HG7021 Computational Grammars

(De)composition in DELPH-IN MRS

Francis Bond
Division of Linguistics and Multilingual Studies
http://www3.ntu.edu.sg/home/fcbond/
bond@ieee.org

Lecture 7

Overview

- When do words and predicates get out of sync?
  - Semantically empty words
  - Constructions
  - Decomposed words
  - Idioms
- Boundary issues
  - Dealing with tokenizers
  - When to make decisions
- Grammar and grammar

Outline

Often, the mapping between predicate and word is not one-to-one

- Some words add no predicates:
  - auxiliary ***be***
  - infinitive ***to***
- Some constructions add predicates:
  - compound-rule
  - pumping rules (N$\rightarrow$ NP, NP$\rightarrow$ PP, AdjP$\rightarrow$ NP, . . .)

- Some words add multiple predicates:
  - ***here*** “this place” (“in this place”)
  - ***there*** “that place” (“in that place”)
  - ***where*** “which place” (“in which place”)
- Sometimes multiple predicates combine to form a special meaning
  - ***make note of*** “note”
  - ***play ball*** “cooperate”
  - ***behind schedule*** “late”
  - ***rack one’s brains*** “think hard”

Empty Predicates

- Our grammars treat some (very common) words as basically structural: they link other parts together, but add no predicate themselves.
- They pass the hook up and do little else
- To generate these
  - Add them all in every time (ineffecient)
  - Write trigger rules: add them in when needed

Constructions

Pumping Rules

There are some incomplete phrases, that act as though there is a missing element:

- *I want to go home* “to my house”
- *I put it here* “in this place”
- *I like gold* “some gold”
- *I like the rich* “the rich people”

We do this with pumping rules:

Head-Specifier Rule

$$\begin{bmatrix} \textit{phrase} \\ \text{VAL} \quad \begin{bmatrix} \text{COMPS} & \langle\rangle \\ \text{SPR} & \langle\rangle \end{bmatrix} \end{bmatrix} \rightarrow \boxed{2} \quad \mathbf{H} \begin{bmatrix} \text{VAL} \quad \begin{bmatrix} \text{COMPS} & \langle\rangle \\ \text{SPR} & \langle \boxed{2} \rangle \end{bmatrix} \end{bmatrix}$$

NP Pumping Rule: add the specifier

$$\left[\begin{array}{l} \textit{phrase} \\ \text{SYN}\left[\begin{array}{ll}\text{VAL} & \left[\begin{array}{ll}\text{COMPS} & \langle\rangle \\ \text{SPR} & \langle\rangle\end{array}\right]\end{array}\right] \\ \text{SEM}\left[\begin{array}{ll}\text{IND} & x \\ \text{RESTR} & \left\langle\left[\begin{array}{ll}\text{PRED} & \textit{udef\_q} \\ \text{IND} & x\end{array}\right] \oplus \boxed{A}\right\rangle\end{array}\right]\end{array}\right]$$

$$\rightarrow \mathbf{H}\left[\begin{array}{l}\text{SYN}\left[\begin{array}{ll}\text{CAT} & \textit{noun} \\ \text{VAL} & \left[\begin{array}{ll}\text{COMPS} & \langle\rangle \\ \text{SPR} & \langle\ \ldots\ \rangle\end{array}\right]\end{array}\right] \\ \text{SEM}\left[\text{RESTR}\left\langle\boxed{A}\right\rangle\right]\end{array}\right]$$

basic-bare-np-phrase

- The type in the matrix is *basic-bare-np-phrase*
- The predicate is added in C-CONT
- Iff the specifier is marked as OPT +

Decomposed Words

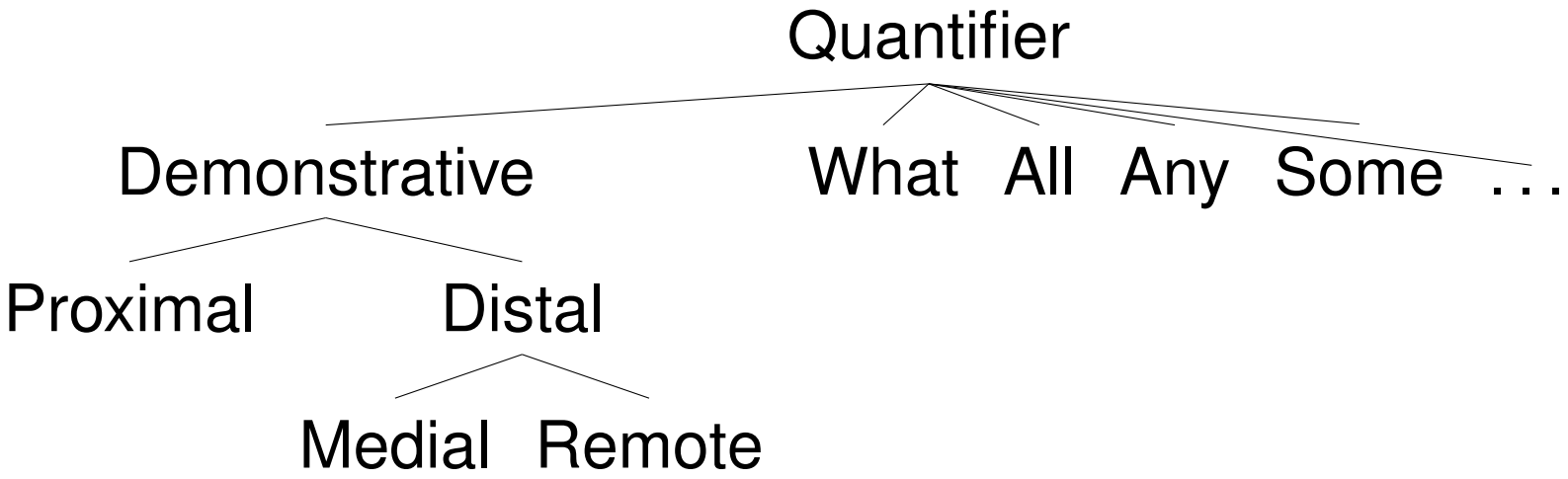
- Add two predicates for a single word
  - use LKEYS.KEYREL for the first
  - use LKEYS.ALTKEYREL for the second

Pronouns

- Many languages (all)? have demonstrative modifiers as well as pronouns
- We can model the pronouns as decomposed predicates

(1) *I like this ball*
(2) *I like this* "this thing"
(3) *I like kono tama*
(4) *I like kore* "kono mono"

Demonstrative Types

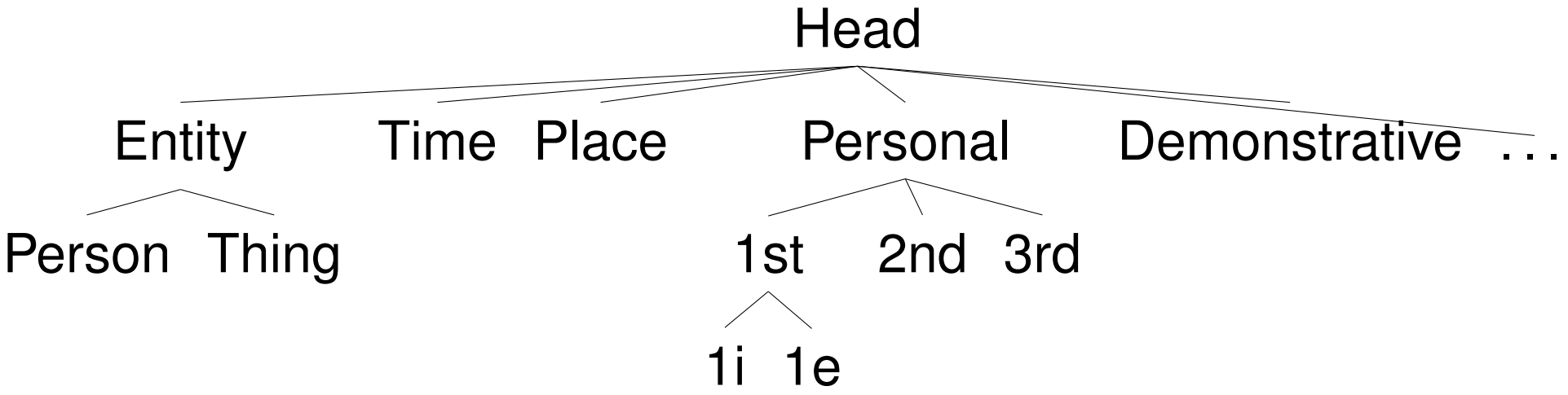


Universal Names

```
quant_q_rel := predsort.
demon_q_rel := quant_q_rel
proximal_q_rel := demon_q_rel.
dist_q_rel := demon_q_rel.
medial_q_rel := dist_q_rel.
remote_q_rel  := dist_q_rel.
which_q_rel  := quant_q_rel.
all_q_rel := quant_q_rel.
any_q_rel := quant_q_rel.
...
```

It is almost certainly more complicated than this.

Head Types



Do we really need Demonstrative?

Universal Names

```
generic_n_rel := predsort.
entity_n_rel := generic_n_rel
person_n_rel := entity_n_rel.
thing_n_rel := entity_n_rel.
time_n_rel  := generic_n_rel.
# where
place_n_rel  := generic_n_rel.
# why
reason_n_rel := generic_n_rel.
# how
manner_n_rel := generic_n_rel.
```

So how do we build them?

```
noun+det-lex-item := norm-hook-lex-item &
                     non-mod-lex-item &
[SYNSEM [LOCAL [CAT [ HEAD noun,
                      VAL [ SPR < >,    COMPS < >,
                            SUBJ < >,   SPEC < > ]],
                CONT [RELS <! relation &
                              [LBL #nh, ARG0 #s ],
                              quant-relation & #det &
                              [ARG0 #s, RSTR #h ]!>,
                      HCONS <! qeq & [ HARG #h,
                                       LARG #nh ] !> ]],
          LKEYS [ KEYREL relation,
                  ALTKEYREL #det ]]].


n+det-lex := noun+det-lex-item.
```

lexicon.tdl

```
kono := determinative-lex &
  [ STEM < "kono" >,
    SYNSEM.LKEYS.KEYREL.PRED "proximal_q_rel" ].

sono := determinative-lex &
  [ STEM < "sono" >,
    SYNSEM.LKEYS.KEYREL.PRED "medial_q_rel" ].

ano := determinative-lex &
  [ STEM < "ano" >,
    SYNSEM.LKEYS.KEYREL.PRED "remote_q_rel" ].
```

```
kore := n+det-lex &
  [ STEM < "kore" >,
    SYNSEM.LKEYS [KEYREL.PRED thing_n_rel,
                  ALTKEYREL.PRED proximal_q_rel ]].

sore := n+det-lex &
  [ STEM < "sore" >,
    SYNSEM.LKEYS [KEYREL.PRED thing_n_rel,
                  ALTKEYREL.PRED medial_q_rel ]].

mono := common-noun-lex  &
  [ STEM < "mono" >,
    SYNSEM.LKEYS.KEYREL.PRED thing_n_rel ].
```

Caveats

- Really, we should have a different predicate for the word *mono*

```
mono_n_rel := thing_n_rel.

mono := common-noun-lex  &
  [ STEM < "mono" >,
    SYNSEM.LKEYS.KEYREL.PRED mono_n_rel ].
```

  So we don't overgenerate: but for now let's!

- It's possible that dem_q_rel and so forth should be dem_a_rel, and we get the quantifier from somewhere else: can we say *this the man*?

Idioms

- Idioms can be flexible
- Match them in the semantics
- Look for at least one element marked [IDIOM +]
- [IDIOM +] consults with `idioms.mtr`
  - Each rule identifies an idiom
  - If the sentence has all the elements accept the sentence
    and mark the idiom?
  - Otherwise reject it

Check out the DELPH-IN Wiki: `http://moin.delph-in.net/JacyIdiom`