Grammar Implementation: XMG

XMG Tutorial

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

HHU Düsseldorf

WS 2017/2018

Defining abstractions

Classes allow to:

- Control the scope of variables
- Make (parametrized) abstractions

Examples (just headers):

```
class kicked_the_bucket
import nx0Vnx1[]
declare ?X0 ?X1
```

```
class nx0Vnx1
export ?S ?NP_Subj ?VP ?V ?NP_Obj
declare ?S ?NP_Subj ?VP ?V ?NP_Obj ?X0 ?X1
```

Defining abstractions

```
class Intransitive
declare ?S ?NP ?VP ?V
{
  <syn>{
      node ?S [cat=s];
      node ?VP [cat=vp];
      node ?V (mark=anchor) [cat=v];
      node ?NP (mark=subst) [cat=n];
      ?S -> ?VP; ?VP -> ?V;
      ?S -> ?NP; ?NP >> ?VP
  }
}
```

Using abstractions

Classes can be used by other classes by two means:

- Importing the class in the header: all the (exported) variables are added to the scope, all the constraints from the class are added to the current set of constraints
- Calling the class in the body: variables are not added to the scope

Calling classes has two advantages:

- alternatives are possible (disjunction)
- it allows to use parameters

Examples:

```
CanObj[] | RelObj[]
```

```
?C=Class[?X]
```

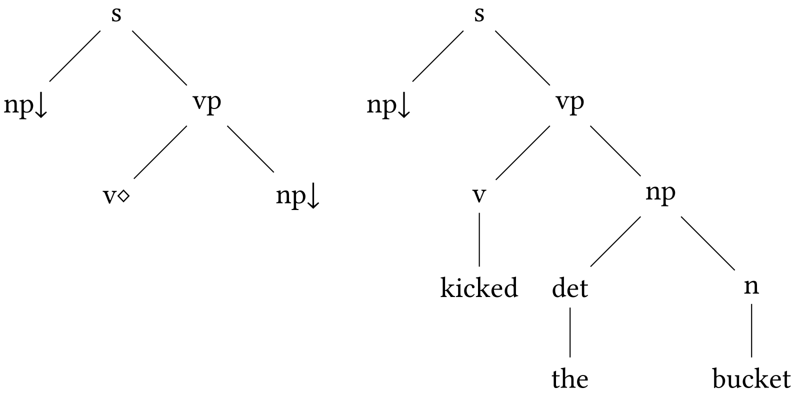
Classes: examples (1)

```
class a
export ?A
declare ?A ?S
{
  <syn>{
    ?S -> ?A
  }
}

class b
import a[]
declare ?B
{
  <syn>{
    ?B -> ?A
  }
}
```

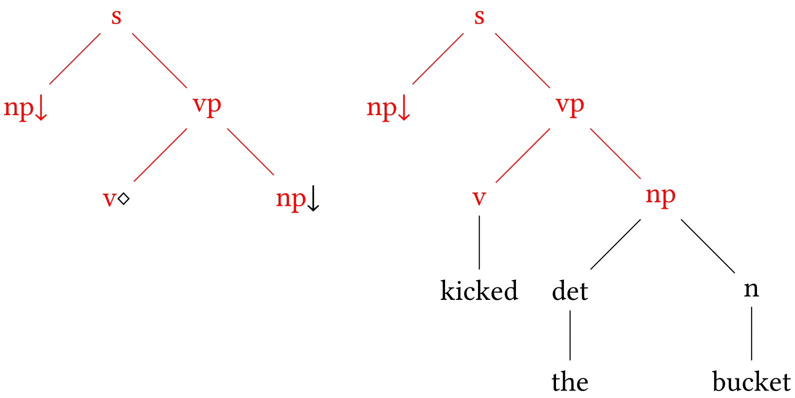
Classes: examples (2)

```
class a
export ?S
declare ?A ?S
{
  <syn>{
    ?S -> ?A
  }
}

class b
import a[]
declare ?A
{
  <syn>{
    ?S -> ?A
  }
}
```

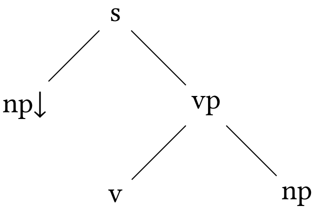
Redundancy



Redundancy

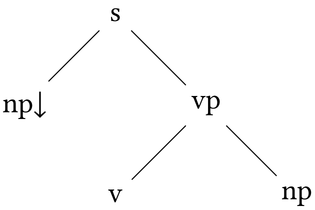


First step: a generic tree fragment for transitive verbs



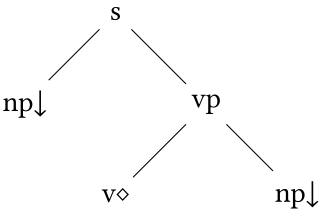
```
class nx0Vnx1
export ?S ?NP_Subj ?VP ?V ?NP_Obj
declare ?S ?NP_Subj ?VP ?V ?NP_Obj
{
  <syn>{
     node ?S [cat=s] {
       node ?NP_Subj (mark=subst) [cat=np]
       node ?VP [cat=vp] {
         node ?V [cat=v]
         node ?NP_Obj [cat=np] }}
  }
}
```

First step: a generic tree fragment for transitive verbs



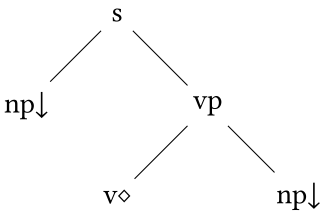
```
class nx0Vnx1
export ?S ?NP_Subj ?VP ?V ?NP_Obj
declare ?S ?NP_Subj ?VP ?V ?NP_Obj
{
  <syn>{
     node ?S [cat=s] {
       node ?NP_Subj (mark=subst) [cat=np]
       node ?VP [cat=vp] {
         node ?V [cat=v]
         node ?NP_Obj [cat=np] }}
  }
}
```

Standard tree for transitive verbs



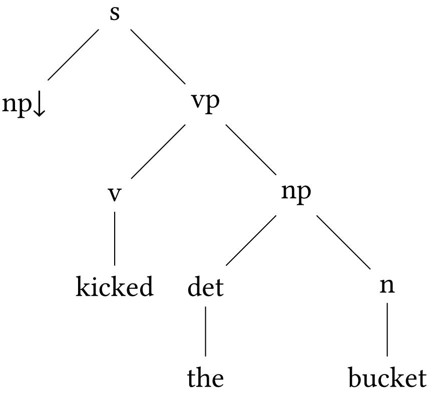
```
class kick
import nx0Vnx1[]
{
  <syn>{
    node ?V (mark=anchor);
    node ?NP_Obj (mark=subst)
  }
}
```

Standard tree for transitive verbs



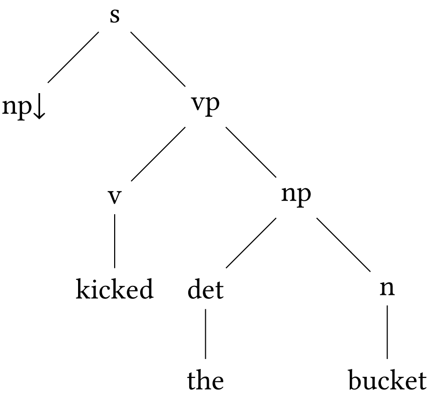
```
class kick
import nx0Vnx1[]
{
  <syn>{
    node ?V (mark=anchor);
    node ?NP_Obj (mark=subst)
  }
}
```

Tree for "kicked the bucket"



```
class kicked_the_bucket
import nx0Vnx1[]
{
  <syn>{
     ?V [e=?X0] "kicked";
     ?NP_Obj [] {
       [cat=det] "the"
       [cat=n]   "bucket"
     }
  }
}
```

Tree for "kicked the bucket"



```
class kicked_the_bucket
import nx0Vnx1[]
{
  <syn>{
     ?V [e=?X0] "kicked";
     ?NP_Obj [] {
       [cat=det] "the"
       [cat=n]   "bucket"
     }
  }
}
```

Principles: motivation

- As fragments become more numerous, controlling their combination (and the scope of variables) gets difficult
- Idea: adding new constraints on top of dominance and precedence
- Principles: sets of additionnal constraints for the solver$^{\textbf{CrabbeDuchier:04}}$

A set of principles

XMG offers several sets of additionnal constraints over the models (principles):

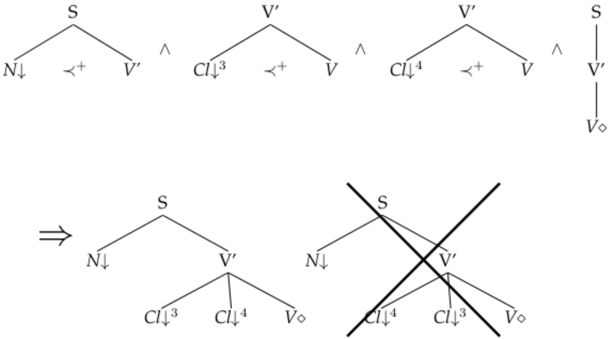
- colors: polarities for node unification
- rank: linear order constraints on nodes
- unicity: uniqueness of a feature inside a model

Rank: Clitics ordering

- The ordering of clitic pronouns (in Spanish or French for example) is known to be problematic when formalizing a grammar
- In a metagrammar, when combining fragments, nodes representing these clitics have to come in a specific order

- Pedro nos la da
- *Pedro la nos da
- Je le lui laisse
- *Je lui le laisse

Rank: Clitics ordering (in French)



- Every produced model has to satisfy the order constraint

Using principles: rank

```
use rank with () dims (syn)
type RANK=[1..7]
property rank: RANK
```

```
class CliticIobjectII
import nonReflexiveClitic[]
{
  <syn>{
    node xCl(rank=2)
            [top=[func=iobj, pers = @{1,2}]]
  }
}
```

Using principles: unicity

```
use unicity with (rank=1) dims (syn)
use unicity with (rank=2) dims (syn)
use unicity with (rank=3) dims (syn)
use unicity with (rank=4) dims (syn)
use unicity with (rank=5) dims (syn)
use unicity with (rank=6) dims (syn)
use unicity with (rank=7) dims (syn)
```

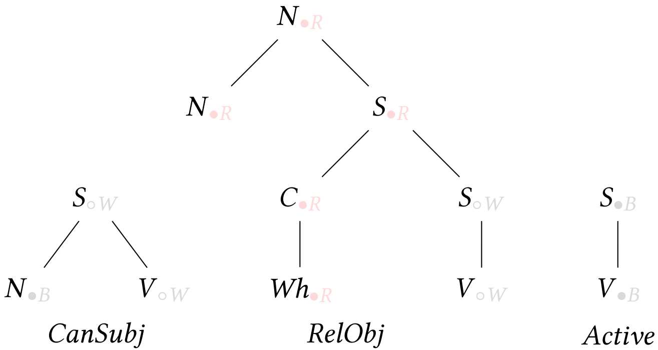
Using principles: colors

- Colors are a solution to guide the combination of fragments
- A color is affected to every node
- New constraints on node unification

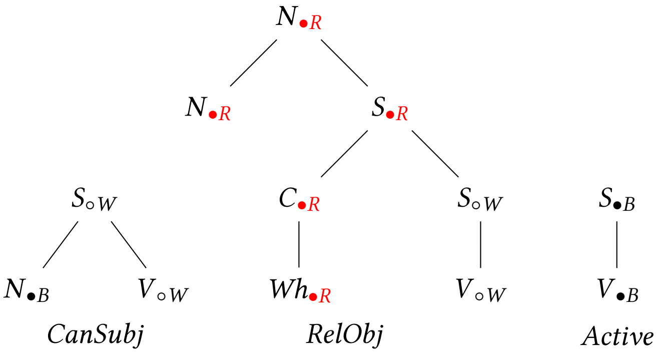
	$\bullet_{B}$	$\bullet_{R}$	$\circ_{W}$	$\perp$
$\bullet_{B}$	$\perp$	$\perp$	$\bullet_{B}$	$\perp$
$\bullet_{R}$	$\perp$	$\perp$	$\perp$	$\perp$
$\circ_{W}$	$\bullet_{B}$	$\perp$	$\circ_{W}$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

- Valid models only have red and black nodes

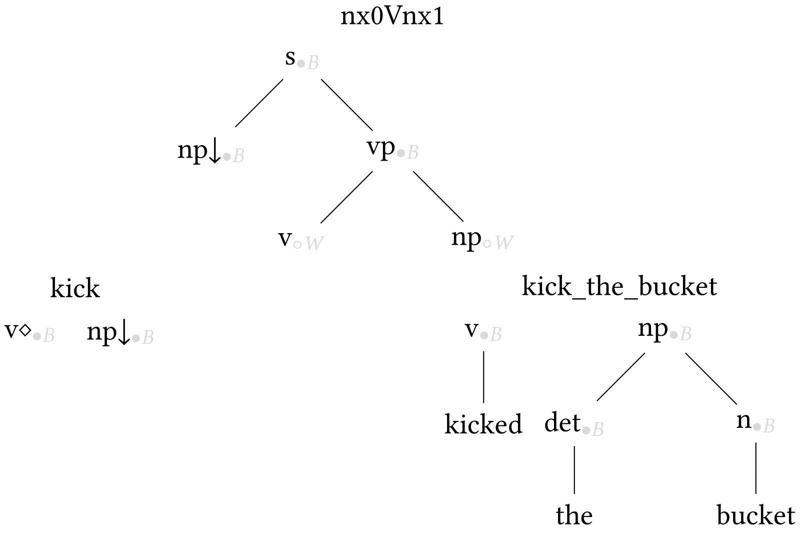
Combination with polarities



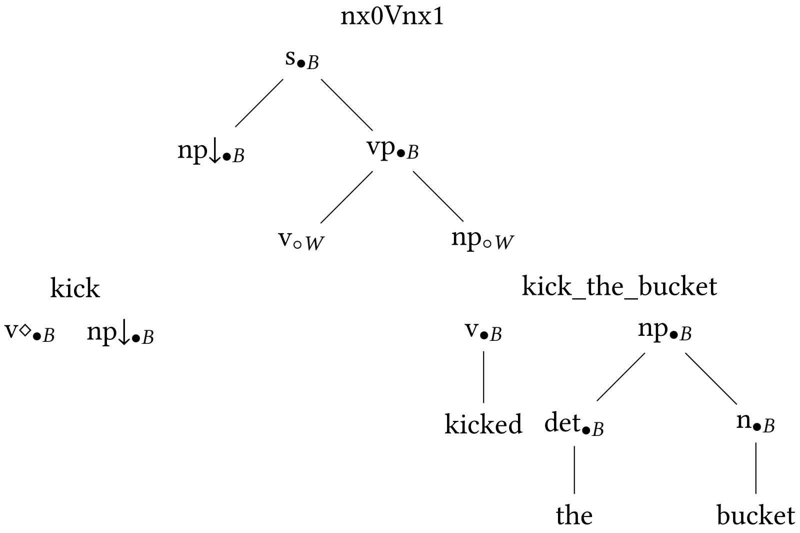
Combination with polarities



Combination with polarities



Combination with polarities



Using principles: colors

```
use color with () dims (syn)
type COLOR={red,black,white}
property color: COLOR
```

```
class nx0Vnx1
declare ?S ?NP_Subj ?VP ?V ?NP_Obj
{
  <syn>{
     ?S (color=red)[cat=s] {
       ?NP_Subj (color=black, mark=subst) [cat=np]
       ?VP (color=black)[cat=vp] {
         ?V (color=white)[cat=v]
         ?NP_Obj (color=white)[cat=np]
       }
     }
  }
}
```