Today Lecture 2, Part II: Grammar Engineering

Prolog Programs:

`experiment1.pl`
`experiment2.pl`
`experiment3.pl`
`lambda.pl`
`betaConversion.pl`
`betaConversionTestSuite.pl`
`alphaConversion.pl`
`englishLexicon.pl`
`englishGrammar.pl`
`sentenceTestSuite.pl`

Grammar Engineering

- The explicit notation for functional application and the implementation of β-conversion are the basic tools we shall work with in this course

- So it is time to define a bigger grammar and start exploring computational semantics

- But let's try to observe some basic principles of grammar engineering as we do so!

- we should strive for a grammar that is:
    - *modular*
    - *extendible*
    - *reusable*

Four-Level Grammar Architecture

- (1) The Syntax Rules (Prolog DCGs)
- (2) The Lexicon (a dictionary of words)
- (3) The Semantic Rules (mirroring the syntax rules)
- (4) The Semantic Macros (semantic representations for the lexical items)

The **Syntax Rules** and **Lexicon** stay fixed in the course.

The **Semantic Rules** and **Semantic Macros** are the levels that involve modifications as we do most of our semantic work here.

(1) The Syntax Rules (DCGs)

```
t --> s.
s --> np, vp.

np --> np, coord, np.
np --> det, n.
np --> pn.

n --> n, coord, n.
n --> adj, n.
n --> noun.
n --> noun, nmod.

rc --> relpro, vp.
```

```
vp --> vp, coord, vp.
vp --> av, vp.
vp --> cop, np.
vp --> iv.
vp --> tv, np.

pp --> prep, np.

nmod --> pp.
nmod --> rc.
nmod --> pp, nmod.
```

Adding features for syntactic constraints (1/2)

```
s([])-->
   np([num:Num]),
   vp([inf:fin,num:Num]).
```

```
vp([inf:I,num:Num])-->
   tv([inf:I,num:Num]),
   np([num:_]).
```

Number agreement (`num`) between NP and VP (values: `sg` and `pl`)

Examples (∗ indicates ungrammatical sentences):

Mia smokes
Mia and Vincent smoke
∗ Mia smoke
∗ Mia and Vincent smokes
Mia or Vincent smokes

Adding syntactic constraints (2/2)

```
s([])-->
   np([num:Num]),
   vp([inf:fin,num:Num]).
```

```
vp([inf:I,num:Num])-->
   tv([inf:I,num:Num]),
   np([num:_]).
```

Inflection of verbs (`inf`): (values: `inf`, `fin`)

Examples:

Mia smokes
Mia does not smoke
∗ Mia does not smokes

Adding semantic annotations

The Syntax Rules have a placeholder for semantic information:

```
s([sem:S])-->
   np([num:Num,sem:NP]),
   vp([inf:fin,num:Num,sem:VP]),
   {combine(s:S,[np:NP,vp:VP])}.

vp([inf:I,num:Num,sem:VP])-->
   tv([inf:I,num:Num,sem:TV]),
   np([num:_,sem:NP]),
   {combine(vp:VP,[tv:TV,np:NP])}.
```

How the semantic information is passed upwards the tree is specified by the semantic rules (instances of `combine/2`)

Lexical rules

Lexical rules apply to terminal symbols (the actual strings in the input of the parser) and need to call the lexicon to check if a string belongs to the syntactic category searched for.

```
noun([sem:Sem])-->
   {lexEntry(noun,[symbol:Sym,syntax:Words,_])},
   Words,
   {semMacro(noun,[symbol:Sym,sem:Sem])}.
```

Each lexical category is associated with such a macro, and this enables us to abstract away from specific types of structures. So we have set up the syntax rules in such a way that we are independent from the semantic theory we want to work with!

(2) The Lexicon

The general format of a lexical entry is `lexEntry(Cat,Features)` where

- `Cat` is the syntactic category
- `Features` is a list of feature-value pairs

For example, the entries for the intransitive verb ‘to walk’ are:

```
lexEntry(iv,[symbol:walk,syntax:[walk],inf:inf,num:sg]).
lexEntry(iv,[symbol:walk,syntax:[walks],inf:fin,num:sg]).
lexEntry(iv,[symbol:walk,syntax:[walk],inf:fin,num:pl]).
```

(3) The Semantic Rules

The required semantic annotations for our implementation of the lambda calculus are utterly straightforward; they are simply the obvious "apply the function to the argument statements" expressed with the help of app:

```
combine(s:app(A,B),[np:A,vp:B]).
combine(n:app(app(B,A),C),[n:A,coord:B,n:C]).
combine(np:A,[pn:A]).
```

Of course, because `combine/3` offers us extreme flexibility in implementing semantic construction, we can also choose to apply β-conversion directly:

```
combine(s:Converted,[np:A,vp:B]):-
   betaConvert(app(A,B),Converted).
```

(4) The Semantic Macros

The semantic macros specify the lexical semantics

```
semMacro(noun,M):-
   M = [symbol:Sym,
        sem:lam(X,Formula)],
   compose(Formula,Sym,[X]).

semMacro(tv,M):-
   M = [symbol:Sym,
        sem:lam(K,lam(Y,app(K,lam(X,Formula))))],
   compose(Formula,Sym,[Y,X]).
```

The macros for the determiners

```
semMacro(det,M):-
   M = [type:uni,
        sem:lam(P,lam(Q,all(X,imp(app(P,X),app(Q,X)))))].

semMacro(det,M):-
   M = [type:indef,
        sem:lam(P,lam(Q,some(X,and(app(P,X),app(Q,X)))))].
```

How to run the code yourself (at NASSLLI)

1. `ssh steel.ucs.indiana.edu` (see instructions in welcome package)

2. include `~achagrov/shared/bin` in the path (again, see instructions)

3. `dce_login`

4. You'll find the programs in `achagrov/shared/comsem/`
   You can either run the programs there or copy them to your own account and run them there.

5. For instance, to run the `lambda.pl`, you'll need to type:
   - (a) `pl` (start Prolog)
   - (b) `[lambda].` (load the program lambda.pl)
   - (c) `lambda.` (start parsing a sentence)

Tomorrow's Lecture

- Scope Ambiguities (recall "Every boxer loves a woman")
- Several Methods for Dealing with Scope Ambiguities
  - Storage Methods
  - Underspecification
- Integration of scope ambiguities in the grammar we have