XMG: a tool for implementing frames

Timm Lichte[1], Simon Petitjean[2], Laura Kallmeyer[1] &
Younes Samih[1]

[1]University of Düsseldorf, Germany, and [2]Université d'Orléans, France

CTF14, August 26, 2014

Table of contents

Introduction

Goal of this work: Develop a tool that allows to

1. specify frames where frames are taken to be typed base-labeled feature structures:
   - Certain nodes in the frame can be labeled (in order to access them directly),
   - a frame node has (possibly several) types,
   - and frames need not have a unique root but every frame node is accessible from a labeled node via a path.
2. specify constraints on frames such as
   - possible types, attributes required for them and the types of their values,
   - path identities required for certain types,
   - subtype relations (= type entailments) and incompatibilities of types.

Introduction

The tool is part of XMG (Crabbé et al., 2013). XMG allows to specify grammatical structures of various types: trees, semantic representations, frames ... These different structure specifications can be used in combination, for instance in the context of a syntax-semantics interface in LTAG (Lexicalized Tree Adjoining Grammar).

Besides this, the frame component can be used on its own for

- the implementation of single frames;
- the unification of frames (= identification of their roots);
- more general the identification of nodes from different frames, for instance for argument insertion;
- the development of theories in the form of constraints on frames that can be compiled out into the different possible object frames predicted by the theory.

Signature and frame descriptions

To a large extent, the XMG frame component (Lichte and Petitjean, to appear) implements the feature logic for base labeled feature structures proposed in (Kallmeyer and Osswald, 2013).

A frame signature is a tuple $\langle A, T, B \rangle$ with a finite set of attributes (or features) A, a finite set of elementary types T, and an infinitely countable set of base labels B.

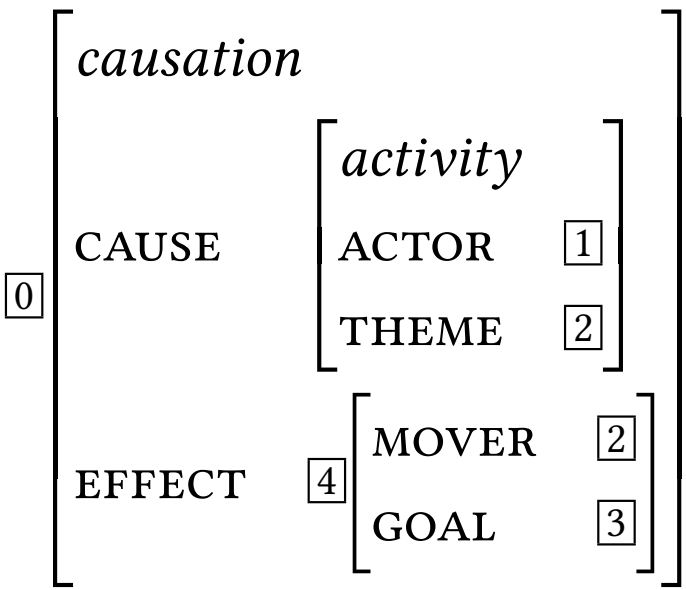
Encoding in XMG:
`frame_attributes = {...}`
`frame_types = {...}`

Special types are $\top$ (`true` or `+` in XMG) and $\bot$ (`false` or `-` in XMG).

Base labels can be any XMG variable (= expression starting with a `?`), for instance `?X0, ?X1, ?root, ...`.

Signature and frame descriptions

The syntax of frame descriptions in XMG is close to standard AVM representations:




```
<frame>{
?X0[causation,
    cause:[activity,
           actor:?X1,
           theme:?X2],
    effect:?X4[mover:?X2,
               goal:?X3]
]}
```

Signature and frame descriptions

Important: In XMG, we *describe frames*, we do not give object frames. The latter are obtained by compilation.

XMG allows to use conjunction and disjunction within the descriptions. But neither negation nor quantification.

Example:

```
<frame>{
    ?X0[causation,
        cause:[activity,
               actor:?X1,
               theme:?X2[ball]],
        effect:[motion,
                mover:?X2]];
    {?X1[John] | ?X1[Mary]}}
```

Signature and frame descriptions

When compiling this description with XMG (under appropriate type constraints, e.g. nothing can be *Mary* and *John* at the same time), we obtain the two object frames

$$\boxed{0}\begin{bmatrix} \textit{causation} & & \\ \textsc{cause} & \begin{bmatrix} \textit{activity} & \\ \textsc{actor} & \boxed{1}\begin{bmatrix}\textit{Mary}\end{bmatrix} \\ \textsc{theme} & \boxed{2}\begin{bmatrix}\textit{ball}\end{bmatrix} \end{bmatrix} \\ \textsc{effect} & \begin{bmatrix} \textit{motion} & \\ \textsc{mover} & \boxed{2}\begin{bmatrix}\textit{ball}\end{bmatrix} \end{bmatrix} \end{bmatrix} \qquad \boxed{0}\begin{bmatrix} \textit{causation} & & \\ \textsc{cause} & \begin{bmatrix} \textit{activity} & \\ \textsc{actor} & \boxed{1}\begin{bmatrix}\textit{John}\end{bmatrix} \\ \textsc{theme} & \boxed{2}\begin{bmatrix}\textit{ball}\end{bmatrix} \end{bmatrix} \\ \textsc{effect} & \begin{bmatrix} \textit{motion} & \\ \textsc{mover} & \boxed{2}\begin{bmatrix}\textit{ball}\end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Type hierarchy and frame constraints

Besides frame descriptions, one can express universal constraints on frames in XMG. These can be

- **sub-type relations** (= type hierarchy), for instance `activity -> event` ("every *activity* is an *event*")
- **type incompatibilities**, for instance `[event,person] -> false` ("nothing can be an *event* and a *person*")
- **attribute specifications for certain types**, for instance `motion -> mover:+` ("every *motion* has an attribute MOVER") `causation -> cause:event` ("every *causation* has an attribute CAUSE with a value of type *event*")
- **path identities for certain types**, for instance `[activity,motion] -> actor=mover` ("in a *motion* that is an *activity*, the ACTOR and MOVER are identical")

Type hierarchy and frame constraints

Currently, XMG provides the following syntax for implementing this:
`frame_constraints = {...}`

Example:

```
frame_constraints = {
    activity -> event,
    activity -> actor: +,
    motion -> event,
    motion -> mover: +,
    [activity,motion] <-> locomotion,
    locomotion -> actor=mover,
    [house,event] -> false,
    [John,event] -> false,
    [John,house] -> false
    }
```

Type hierarchy and frame constraints

Besides the `frame_constraints = ...` statement, one can also specify the type hierarchy in a more compact way. (This is not yet supported by the implementation but will be included soon.)

```
frame_type_hierarchy = {
    [event, [activity, actor:+, [locomotion]],
            [motion, mover:+, [locomotion]]]
    }
```

which is equivalent to

```
frame_constraints = {
     activity -> event, activity -> actor: +,
     motion -> event, motion -> mover: +,
     locomotion -> activity, locomotion -> motion
     }
```

Frame unification with XMG

In order to unify two frames, we can give descriptions of both with identical labels for their root nodes:

$$\begin{bmatrix} \textit{activity} & \\ \textsc{actor} & \begin{bmatrix} \textit{John} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \textit{motion} & \\ \textsc{goal} & \begin{bmatrix} \textit{house} \end{bmatrix} \end{bmatrix}$$

```
<frame>{
 ?root[activity,
    actor:[John]];
 ?root[motion,
    goal:[house]]}
```

Compilation of this description under the frame constraints from slide 9 yields

$$\begin{bmatrix} \textit{activity, motion, locomotion, event} & \\ \textsc{actor} & \boxed{1}\begin{bmatrix} \textit{John} \end{bmatrix} \\ \textsc{mover} & \boxed{1}\begin{bmatrix} \textit{John} \end{bmatrix} \\ \textsc{goal} & \boxed{2}\begin{bmatrix} \textit{house} \end{bmatrix} \end{bmatrix}$$

Frame unification with XMG

- Unification in the previous example amounted to identifying root nodes.
- Any other labeld nodes in frames can be identified in the same way.
- In particular, argument slots can be identified with the argument filling frames.

```
<frame>{
   ?X0[motion,
       mover:?X1,
       goal:?X2];
   ?X1[John];
   ?X2[house]}
```

Frame descriptions and underspecification

- In general, frame descriptions in XMG can be highly underspecified and therefore lead to more than one object frame (= minimal model).
- In particular, frame descriptions can be used as a kind of abstract meta-frames where the object frames are instances of these meta-frames.
- One application could be the definition of *classificatory frames* for scientific theory, as pursued in the CRC 991 by Gerhard Schurz.

Frame descriptions and underspecification

A highly simplified example provided by Gerhard Schurz:

Theory of *fowls*:

- Fowls have a beak that is either round or pointed.
- Fowls have legs that are either short or long.
- Fowls have feet that are either webbed or unwebbed.
- The beak of a fowl is round if and only if its legs are short.

We can express this in a *fowl* frame description in XMG.

Frame descriptions and underspecification

Incompatibility constraints:

```
frame_constraints = { [pointed,round] -> false,
                      [long,short] -> false,
                      [unwebbed,webbed] -> false }
```

Classificatory frame for *fowl*:

```
<frame>{
  [fowl,
   beak:?B,
   leg:?L,
   foot:?F];
  {?B[pointed] | ?B[round]};
  {?L[long]|?L[short]};
  {?F[unwebbed]|?F[webbed]};
  {?B[round] AND ?L[short] | ?B[pointed] AND ?L[long]}}
```

Frame descriptions and underspecification

Checking the predictions of such a theory can be done by compiling out the corresponding frame description.

In the *fowl* case, this gives four possible *fowl* frames.

$$\begin{bmatrix} \textit{fowl} & \\ \textsc{beak} & \begin{bmatrix}\textit{round}\end{bmatrix} \\ \textsc{leg} & \begin{bmatrix}\textit{short}\end{bmatrix} \\ \textsc{foot} & \begin{bmatrix}\textit{webbed}\end{bmatrix} \end{bmatrix} \quad \begin{bmatrix} \textit{fowl} & \\ \textsc{beak} & \begin{bmatrix}\textit{round}\end{bmatrix} \\ \textsc{leg} & \begin{bmatrix}\textit{short}\end{bmatrix} \\ \textsc{foot} & \begin{bmatrix}\textit{unwebbed}\end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \textit{fowl} & \\ \textsc{beak} & \begin{bmatrix}\textit{pointed}\end{bmatrix} \\ \textsc{leg} & \begin{bmatrix}\textit{long}\end{bmatrix} \\ \textsc{foot} & \begin{bmatrix}\textit{webbed}\end{bmatrix} \end{bmatrix} \quad \begin{bmatrix} \textit{fowl} & \\ \textsc{beak} & \begin{bmatrix}\textit{pointed}\end{bmatrix} \\ \textsc{leg} & \begin{bmatrix}\textit{long}\end{bmatrix} \\ \textsc{foot} & \begin{bmatrix}\textit{unwebbed}\end{bmatrix} \end{bmatrix}$$

Frame descriptions and underspecification

So far, dependencies between attribute values of the type "*beak* is ROUND iff *leg* is SHORT" are not supported by XMG but they will be included.

Such constaints will then be possible in the universal `frame_constraints`:

```
frame_constraints = { [pointed,round] -> false,
                      [long,short] -> false,
                      [unwebbed,webbed] -> false,
                      beak:round <-> leg:short}
```

XMG: eXtensible MetaGrammar

- The frame implementation tool presented so far is part of XMG.
- XMG (eXtensible MetaGrammar, Crabbé et al. 2013) stands both for metagrammatical descriptions and the compiler for these descriptions.
- The descriptions are organized into *classes*, that can be reused (i. e. "imported" or instantiated) by other classes.
- Borrowing from object oriented programming, classes are encapsulated, which means that each class can handle the scopes of their variables explicitly, by declaring variables and choosing which ones to make accessible for (i. e. to "export to") other instantiating classes.

XMG: eXtensible MetaGrammar

- Crucial elements of a class are the so-called *dimensions*. Dimensions can be equipped with specific description languages and are compiled independently, thereby enabling the grammar writer to treat the levels of linguistic information separately.
- The standard dimensions are `<syn>` for syntax, and `<sem>` for semantics.
  More recently, a dimension `<morph>` for morphological descriptions has been added (Duchier et al., 2012; Lichte et al., 2013).
- `<frame>` is the dimension used for frames.
- Crucially, variables can be shared between dimensions since they are local to the class. This way, interfaces can be modeled.

XMG: eXtensible MetaGrammar

```
class rolling
export ?X0
declare ?X0 ?X1
{<frame>{
 ?X0[motion,
   theme:?X1,
   mover:?X1,
   manner:[rolling]]}}
```

```
class directionalPP
export ?X0
declare ?X0
{<frame>{
    ?X0[motion,
      goal:+]}}
```

Class for
directed motion:

```
class directedMotion
declare ?Frame1 ?Frame2
{   ?Frame1 = rolling[];
    ?Frame2 = directionalPP[];
    ?Frame1.?X0 = ?Frame2.?X0
```

Implementation

- Currently, the frame component of XMG is already running but is still under implementation. The examples shown in this talk are supported by the XMG compiler.
- In the near future, we will develop a web-based GUI that allows to use XMG, in particular the frame component:
  - Frame constraints can be entered with the syntax shown above.
  - Frame descriptions can be defined via a graphical user interface.
  - Object frames resulting from compilation can be viewed via the GUI.
  - The XMG Frame Viewer will make use of the latest browser-based technologies (HTML5, XSLT, XML, MathML, SVG, and Javascript) to render AVMs.
  - Example of AVM viewer: slide8.html

Conclusion

- We have presented a frame implementation tool that is part of XMG.
- The XMG frame component can be used in the context of grammar development, pairing for instance frames with syntactic trees.
- It can also be used on its own for frame implementation.
- A web-based GUI that allows an easy use of the tool is planned for the near future.

Crabbé, B., Duchier, D., Gardent, C., Le Roux, J., and Parmentier, Y. (2013). XMG: eXtensible MetaGrammar. *Computational Linguistics*, 39(3):1–66.

Duchier, D., Ekoukou, B. M., Parmentier, Y., Petitjean, S., and Schang, E. (2012). Describing morphologically rich languages using metagrammars: a look at verbs in Ikota. In *Workshop on Language Technology for Normalisation of Less-Resourced Languages (SALTMIL 8 - AfLaT 2012)*, pages 55–59.

Kallmeyer, L. and Osswald, R. (2013). Syntax-driven semantic frame composition in Lexicalized Tree Adjoining Grammar. *Journal of Language Modelling*, 1:267–330.

Lichte, T., Diez, A., and Petitjean, S. (2013). Coupling trees, words and frames through XMG. In *Proceedings of the ESSLLI 2013 workshop on High-level Methodologies for Grammar Engineering*.

Lichte, T. and Petitjean, S. (to appear). Adding semantic frames to XMG. *Journal of Language Modelling*.