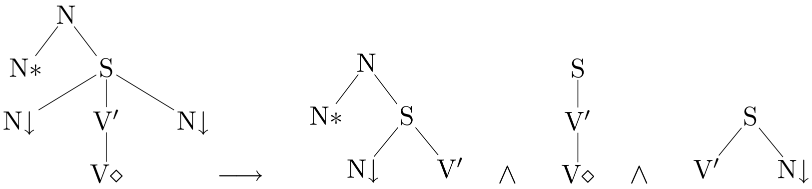
- Canonical Subject
- Clitic Object
- Active Verb Form

La pomme que Jean mange



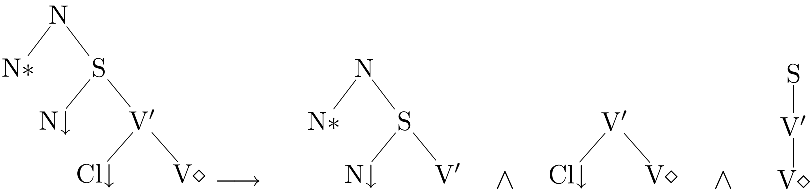
- Relative Object
- Canonical Subject
- Active Verb Form

Jean qui mange la pomme



- Relative Subject
- Active Verb Form
- Canonical Object

Jean qui la mange



- Relative Subject
- Clitic Object
- Active Verb Form

Outline

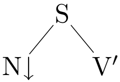
1 Motivations

2 Metagrammars as Logic Programs

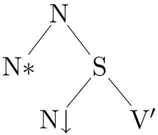
3 XMG Formalism

Subject

SubjectCanonical $\longrightarrow$



SubjectRelative $\longrightarrow$

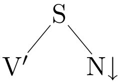


Subject $\longrightarrow$ SubjectCanonical

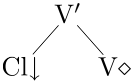
$\vee$ SubjectRelative

Object

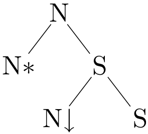
ObjectCanonical $\longrightarrow$



ObjectClitic $\longrightarrow$



ObjectRelative $\longrightarrow$



Object $\longrightarrow$ ObjectCanonical $\vee$ ObjectClitic $\vee$ ObjectRelative

Transitive Active Verb

VerbActiveMorph $\longrightarrow$ 

VerbActiveTransitive $\longrightarrow$ Subject $\wedge$ VerbActiveMorph $\wedge$ Object

Metagrammars as DCGs

- a metagrammar is a DCG
- terminals are tree descriptions

a metagrammar is the grammar of a grammar

Metagrammars as DCGs

- a metagrammar is a DCG
- terminals are tree descriptions

a metagrammar is the grammar of a grammar

Model-Theoretic View

$$\text{Set of tree descriptions} \models \text{Metagrammar}$$
$$\text{Lexical entry} \models \text{Set of tree descriptions}$$

Operational View

- a metagrammar is a logic program
- its execution accumulates tree descriptions
- which are then processed by a solver
- resulting in the production of a lexical entry
- backtrack to obtain the rest of the lexicon

Outline

1 Motivations

2 Metagrammars as Logic Programs

3 XMG Formalism

Type definitions

```
type CAT={n,v,p}
type PERS=[1..3]
type FLEX=[num:NUMBER, gen:GENDER, pers:PERS]
```

Property definitions

```
property extraction : bool
```

Property definitions

```
property extraction : bool {extra = +}
```

$$\texttt{extra} \quad \equiv \quad \texttt{extraction} = +$$

Feature definitions

```
feature num : NUMBER
```

Class definitions

```
class C1
import C2[]
export ?X ?Y
declare ?X ?Y !Z
{
    Statement
}
```

Restricted imports

```
class C1
import C2[] as [?X1,...,?Xn]
export ?X ?Y
declare ?X ?Y !Z
{
    Statement
}
```

Renamings

```
class C1
import C2[] as [?X1=?Y1,...,?Xn]
export ?X=?U ?Y
declare ?X ?Y !Z
{
    Statement
}
```

Statements

$$
\begin{array}{rcl}
S & ::= & S_1 \;;\; S_2 \\
 & | & S_1 \mid S_2 \\
 & | & E_1 = E_2 \\
 & | & \mathit{Class}[E_1, \ldots, E_n] \\
 & | & \langle \mathit{Dim} \rangle \; \{ \; \ldots \; \} \\
 & | & S \mathrel{*=} [f = E, \ldots]
\end{array}
$$

Expressions

$$
\begin{array}{rcl}
E & ::= & ?X \mid !X \\
& \mid & Atom \mid Int \mid String \\
& \mid & @\{Atom, \ldots, Atom\} \\
& \mid & ?X = [f_1 = E_1, \ldots, f_n = E_n] \\
& \mid & E_1 . E_2 \\
& \mid & E(E_1, \ldots, E_n) \\
& \mid & E_1 | E_2 \\
& \mid & (E)
\end{array}
$$

Dimension-specific description languages

$$\langle \mathit{Dim} \rangle \; \{ \ldots \}$$

- tree descriptions
- hole semantics
- AVM, (semi-)lattices

There are currently 3 built-in dimensions: `<syn>` `<sem>` `<dyn>`

Tree Description Language

<syn> { *Syn* }

$$
\begin{array}{lcl}
\mathit{Syn} & ::= & \mathit{Syn}; \mathit{Syn} \\
 & \mid & \mathit{Syn} | \mathit{Syn} \\
 & \mid & \textbf{node} \quad ?X \quad (p = E, \ldots) \quad [f = E, \ldots] \\
 & \mid & E \mathrel{-}> E \mid E \mathrel{-}>^{+} E \mid E \mathrel{-}>^{*} E \\
 & \mid & E >> E \mid E >>^{+} E \mid E >>^{*} E \\
 & \mid & E = E \\
 & \mid & E < E_1, \ldots, E_n >
\end{array}
$$

Alternative Syntax

- dominance

```
node {        node }
node { ...+ node }
node { ...    node }
```

- precedence

```
node            node
node      ,,,+ node
node      ,,,   node
```

Flat Semantics Description Language

<sem> { *Sem* }

$$
\begin{array}{lcl}
\mathit{Sem} & ::= & \mathit{Sem}; \mathit{Sem} \\
& | & \mathit{Sem} | \mathit{Sem} \\
& | & \mathit{Var} : E(E, \ldots, E) \\
& | & \mathit{Var} : \tilde{\ } E(E, \ldots, E) \\
& | & \mathit{Var} << \mathit{Var}
\end{array}
$$

Mutual Exclusion Sets

```
mutex SUBJ-INV
mutex SUBJ-INV += CanonicalObject
mutex SUBJ-INV += InvertedNominalSubject
```

2 classes in the same mutex cannot both be used in the same derivation

Principles