A Comparative Introduction to XDG: Adding the Predicate-Argument Dimension

Ralph Debusmann

and

Denys Duchier

Programming Systems Lab, Saarland University, Saarbrücken, Germany

and

Équipe Calligramme, LORIA, Nancy, France

This presentation

- adding the Predicate-Argument (pa) dimension to the example grammar
- new:
  - type definitions
  - one-dimensional principles (dag, valency)
  - multi-dimensional principles (linking)
  - lexical classes

Defining the new types

- edge labels:

```
deftype "pa.label" {arg1 arg2 arge del root}

deflabeltype "pa.label"
```

- lexical entries:

```
deftype "pa.entry" {in: valency("pa.label")
                    out: valency("pa.label")}

defentrytype "pa.entry"
```

Class of models, valency

```
useprinciple "principle.graph" {
  dims {D: pa}}

useprinciple "principle.dag" {
  dims {D: pa}}

useprinciple "principle.valency" {
  dims {D: pa}
  args {In: _.D.entry.in
        Out: _.D.entry.out}}
```

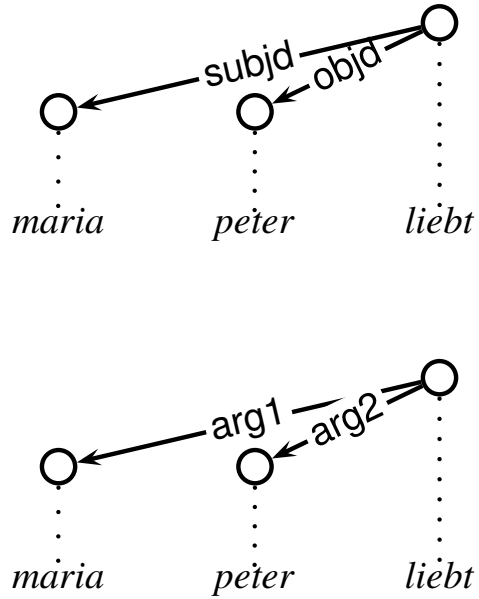
Extending the multi dimension

- add lexical attributes for multi-dimensional principles

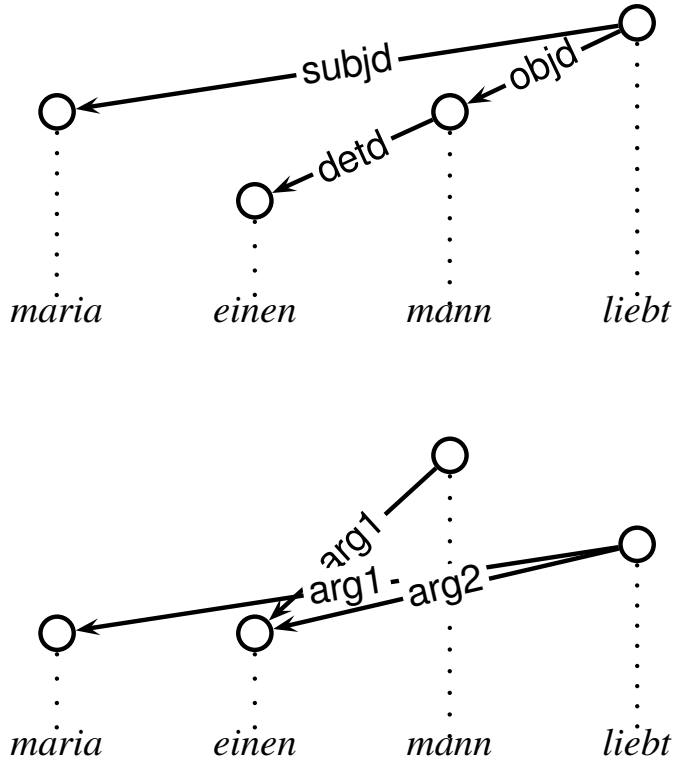
```
defentrytype {%% id/lp multi-dimensional attributes
              blocks_lpid: set("id.label")
              %% ds/id multi-dimensional attributes
              link2_dsid: map("ds.label" iset("id.label"))
              link2_idds: map("id.label" iset("ds.label"))
              %% pa/ds multi-dimensioal attributes
              link1_pads: map("pa.label" set("ds.label"))
              link2_pads: map("pa.label" iset("ds.label"))}
```

- instantiate multi-dimensional principles
  - realize semantic by deep syntactic arguments: linking principle (pa/ds)

Realizing semantic by deep syntactic arguments 1



Realizing semantic by deep syntactic arguments 2



Realizing semantic by deep syntactic arguments 3

```
useprinciple "principle.linking" {
  dims {D1: pa
        D2: ds
        Multi: multi}
  args {Link1: _.Multi.entry.link1_pads
        Link2: _.Multi.entry.link2_pads}}
```

- from pa to ds dimension
- declarative semantics:

$$h \xrightarrow{l}_1 d \quad \Rightarrow \quad (F_1(l) \neq \emptyset \;\Rightarrow\; l' \in F_1(l) \;\wedge\; h \xrightarrow{l'}_2 \ldots \rightarrow_2 d) \wedge (l'' \in F_2(l) \wedge \xrightarrow{l''}_2 d)$$

- $F_1 = \texttt{Link1}$ and $F_2 = \texttt{Link2}$

Lexicon

- lexical classes:
  - new lexical classes to specify pa and ds/pa properties
  - update existing lexical classes to inherit from them
- lexical entries:
  - apply the updated lexical classes

Defining new lexical classes: root_pa, part_pa

```
defclass "root_pa" {
  dim pa {in: {}
          out: {root* del*}}}
```

- *the additional root node collects arbitrary many roots, and arbitrary many deleted nodes*

```
defclass "part_pa" {
  dim pa {in: {del!}}}
```

- *particles are deleted*

Defining new lexical classes: cont, nocont

```
defclass "cont" {
  dim pa {in: {root!|arge!}}}
```

- *words with semantic content, i.e. present on the pa dimension*

```
defclass "nocont" {
  dim pa {in: {del!}}}
```

- *words with no semantic content, i.e. deleted on the pa dimension*

Defining new lexical classes: cnoun_pa, det_pa

```
defclass "cnoun_pa" {
  dim pa {in: {root!}
          out: {arg1!}}
  dim multi {link2_pads: {arg1: {detd}}}}
```

- *a common noun must be a root and requires an argument realized by its determiner*

```
defclass "det_pa" {
  dim pa {in: {arg1* arg2*}}}
```

- *determiners can be arguments of arbitrary many other nodes*

Updating lexical classes: cnoun

```
defclass "cnoun" Word Agrs {
  "cnoun_id"
  "cnoun_lp"
  "cnoun_ds"
  "cnoun_pa"
  dim id {agrs: Agrs}
  dim lex {word: Word}}
```

- *a common noun inherits from the classes for common nouns on the id, lp, ds and pa dimensions, has agreements* `Agrs` *and word form* `Word`

Updating lexical classes: det

```
defclass "det" Word Agrs {
  "det_id"
  "det_lp"
  "det_ds"
  "det_pa"
  dim id {agrs: Agrs}
  dim lex {word: Word}}
```

- *a determiner inherits from the classes for common nouns on the id, lp, ds and pa dimensions, has agreements* `Agrs` *and word form* `Word`

Defining new lexical classes: arg1subjd, arg1

```
defclass "arg1subjd" {
  dim pa {out: {arg1!}}
  dim multi {link1_pads: {arg1: {subjd}}
             link2_pads: {arg1: {subjd detd}}}}
```

- *require an* `arg1` *realized by the deep subject or a determiner below the deep subject*

```
defclass "arg1" {
  "subjdc"
  "arg1subjd"}
```

- *require a deep subject to realize* `arg1`

Defining new lexical classes: arge

```
defclass "arge" Label {
  "vcdLabel" {Label: Label}
  dim pa {out: {arge!}}
  dim multi {link2_pads: {arge: {vcd}}}}
```

- *require an event argument realized by the deep verbal complement*

Updating lexical classes: subjraising, subjcontrol

```
defclass "subjraising" {
  "cont"
  "arge" {Label: vinf}
  "subjsubj"}
```

- *a subject raising verb has semantic content, requires an event argument, and realizes its surface subject by a deep subject*

```
defclass "subjcontrol" {
  "subjraising"
  "arg1"}
```

- *a subject control verb is just like a subject raising verb, and in addition it requires an* `arg1`

Updating lexical classes: objcontrol

```
defclass "objcontrol" {
  "cont"
  "arge" {Label: vinf}
  "objsubj"
  "arg1"
  "arg2"}
```

- *an object control verb has semantic content, requires an event argument, its surface object realizes a deep subject, and it requires* `arg1` *and* `arg2`

Applying the updated lexical classes: raising

```
defentry {
  "subjraising"
  "mainverb" {Word1: "scheint"
              Word2: "scheinen"
              Word3: "geschienen"}}
```

Applying the updated lexical classes: control

```
defentry {
  "subjcontrol"
  "mainverb" {Word1: "versucht"
              Word2: "versuchen"
              Word3: "versucht"}}

defentry {
  "objcontrol"
  "mainverb" {Word1: "ueberredet"
              Word2: "ueberreden"
              Word3: "ueberredet"}}
```