Assignment 2 — Feature grammars

CSC485/2501 – Fall 2018

Computer Science
UNIVERSITY OF TORONTO

Slides are based on ALE user's guide by Carpenter & Penn
`http://www.ale.cs.toronto.edu/docs/man/ale_trale_manual.pdf`

Running TRALE

- Use tcsh.
- Remove the default virtual memory limit:

  ```
  limit vmemoryuse unlimited
  ```

- Starting **TRALE** (ssh with the -X flag):

  ```
  path to trale/trale -fsg
  ```

Getting started (1)

- Load a grammar:

  ```
  ?- compile_gram(grammar).
  ```

- Check lexical entries:

  ```
  ?- lex(puppies).
  ```

- Check rules:

  ```
  ?- rule(srule).
  ```

- Executing a grammar – input for parsing:

  ```
  ?- rec[john,walks].
  ```

  Note:

  - Input string as a list.
  - No variables in the query.

Getting started (2)

- If there are no parses, ’no’ is returned:

  ```
  ?- rec[walks,john].
  STRING:
  0 walks 1 john 2
  no
  ```

- Other lexical entries, rules, parses:

  ```
  ANOTHER? y.
  ```

- The fullstop is necessary with all commands.

Types

- Every feature structure has a type.
- Types have subtypes (more specific instances of the type).
- Type names must be lower-cased.
- The most general type is `bot`.
- Everything is a sub-type of `bot`.
- A simple type specification: the name of the type, followed by the keyword `sub`, followed by a list of its subtypes, e.g.,

```
bot sub [b,c].
   b sub [d,e].
     d sub [].
     e sub [].
   c sub [].
```

Feature structure

- A collection of feature/value pairs.
- E.g., type vp (with no subtypes) has a feature subj with value np:

```
vp sub []
  intro [subj:np].
```

Type system

- Appropriateness: each type must specify:
  - which features it can be defined for,
  - which type of values such features can take.

Lexicon

- Adding lexicon:

```
john −−−> np.
walked −−−> vp.
```

Grammar rules

- Name of the rule: `srule`.
- Atom `rule` specifies type of info for ALE compiler.
- Nonterminal of the mother: `s` (nonterminals are lower-cased).
- Daughter categories indicated by `cat>`.
- Order is important.
- Add comments (using ’%’) to explain what your grammar does.

```
% Grammmar Rule allowing the combination of
% np category with a vp type category

srule rule
s
===>
cat> np,
cat> vp.
```

Simple grammar

```
% Type Hierarchy
bot sub [s,np,vp].  % Three sub-types of bot
s sub [].  % Each with no sub-types
np sub [].
vp sub [].
% Lexical Entries
john ---> np.
walked ---> vp.
% Grammmar Rules
srule rule
s
===>
cat> np,
cat> vp.
```

Simple grammar

```
bot sub [pp,p,np].
pp sub [].
p sub [].
np sub [].
with ---> p.
sam ---> np.
srule rule
pp
===>
cat> p,
cat> np.
```

Variables

- Start with upper-case letters.
- Variables with the same name must unify.

```
cat sub [s,np,vp].
np sub []
   intro [index:index].
vp sub []
   intro [subj:np].
s sub [].
srule rule
s
===>
cat> (np,index:Ind),
cat> (vp,subj:index:Ind).
```

INDEX feature

- Takes the type `index` as its value.
- Contains agreement features, gender, number, and person.

```
index sub []
    intro [p:person,num:number,g:gend].
person sub [first,second,third].
number sub [sing,plural].
gend sub [m,f,n].
```

Exercise (1)

- Modify this grammar in a way that parses *I walk.*

  ```
  bot sub [s,np,vp].
  s sub [].
  np sub [].
  vp sub [].
  john ---> np.
  walks ---> vp.
  srule rule
  s
  ===>
  cat> np,
  cat> vp.
  ```

- Does it parse *I walks?* or *John walk?* How can you avoid it?

Exercise (2)

- Modify the previous grammar in a way that parses *John walks with Sam*.