Towards more complex linguistic data structures

- atoms, e.g. `verb_ditrans_first_sing_fin`
- compound terms, e.g. `verb(first,sing,fin,[np,np])`
- feature structures, e.g.

```
category: verb,
vform: fin,
person: first,
number: sing,
subcat: [np, np]
```

- typed feature structures, e.g.

```
category: (verb,
           vform: (fin,
                   person: nfirst,
                   number: sing),
           subcat: [np, np])
```

Typed feature logic
(King, 1989, 1994; Carpenter, 1992)

The linguistic ontology is defined in the *signature*.

- Type hierarchy: which type of objects exist.
- Appropriateness Conditions: which objects have which properties.

Using a formal description language, one can make statements about

- objects and
- the value of attributes of objects

with respect to the

- type of an object, and the
- token identity of two objects (path equality)

These atomic formulae are combined to more complex ones using

- conjunction,
- disjunction, and
- negation.

A typed-feature based parsing system

- The Attribute-Logic Engine (ALE) is a freeware logic programming and grammar parsing and generation system developed by Bob Carpenter and Gerald Penn:
  http://www.sfs.nphil.uni-tuebingen.de/~gpenn/ale.html
- A basic ALE grammar consists of
  - signature
  - theory
    - lexical entries
    - phrase structure rules

A first example grammar (grammar0.pl)
signature

```
bot sub [s,np,vp].

s sub [].
np sub [].
vp sub [].
```

A first example grammar (grammar0.pl)
theory

```
% --- lexical entries ----------
sandy ---> np.
snored ---> vp.

% --- phrase structure rule ----
np_vp rule
s
===>
cat> np,
cat> vp.
```

Starting ALE

1. To set things up, add to your file `.sicstusrc` the line:

   ```
   ale :- compile(’~dm/.local/lib/ale/ale.pl’).
   ```

2. To start ale, type the following
   (a) at unix prompt: `xemacs &`
   (b) in xemacs: `M-x run-prolog`
   (c) at prolog prompt: `ale.`

Compiling a grammar in ALE

1. Take or write a grammar, e.g.: `grammar0.pl`
2. Start ALE
3. Compile the grammar in ALE: `compile_gram(grammar0).`

   Note: Every time you change something in the grammar, you need to recompile!

Inspecting a compiled grammar in ALE

- Lexical entries: `lex(sandy).` or `lex(X).`
- Phrase structure rules: `rule(np_vp).` or `rule(Y).`

At the question `ANOTHER?` type

- `y.` to proceed and
- `n.` to not check for further solutions.

Recognizing a string after compiling a grammar

```
rec [sandy, snored].

STRING:
0 sandy 1 snored 2

CATEGORY:

s

ANOTHER?
```

Grammar 1

```
bot sub [sign,cat,head].

sign sub []
   intro [cat:cat].

cat sub []
   intro [head:head,
          subj:head].

head sub [noun,verb].
noun sub [].
verb sub [].
```

```
% Lexical Entries
sandy ---> cat:head:noun.

snored ---> cat:(head:verb,
                 subj:noun).

% Grammmar Rules
subject_head rule
(cat:head:verb)
===>
cat> (cat:head:Head),
cat> (cat:subj:Head).
```

Grammar 2

```
bot sub [sign,list,cat,head].

sign sub []
   intro [cat:cat].

cat sub []
   intro [head:head,
          subj:list].

head sub [noun,verb].
noun sub [].
verb sub [].
```

```
list sub [e_list,ne_list].

ne_list sub []
  intro [hd:sign,
         tl:list].

e_list sub [].
```

```
% Lexical Entries

sandy  ---> cat:(head:noun, subj:[]).
snored ---> cat:(head:verb, subj:[cat:head:noun]).

% Grammmar Rules

subject_head rule
(cat:head:verb)
===>
cat> (cat:head:Head),
cat> (cat:subj:[cat:head:Head]).
```

Grammar 3

```
% New comps attribute
cat sub []
   intro [head:head,
          subj:list,
          comps:list].

% Macro
np macro
  cat:(head:noun,
       subj:[],
       comps:[]).
```

```
% Lexical Entries

sandy ---> @np.
kim   ---> @np.
books ---> @np.

snored ---> cat:(head:verb, subj:[@np], comps:[]).
likes ---> cat:(head:verb, subj:[@np], comps:[@np]).
gives ---> cat:(head:verb, subj:[@np], comps:[@np,@np])
```

```
% Grammmar Rules
subject_head rule
(cat:subj:[])
===>
cat> (cat:head:Head),
cat> (cat:(subj:[cat:head:Head], comps:[])).

head_complement rule
(cat:comps:[])
===>
cat> (cat:comps:Comps),
cats> (Comps,ne_list).
```

References

Carpenter, Bob (1992). *The Logic of Typed Feature Structures – With Applications to Unification Grammars, Logic Programs and Constraint Resolution*, Volume 32 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge, UK: Cambridge University Press.

King, Paul John (1989). *A Logical Formalism for Head-Driven Phrase Structure Grammar*. Ph. D. thesis, University of Manchester, Manchester.

King, Paul John (1994). *An Expanded Logical Formalism for Head-driven Phrase Structure Grammar*. Arbeitspapiere des SFB 340 Nr. 59. Tübingen: Universität Tübingen.