May 10, 2004

Optional arguments

General Q&A

Overview

- Optional arguments
- Analysis of optional arguments
- Circular lexical rules
- Spinning lexical rules
- Q&A

Optional arguments

- There are many cases in which an argument may be semantically present but syntactically absent.
- Semantically, these cases can be categorized by how the missing argument is interpreted.
- Syntactically, these cases can be categorized by how the missing argument is licensed.

Semantic classification

- Indefinite null instantiation: *I ate.*

  The referent of the missing argument is indefinite, not (necessarily) recoverable from context.

- Definite null instantiation: *I told you already.*

  The referent of the missing argument is definite, i.e., it should be recoverable from context.

- Constructional null instantiation: *Eat!*, *I told Kim to eat*

  The referent of the missing argument is determined by the syntactic construction.

Syntactic classification

- Lexical: The potential for an argument to be missing is determined by the lexical type/entry of the selecting head. E.g., *eat* allows indefinite null instantiation of its object, *devour* does not.
- Systematic: Arguments (perhaps of a certain syntactic type, such as NP or a particular grammatical function) in general can be missing. E.g., Japanese-style any argument pro-drop or Spanish-style subject pro-drop. By hypothesis, systematic pro-drop is given the definite interpretation (i.e., it corresponds to one use of over pronouns in other languages).

Lining up syntactic and semantic classifications

- Claim 1: A language with systematic pro-drop will allow definite interpretations of all dropped arguments.
- Claim 2: A language with systematic pro-drop will also allow indefinite interpretations of some dropped arguments, corresponding roughly to where a language without systematic pro-drop would allow indefinite null instantiation.
- Claim 3: No language allows indefinite null instantiation of subjects. [I rather expect this one to be falsified, cf. German impersonal passives.]
- Claim 4: It follows from these hypotheses that there is no need for lexically licensed definite null instantiation in languages with Japanese-style pro-drop.

Example (Japanese)

Tabeta

Ate

‘I/you/he... ate.’/‘I/you/he... ate it.’

- Japanese has systematic pro-drop of all arguments.
- It also appears to have lexically licensed INI.
- Thus *Tabeta* is ambiguous, and we would like to be able to translate it into two different English strings.
- Nonetheless, it would be nice to avoid assigning two different tree structures, and rather provide an underspecified semantic representation.

Proposed analysis in the Matrix: Overview (1/2)

- Constructional null instantiation covered by analysis of imperatives, raising, etc.
- Distinction between definite and indefinite null instantiation handled by a feature on indices [DEF bool].
  - Pronouns, arguments subject to DNI (and possibly definite NPs) are INDEX.DEF +.
  - Arguments subject to INI (and possibly indefinite NPs) are INDEX.DEF -.

Proposed analysis in the Matrix: Overview (2/2)

- Posit opt-comp and opt-subj rules parallel to the covert-det rules.
- Use a feature [OPT bool] to code lexically licensed null instantiation (leaving it underspecified in languages where there is systematic pro-drop).
- Use a second feature [OPT-DEF bool] to allow lexical items to specify whether any given optional argument would be interpreted as definite or indefinite in case of null instantiation. (As a stand-in for a semantic-interface based approach.)

The feature OPT

- OPT and DEF-OPT will both be features of *synsem*s.
- However, nothing constrains its own OPT value (that is, no phrases are inherently optional or non-optional, independent of which head they are dependent on).
- Rather, heads constrain certain arguments to be [OPT −], which blocks the optional complement/subject rules from applying, since these look for argument which are (compatible with) [OPT +].

The feature DEF-OPT

- DEF-OPT is just a ‘junk slot’ to allow a lexical head to store information about how an argument will be interpreted if it is unexpressed.
- The opt-comp rule will identify the DEF-OPT and HOOK.INDEX.DEF values of any argument it caches out as unrealized.
- Because the HOOK.INDEX of every argument is identified with some ARGn position in the head’s key relation, this information will be encoded in the semantics.

An opt-comp rule

```
basic-head-opt-comp-phrase := head-valence-phrase & basic-unary-phrase &
                              head-compositional &
  [ SYNSEM canonical-synsem &
         [ LOCAL.CAT [ VAL [ SUBJ #subj,
                             SPR #spr,
                             COMPS #comps,
                             SPEC #spec ]]],
    HEAD-DTR #head &
        [ SYNSEM [ LOCAL.CAT [ VAL [ SUBJ #subj,
                                     SPR #spr,
                                     COMPS < unexpressed &
                                             [ OPT +,
                                               DEF-OPT #def,
                                               ..INDEX.DEF #def ] .
                                            #comps >,
                                     SPEC #spec ]]]],
    C-CONT [ RELS <! !>,
             HCONS <! !> ],
    ARGS < #head > ].
```

For a language with systematic pro-drop

- Allow definite null instantiation (pro-drop) everywhere.
- Also allow indefinite null instantiation if lexically specified.
- Two rules:
  - One head-opt-comp rule doesn't look at DEF-OPT value and just puts in DEF +.
  - The other requires [DEF-OPT −] and copies that to DEF.
- Lexical items specify [DEF-OPT +] if they don't allow indefinite null instantiation (of that argument).

For Lab 5 (1/2)

- Determine whether your language allows systematic pro-drop, and if so, under what conditions (subjects only, all arguments, nearly all arguments, complements of verbs but not of adpositions, ...)
- Determine whether your language allows indefinite null instantiation for the objects of any verbs in your lexicon (*eat* would be a good guess).

For Lab 5 (2/2)

- If your language doesn’t allow pro-drop everywhere, determine whether it nonetheless allows lexically licensed definite null instantiation.
- Try to find out whether your language allows indefinite null instantiation of subjects (whether or not it’s a pro-drop language). Good places to look are translations of *There was dancing at the party*, and similar.

Circular lexical rules (1/2)

- Lexical rules which have compatible input and output (basically, SYNSEM and DTR.SYNSEM, although INFLECTED and DTR.INFLECTED a couple of other things are also compared).
- A problem for generation, as the generator can’t know when to stop applying it.
- The generator is smart, however, and recognizes the situation, giving the error “probable circular rule”.

Circular lexical rules (2/2)

- Never a problem with ltow rules.
- Frequently a problem with ltol rules, and in particular that are just monotonically adding SYNSEM information.
- Fix this by making the mother and daughter incompatible, perhaps with additional features, perhaps with the type of the DTR value.

Spinning lexical rules

- For some reason, orthography subrules of the form `%affix (!lfoo !l)` cause the LKB to spin.
- In this case, it's the morphology component which is spinning, while trying to strip the affixes.
- (If you'd like to see this in action, try typing *morph-trees* at the LKB prompt in the *common-lisp* buffer, while the LKB is spinning.)
- This bug has been reported, but we don't have a work around. Sorry :-(

Overview

- Optional arguments
- Analysis of optional arguments
- Circular lexical rules
- Spinning lexical rules
- Q&A