A Comparative Introduction to XDG: Adding the Scope Dimension

Ralph Debusmann

and

Denys Duchier

Programming Systems Lab, Saarland University, Saarbrücken, Germany

and

Équipe Calligramme, LORIA, Nancy, France

This presentation

- adding the SCope (sc) dimension to the example grammar
- new:
  - type definitions
  - one-dimensional principles (tree, valency)
  - multi-dimensional principles (Idominance)
  - lexical classes

Defining the new types

- edge labels:

```
deftype "sc.label" {r s a del root}

deflabeltype "sc.label"
```

- lexical entries:

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

defentrytype "sc.entry"
```

Instantiating the sc principles

- all principles re-used from the other dimensions (id, lp, ds, pa):
  - class of models: graph principle, tree principle
  - scope valency

Class of models, scope valency

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

useprinciple "principle.tree" {
  dims {D: sc}}

useprinciple "principle.valency" {
  dims {D: sc}
  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-dimensional attributes
              link1_pads: map("pa.label" set("ds.label"))
              link2_pads: map("pa.label" iset("ds.label"))
              %% sc/pa multi-dimensional attributes
              lcodom_pasc: map("pa.label" set("sc.label"))
              lcontradom_pasc: map("pa.label" set("sc.label"))}
```

- instantiate multi-dimensional principles:
  - inducing dominance relationships: ldominance principle (pa/sc)

Inducing dominance relationships

```
useprinciple "principle.ldominance" {
  dims {D1: pa
        D2: sc
        Multi: multi}
  args {LCodom: _.Multi.entry.lcodom_pasc
        LContradom: _.Multi.entry.lcontradom_pasc}}
```

- from pa to sc dimension
- declarative semantics:

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

Lexicon

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

Defining new lexical classes: root_sc, part_sc

```
defclass "root_sc" {
  dim sc {in: {}
          out: {root* del*}}}
```

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

```
defclass "part_sc" {
  dim sc {in: {del!}}}
```

- *particles are deleted*

Defining new lexical classes: cont, nocont

```
defclass "cont" {
  dim pa {in: {root!|arge!}}
  dim sc {in: {r? s? a? root?}}}
```

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

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

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

Defining new lexical classes: cnoun_sc, det_sc

```
defclass "cnoun_sc" {
  dim sc {in: {r? s? root?}}}
```

- *a common noun can either be in the restriction or scope of another node, or it can be root*

```
defclass "det_sc" {
  dim sc {in: {r? s? root?}
          out: {r! s!}}}
```

- *determiners can either be in the restriction or scope of another node, or it can be root, and they have a restriction and a scope*

Updating lexical classes: cnoun

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

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

Updating lexical classes: det

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

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

Updating lexical classes: arg1subjd

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

- *require an* `arg1` *realized by the deep subject or a determiner below the deep subject, and is s-dominated by the* `arg1`

Defining new lexical classes: arge

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

- *require an event argument realized by the deep verbal complement, and a-dominate it*