Das LKB-System & die Matrix

Ein kurzer Überblick

LKB - Die wichtigsten Dateien

lexicon.tdl

Das Lexikon der Grammatik.
Hier werden sämtliche lexikalische Einträge notiert.

```
dog := noun-lxm &
       [ ORTH.LIST.FIRST "dog",
         SEM.KEY.PRED "dog_rel" ].
```

LKB - Die wichtigsten Dateien

inflr.tdl oder irules.tdl

Hier werden Einträge zur Flexion gemacht.
Zum Beispiel Bildung des Past Participles:

```
past-v_irule :=
      %suffix (* ed) (!ty !tied) (e ed) (!t!v!c !t!v!c!ced) (give gave)
      past-verb.
```

Folgende Regel bewirkt, dass die Schreibweise eines Nomens im Singular nicht verändert wird:

```
sg-noun_irule := sing-noun &
              [ ORTH #1,
                ARGS < [ ORTH #1 ] > ].
```

In der Matrix sind diese Regeln Instanzen von (Subtypen) von infl-ltol-rule oder infl-ltow-rule (ltol = lexeme to lexeme, ltow = lexeme to word).

LKB - Die wichtigsten Dateien

lrules.tdl

In dieser Datei stehen lexikalische Regeln.
Diese sorgen zum Beispiel dafür, dass die Schreibweise eines Wortes erhalten bleibt (spelling preserving lexical rules).

In der Matrix sind diese Regeln Instanzen von (Subtypen) von const-ltol-rule oder const-ltow-rule.
D.h. `%suffix` taucht in dieser Datei NICHT auf !

LKB - Die wichtigsten Dateien

types.tdl

Festlegung der Typenhierarchie. Sämtliche Typen, Subtypen sowie Merkmale (Features) werden in diese Datei eingetragen. Es gibt einen Obertyp *top*, von dem alle anderen Typen (direkt oder indirekt) erben.

```
feat-struc := *top*.

syn-struc  := feat-struc &
           [ ORTH *dlist*,
             HEAD pos,
             SPR *list*,
             COMPS *list*,
             SEM semantics,
             ARGS *list* ].

  pernum := feat-struc.
  3sing := pernum.
  non-3sing := pernum.

  pos := feat-struc & [ MOD *list* ].

  nominal := pos & [ AGR pernum ].
```

LKB - Die wichtigsten Dateien

rules.tdl

Diese Datei enthält Regelinstanzen (Phrasen-Struktur-Regeln) die gemäß den Regeltypen (welche z.B. in matrix.tdl oder types.tdl festgelegt sind) konstruiert werden. D.h. die Regeln in rules.tdl erben von einem in matrix.tdl (oder types.tdl) festgelegten Typ.

LKB - Die wichtigsten Dateien

in rules.tdl:

```
head-complement-rule-1 := binary-head-initial &
                      [ SPR #spr,
                        ARGS < word &
                          [ SPR #spr,
                            COMPS < #1 > ],
                            #1 >  ].
```

in types.tdl:

```
binary-head-initial := binary-rule & head-initial.


binary-rule := phrase &
       [ ORTH [LIST #ofront, LAST #otail],
         SEM.RELS [LIST #cfront, LAST #ctail ],
         ARGS < [ ORTH [LIST #ofront, LAST #omiddle ],
                SEM.RELS [LIST #cfront, LAST #cmiddle ] ],
              [ ORTH [LIST #omiddle, LAST #otail ],
                SEM.RELS [LIST #cmiddle, LAST #ctail ] ] > ].


head-initial := phrase &
        [ HEAD #head,
          SEM [ INDEX #index,
                KEY   #key ],
          ARGS < [ HEAD #head,
                   SEM [ INDEX #index,
                         KEY   #key ] ], ... > ].
```

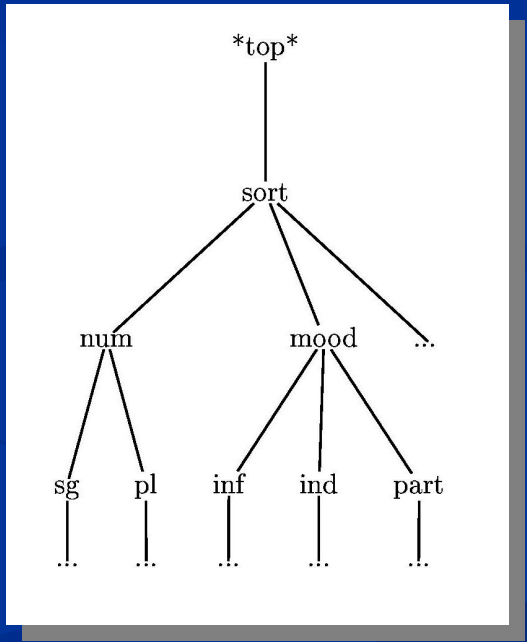
Die head-complement-rule-1 verbindet ein transitives Verb mit seinem Objekt. Sie erbt von der binary-rule und head-initial.

Matrix - Dateien

my-language.tdl = lexikalische Typen
lexicon.tdl = Lexikoneinträge

matrix.tdl = Regeltypen (sämtliche Grundlagen)
rules.tdl = Regelinstanzen

Kleiner Ausschnitt aus der Typhierarchie:



Bsp.:

In matrix.tdl wird `sort` als Untertyp von `*top*` eingeführt.
In my-language.tdl kann man dann z.B. einen Typ `num` definieren, der direkt von sort erbt.

Matrix - Überblick

- **Basic types:** `*top*`

```
avm
instloc (semantic information)
sort
tree-node-label (label name in parse trees)
```

- **Types for Sign, Word, Phrase and Lex-Entry**
- **Affixation**
- **SYNSEM values**
- **LOCAL & NON-LOCAL values: e.g. Types for distinguishing scopal v. intersective modifiers.**
- **CAT values**
- **HEAD & VAL values:**

```
head := head-min &
     [ MOD list ].

valence := valence-min &
         [ SUBJ list,
           SPR list,
           COMPS list,
           SPEC list,
           --KEYCOMP avm ].
```

Matrix - Überblick

- CONT values (types für mrs)
- KEY relations (for semantic selection)
- CTXT values
- Basic Semantic types: e.g.
  ```
  message
  qeq
  proposition
  question
  index
  ```
- Types encoding agreement information: e.g.
  ```
  png (person, numerus, gender)
  tense
  aspect
  mood
  tam (tense, aspect, mood)
  ```

(subtypes and features are highly language dependent !)

Matrix – Überblick

- Basic relation types
- HEAD types (values for head features such as CASE, VFORM, ...)
- kinds of lists: e.g. `olist` (list of optional arguments)

Matrix - Listen

Listen

- Verbindet mehrere Listen miteinander:

```
cons := list &
     [ FIRST *top*,
       REST *top* ].
```

- Der Typ für eine Liste mit null oder einem Element:

```
0-1-list := list.
```

- Eine Liste mit genau einem Element:

```
1-list := 0-1-list & cons &
       [ REST null ].
```

Matrix – Listen

Listen (Fortsetzung)

- **Der Typ für eine leere Liste:**

```
null := 0-1-list.
```

- **Eine Liste mit mehr als einem Element:**

```
1-plus-list := cons &
               [ REST cons ].
```

Matrix – Diff-Listen

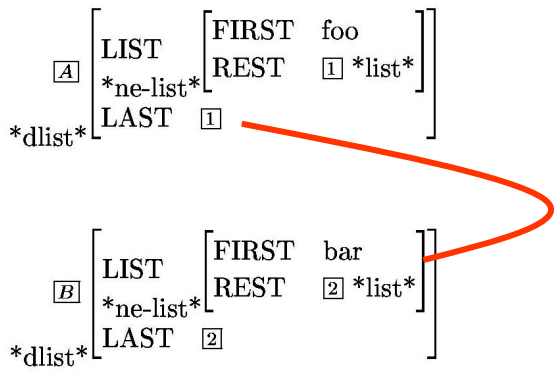
Differenzlisten

Differenzlisten werden z.B. benutzt, um die ORTH-Werte von Wort zu Phrase hochzureichen und durch Listenkonkatenation dadurch die Orthographie einer Phrase zu bekommen.

--> Listenkonkatenation mithilfe von Unifikation

Die Differenzliste wird aufgeteilt in LIST und LAST, wobei LAST mit dem letzten Element von LIST koindiziert ist (Pointer) !

```
diff-list := avm &
             [ LIST list,
               LAST list ].
```



A und B können durch konkateniert werden, indem man den LIST Wert von B mit dem LAST-Wert von A (also 1) unifiziert. Der LAST Wert von B wird somit zum neuen tail des Ergebnisses der Konkatenation.

Matrix – Diff-Listen

Differenzlisten (Fortsetzung)

- **Eine Diff-Liste mit null oder einem Element:**

```
0-1-dlist := diff-list &
          [ LIST 0-1-list ].
```

- **Eine leere Diff-Liste:**
  (LAST zeigt auf das letzte Element der Liste in LIST; das kann es aber nur, wenn LIST null entählt !)

```
0-dlist := 0-1-dlist &
        [ LIST #list,
          LAST #list ].
```

- **Eine Diff-Liste mit genau einem Element:**

```
1-dlist := 0-1-dlist &
        [ LIST 1-list &
            [ REST #rest & null ],
          LAST #rest ].
```

Matrix – Lexical Rules

Lexical Rules

variieren zwischen: **(1) lexeme-to-lexeme oder lexeme-to-word**
(2) spelling changes

- **die Lexeme / Wort Unterscheidung geschieht durch das Merkmal `[ INFLECTED bool ]`**
- ***lexeme-to-word-rule*** **(fügt lediglich SYNSEM-Information hinzu)**

```
lexeme-to-word-rule := lex-rule &
                       [ INFLECTED +,
                         KEY-ARG #keyarg,
                         SYNSEM  #synsem,
                         ROOT    #root,
                         DTR [ INFLECTED -,
                               KEY-ARG #keyarg,
                               SYNSEM  #synsem,
                               ROOT    #root ],
                               C-CONT.LISZT <! !> ].
```

Matrix – Lexical Rules

- ***lexeme-to-lexeme-rule*** **(stärkere Änderungen des SYNSEM Wertes)**

```
lexeme-to-lexeme-rule := lex-rule &
                         [ INFLECTED -,
                           SYNSEM.LOCAL [ CAT [ MC #mc ],
                                          KEYS.MESSAGE #msg ],
                           DTR [ SYNSEM.LOCAL [ CAT [ MC #mc ],
                                                KEYS.MESSAGE #msg ],
                                 INFLECTED - ]].
```

- **Spelling changing rule**

```
inflecting-lex-rule := lex-rule & [ NEEDS-AFFIX + ].
```

→ Hier wird ein Affix benötigt.

- **Spelling-preserving rule**

```
constant-lex-rule := lex-rule &
                     [ STEM #stem,
                       DTR [ STEM #stem ]].
```

→ Der STEM Wert (Orthographie) der Tochter wird nach oben kopiert.

Matrix – PS-Rules

Phrase-Structure-Rules

- **headed-phrase: der HEAD-Wert der HEAD-Tochter ist gleich dem HEAD-Wert der Mutter**
- **non-headed-phrase**
- **basic-unary-phrase (der STEM Wert wird hochkopiert)**
- **basic-binary-phrase**
- **head-initial**
- **head-final**
- **Head-Filler-Phrase**

Matrix – PS-Rules

```
basic-head-filler-phrase := binary-phrase & phrasal &
    [ SYNSEM [ LOCAL [ CAT [ VAL [ COMPS < >,
                                   SPR < anti-synsem > ],
                             POSTHEAD + ] ],
               NON-LOCAL.SLASH 0-dlist ],
      ARGS < [ SYNSEM [ LOCAL #slash & local &
                            [ CAT.VAL [ SUBJ olist,
                                        COMPS olist,
                                        SPR olist ],
                              CTXT.ACTIVATED + ],
                       NON-LOCAL.SLASH 0-dlist ] ],
             [ SYNSEM [ LOCAL.CAT [ VAL.COMPS olist ],
                        NON-LOCAL [ SLASH 1-dlist &
                                      [ LIST [ FIRST #slash,
                                               REST < > & #last ],
                                        LAST #last ],
                                    QUE 0-dlist,
                                    REL 0-dlist ] ] ] > ].
```

- basic-head-subj-phrase
- basic-head-spec-phrase
- basic-head-comp-phrase