ESSLLI'99, Utrecht, August 9–22, 1999
M. Butt, A. Frank, J. Kuhn:
Development of large scale LFG grammars:
Linguistics, Engineering and Resources

Templates and Lexical Rules Generalizations, Transparency and Maintainability

Anette Frank
XRCE Grenoble

Anette.Frank@xrce.xerox.com

- Capture linguistic generalizations within a grammar and across grammars
- Templates in XLE
- Lexical rules in XLE
- Grammar engineering aspects

Linguistic generalizations within a grammar and across grammars

– Examples –

- If a clause is specified for (any) tense, the clause is finite.

  $\forall f.\ ((f\ \ \text{tns-asp}\ \ \text{tense}) \rightarrow (f\ \ \text{fin})=\ +)$

- If a clause is finite, its subject bears nominative case (in some languages).

  $\forall f.\ ((f\ \text{fin})=\ + \rightarrow (f\ \text{subj case})=\ \text{nom})$

- If a clause is finite, the subject agrees with person and number features of the verb (in some languages).

  $$\forall f.\ ((f\ \text{fin})=\ + \rightarrow \begin{array}{l}(f\ \text{subj num})=\ (f\ \text{num})\\ (f\ \text{subj pers})=\ (f\ \text{pers}))\end{array}$$

- Transitive verbs may passivize:

  $$\begin{array}{ll}\forall f.(& (f\ \text{pred})=\ \text{‘\%stem}\ \langle (f\ \text{subj})(f\ \text{obj})\rangle\text{’}\\ & (f\ \text{passive})=\ \text{-}\\ \rightarrow & \\ & (f\ \text{pred})=\ \text{‘\%stem}\ \langle\ \text{null}\ (f\ \text{subj})\rangle\text{’}\\ & (f\ \text{passive})=\ +\ )\end{array}$$

Encoding linguistic generalizations

- with templates

  tense(_t) = $(\uparrow$ tns-asp tense)= _t
  $(\uparrow$ fin)= +.

  fin = @(assign-case SUBJ nom).

  assign-case(_gf _c) = $(\uparrow$ _gf case)= _c.

- or with lexical rules

  $(\uparrow$passive)$=_c$ +
  $(\uparrow$obj) $\rightarrow$ $(\uparrow$subj)
  $(\uparrow$subj) $\rightarrow$ null

Templates in XLE

Templates are defined as:

`template_name= template_body.` or
`template_name(_par1, .., _parn)= template_body.`

A template body can consist of

– functional descriptions (↑attr)= val and/or
– template calls

A template may specify optional parameters.

Parameters can be used in the template body as
– values of attributes or
– as attributes (paths) in functional descriptions, or else
– as parameters in embedded template calls.

Though complex template hierarchies may be defined, our template hierarchies do not represent an exhaustive model of grammatical description.

Templates can be called in lexical entries and in functional annotations of phrase structure rules.

Template calls are specified as follows:

`@template_name` or
`@(template_name _par1 ... _parn)`

Examples

Template call in lexical entry:

kommt V $(\uparrow$pred$) =$ 'kommen$\langle$ $(\uparrow$subj$)\rangle$'
@(tense_fin present 3 sg).

Template definitions:

tense_fin(_t _pers _num)= @(tense _t)
@(fin _pers _num).

tense(_t)= $(\uparrow$ tns-asp tense$)=$ _t.

fin(_pers _num)= $(\uparrow$ fin$)= +$
@(num subj _num)
@(pers subj _pers).

num(_gf _num) = $(\uparrow$ _gf num$)=$ _num.
pers(_gf _pers) = $(\uparrow$ _gf pers$)=$ _pers.

Template call in PS rule annotations:

– Noun–adjective agreement (French)

*un grand/*e chien*
les grand(s) chiens*

$$\begin{array}{lll} \text{NPap} \longrightarrow & \text{AP:} & \downarrow \ \$ \ (\uparrow \ \text{adjunct}) \\ & & @(\text{num\_gend\_agr} \downarrow) \\ & \text{N:} & \uparrow \ = \downarrow. \end{array}$$

$$\begin{array}{ll} \text{num\_gend\_agr(\_path) =} & (\text{\_path num}) = (\uparrow \ \text{num}) \\ & (\text{\_path gend}) = (\uparrow \ \text{gend}). \end{array}$$

– Anaphoric control

Templates are easy to define.
Defining these templates was easy.
It is sufficient to press and hold the start button.

$$\begin{array}{lll} \ldots \rightarrow & \ldots & \\ & \text{VPinf} & (\uparrow \ \text{comp}) = \downarrow \\ & & @(\text{null-pron } (\downarrow\text{subj})) \end{array}$$

Template definition:

$$\begin{array}{ll} \text{null-pron(\_path) =} & (\text{\_path pred}) = \text{'pro'} \\ & (\text{\_path pron-type}) = \text{null}. \end{array}$$

Lexical Rules in XLE

In XLE, lexical rules (designator rewrites) can only be defined in template definitions:

Example:

```
trans( _P ) =     @(pass (↑ pred)= '_P⟨(↑ subj)(↑ obj)⟩').
ditrans( _P ) =   @(pass (↑ pred)= '_P⟨(↑ subj)(↑ obj)(↑ obj2)⟩').

pass( schemata ) =  { schemata
                        @nopassive
                     | schemata
                        @passive
                        { (↑ subj) → (↑ obl_ag)
                          | (↑ subj) → null}
                        (↑ obj) → (↑ subj)}.

@nopassive =   (↑ vform) = part
               (↑ passive)= -.
@passive =     (↑ vform)=c part
               (↑ passive)= +.
```

Expansion of `@(TRANS kick)`

– substitution of template parameters of `TRANS`:

@(pass (↑ pred)= 'kick⟨(↑ subj)(↑ obj)⟩').

– substitution of template parameters of `PASS`:

```
{   (↑ pred)= 'kick⟨(↑ subj)(↑ obj)⟩'
    @nopassive
 |  (↑ pred)= 'kick⟨(↑ subj)(↑ obj)⟩'
    @passive
    { (↑ subj) → (↑ obl_ag)
    | (↑ subj) → null}
    (↑ obj) → (↑ subj)}.
```

– conversion to disjunctive normal form:

```
{ (↑ pred)= 'kick⟨ (↑ subj)(↑ obj)⟩'
  (↑ vform) = part
  (↑ passive)= -
| (↑ pred)= 'kick⟨ (↑ subj)(↑ obj)⟩'
  (↑ vform)=c part
  (↑ passive)= +
  (↑ subj) → (↑ obl_ag)
  (↑ obj) → (↑ subj)
| (↑ pred)= 'kick⟨ (↑ subj)(↑ obj)⟩'
  (↑ vform)=c part
  (↑ passive)= +
  (↑ subj) → null
  (↑ obj) → (↑ subj)}.
```

– executing designator rewrites **match** → **result**:

$$
\begin{array}{l}
\{\ (\uparrow\ \text{pred}) = \text{'kick}\langle\ (\uparrow\ \text{subj})(\uparrow\ \text{obj})\rangle\text{'} \\
\quad (\uparrow\ \text{vform}) = \text{part} \\
\quad (\uparrow\ \text{passive}) = \text{-} \\
\ |\ (\uparrow\ \text{pred}) = \text{'kick}\langle\ (\uparrow\ \text{obl\_ag})(\uparrow\ \text{subj})\rangle\text{'} \\
\quad (\uparrow\ \text{vform}) =\text{c part} \\
\quad (\uparrow\ \text{passive}) = + \\
\ |\ (\uparrow\ \text{pred}) = \text{'kick}\langle\ \text{null}\ (\uparrow\ \text{subj})\rangle\text{'} \\
\quad (\uparrow\ \text{vform}) =\text{c part} \\
\quad (\uparrow\ \text{passive}) = +\}.
\end{array}
$$

Grammar Engineering Aspects

- **Maintenance** of feature naming conventions is facilitated by abbreviatory devices

  Base templates: local changes of naming conventions

  case(_c) = (↑ case)= _c.

  fin = (↑ fin)= +.
  inf = (↑ inf)= +.

  tense(_t) = (↑ tns-asp tense)= _t.

- **Consistency** in lexica and grammars:

  lex1 .. (↑ inf)= + . . . lexn ... (↑ fin)= -

  lex1 ... @inf. . . . lexn ... @inf.

- **Change once**
  . . . and propagate throughout grammar and lexica!

  old: inf = (↑ fin)= -.
  new: inf = (↑ inf)= +.

  old: adj-rel = (↑ adjunct-rel) = ↓
  (↓ topic rel)$=_c$ +.
  new: adj-rel = ↓ $ (↑ adjunct)
  (↓ topic rel)$=_c$ +.

- **Minimizing encoding errors** (mistyping, etc.), inconsistencies and omissions by use of templates and template hierarchies (linguistic classes)

kommt	$(\uparrow$ pred$)$ = ‘kommen$\langle$ $(\uparrow$ subj$)\rangle$’
	$(\uparrow$ fin$) = +$
	$(\uparrow$ tns-asp tense$)=$ present
	$(\uparrow$ subj case$)=$ nom
	$(\uparrow$ subj pers$)= 3$
	$(\uparrow$ subj num$=$ sg
	$(\uparrow$ passive$)=$ -.

kommt ...	@v_intrans
	@(tense_fin present 3 sg).

- **Transparency** across grammars:

  Uniform template naming conventions are recommended.

  Definition of templates may vary across languages (e.g. passive, case marking, etc.).

  Differences in template definitions express/reveal language specific properties.