Implementing grammars in Trale: Parsing with typed-feature structures

Detmar Meurers: Intro to Computational Linguistics I
OSU, LING 684.01, 3. March 2003

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.

Typed-feature based phrase structure grammars in Trale

signature

```
type_hierarchy
bot
  type feature1:value1 feature2:value2
    subtype1
    subtype2
.
```

theory.pl

```
string ---> preterminal.

rulename rule mother ===>
cat> daughter1
cat> daughter2.
```

A first example grammar
(trale/0)

signature

```
type_hierarchy
bot
  s
  np
  vp
.
```

theory.pl

```
she    ---> np.
smiles ---> vp.

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

A grammar with more interesting feature structures
(trale/1/signature)

```
type_hierarchy
bot
  sign pos:part_of_speech subcat:list
     word
     phrase
  part_of_speech
     verb
     noun
  list
     ne_list hd:bot tl:list
     e_list
.
```

A grammar with more interesting feature structures
(trale/1/theory.pl)

```
she    ---> (word, pos:noun, subcat:[]).
smiles ---> (word, pos:verb, subcat:[(pos:noun,subcat:[])]).

basic_sentence rule
(phrase, pos:H, subcat:[])
===>
cat> Subj,
cat> (pos:(verb,H),subcat:[Subj]).
```

A grammar with abbreviations
(trale/2/theory.pl)

```
np := (pos:noun, subcat:[]).

she    ---> (word, @np).
smiles ---> (word, pos:verb, subcat:[@np]).

basic_sentence rule
(phrase, pos:H, subcat:[])
===>
cat> Subj,
cat> (pos:(verb,H),subcat:[Subj]).
```

Getting started

- A basic grammar consists of two files: `theory.pl` and `signature`
- To start Trale, `cd` to the directory with your grammar, then type `trale&`
- Compile a grammar using: `compile_gram(theory).` or short `c.`

  Every time you change something in the grammar, you need to recompile!
- Check out the TRALE manual (Part 1: ALE, Part 2: TRALE), available from the course web page.

Inspecting a compiled grammar

- Lexical entries: `lex/1`, e.g.:
  - `lex(she).`
  - `lex(X).`
  - `...`
- Phrase structure rules: `rule/1`, e.g.:
  - `rule(basic_sentence).`
  - `rule(Y).`
  - `...`
- Testing descriptions for well-formedness: `mgsat/1`
- Switch on tracing mode of parser: `interp/0`

Recognizing a string

After compiling a grammar, strings are recognized using the predicate `rec/1`, e.g.:

```
| ?- rec([she,smiles]).

STRING:
0 she 1 smiles 2

CATEGORY:
s

Do you want to try for more solutions? (return for <yes>)

no
| ?-
```

References

Carpenter, Bob (1992). *The Logic of Typed Feature Structures – With Applications to Unification Grammars, Logic Programs and Constraint Resolution*, vol. 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*. No. 59 in Arbeitspapiere des SFB 340. Tübingen: Universität Tübingen. `http://www.sfs.uni-tuebingen.de/sfb/reports/berichte/59/59abs.html`.