The Linguistic Knowledge Builder (LKB)

Fabiola Henri

Department of English/Linguistics Program
University of Kentucky
fabiola.henri@uky.edu

LINCD Session 3
February 2014

Introduction

The Linguistic Knowledge Builder (LKB) is a specialized grammar engineering environment for contraint-based grammars

- ☞ Specifically designed for typed feature structures (TFS)

Originally developed by Ann Copestake and later by John Carroll, Rob Malouf, and Stefan Oepen

Requires little knowledge of computers

Although LKB is an open-source that runs on different operating systems, the use of Linux is recommended

Introduction

1. Extensive efficiency improvements, so the system is capable of parsing reasonable length sentences with a large grammar.
2. Default unification is based on YADU, defined in Lascarides and Copestake (1999).
3. Automatic computation of greatest lower bounds in the type hierarchy.
4. Integration with the [incr tsdb()]6 test suite machinery (Oepen and Flickinger, 1998).
5. Integration with MRS semantics (Copestake et al, 1999).
6. Tactical generation from MRS input (relatively experimental).
7. Many new user interface features.

A word on Grammar Engineering

Natural language grammars are implemented in many softwares

Used for both parsing and generation

Precision grammar: requires fully explicit analyses (vs general approaches to syntax)

Applications

Language documentation

Linguistic hypothesis testing

Computer assisted language learning

Machine translation

. . .

Goals of LINCD Session

Familiarize ourselves with the LKB platform

Investigate the implementation of constraints in morphology, syntax, and semantics within HPSG

Outline

LKB grammar files

Types and constraints (*types.tdl*)

Lexical entries (*lexicon.tdl*)

Grammar rules (*rules.tdl*)

Lexical and morphological rules (*lrules.tdl* and *irules.tdl*)

Auxiliary settings

- *script* which loads various files in the grammar
- *globals.lsp* which contains global settings

A first session

Open a web-serv session via Putty (Advanced Syntax folder)

☞ Xming should be loaded before hand

You should find in your directory a lkb-data folder

Open emacs

Run LKB: `M-x lkb RET`

An Xming window (LKB Top) should pop up! **YAY!**

A first session

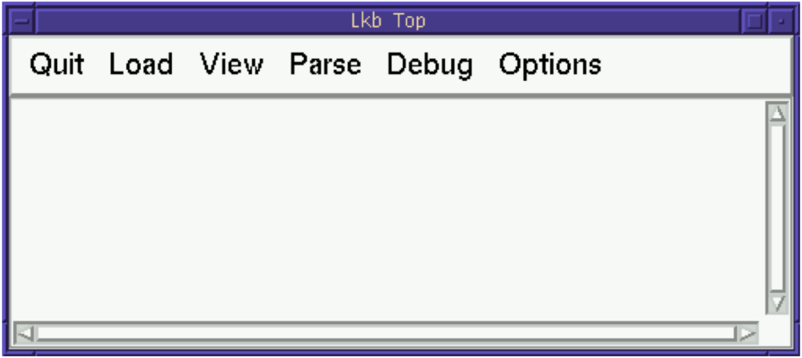


Figure: The LKB interaction window or LKB top menu

Loading a grammar

The LKB comes with a series of grammars

- ☞ a set of files containing types and constraints, lexical entries, grammar rules, etc

The sample grammar: `lkb-data/itfs/g8gap`

Select `Complete grammar` from the LKB `Load` menu, and choose the `script` file from `g8gap` the directory

Loading a grammar

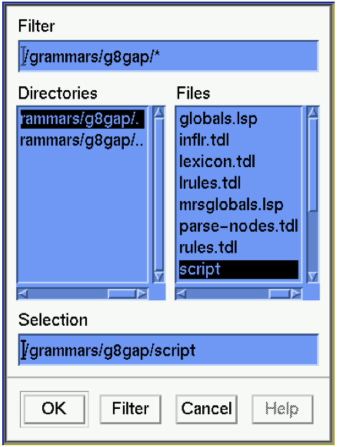


Figure: Selecting the script file

Loading a grammar

Once a file is successfully loaded, the menu commands are all available and a type hierarchy window is displayed

Loading a grammar

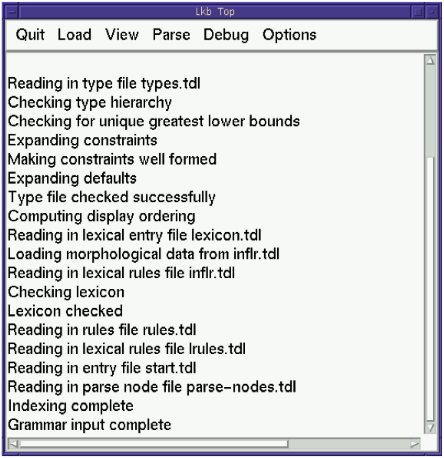


Figure: Loading a grammar

Loading a grammar

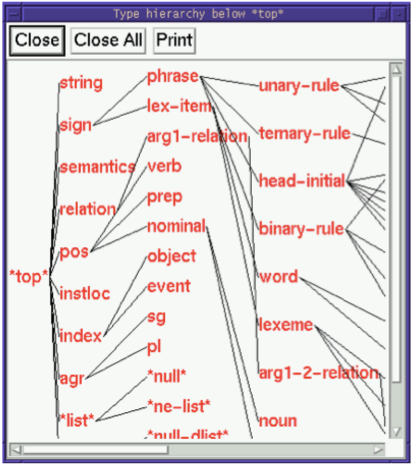


Figure: Type hierarchy window

Parsing

Select “Parse | Parse input ...” from the LKB Top menu and parse the sentence that appears in the dialogue

Outline

A tour of the LKB system

TFS

An indepth examination

Extending the grammar

The Matrix

Readings

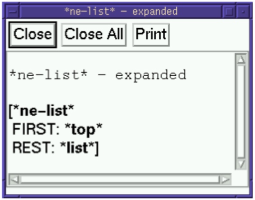
Feature structures and type constraints

The LKB grammar is a type system with constraints expressed as feature structures

- ☞ The most general type displayed at the left of the window `*top*`
- ☞ Some types has more than one parent (*multiple inheritance*)

To inspect the constraint on the type `*ne-list*` (non-empty list), select `Expanded Type` from the menu

- ☞ It has two features, FIRST and REST
- ☞ The value of FIRST is `*top*`: it can unify with any feature structure
- ☞ The value of REST is `*list*`: it can only unify with the type `*list*` or one of its subtypes



Feature structures and type constraints

The entry for the type `*ne-list*` is found in the source file `g8gap/types.tdl`

☞ Open the file in emacs

```
ne-list := *list* &
 [ FIRST *top*,
   REST *list* ].
```

Feature structures and type constraints

The syntax of the language in which the type and its constraint are defined is called *The Description Language*

The type definition obligatorily specifies the parent or parents of a type and optionally defines a constraint

- ☞ Here `*list*` (the parent) does not have any feature in its constraint
- ☞ But type constraints can inherit a lot of information from the parent, allowing for a compact description

Inspect a more complicated type constraint like `phrase`

Feature structures and type constraints

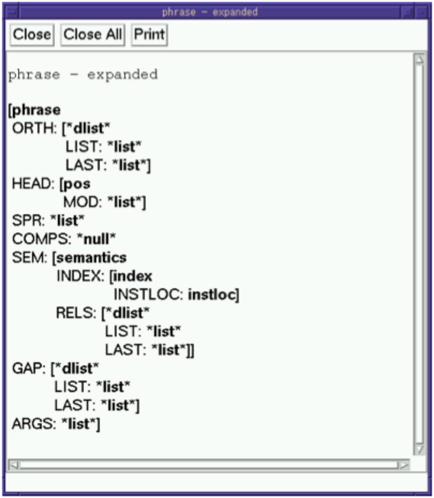


Figure: TFS window for *phrase*

Feature structures and type constraints

The value of a feature in a constraint can be a TFS

```
phrase := sign &
  [ COMPS < > ].
```

The notation $< >$ is a shorthand for `*null*`

The view command

The view commands allows you to look at entities such as lexical entries

Select `View` from the LKB top menu and then select `Word entries`

Enter *dog* when prompted for a word (any entry from `lexicon.tdl`); OK

Identifiers can be added to provide unique lexical entries in case of multiple entries with identical spelling (`dog_1`)

The PHON feature in HPSG = ORTH in LKB

`dlist` = allows for concatenation of lists

The view command

Grammar rules can also be viewed by clicking on `View Grammar rule`

Try for instance `head-specifier-rule`

It is possible to `shrink/expand` to view parts of the structure

☞ The mother is the TFS as a whole while the daughters are the `list` value of the feature ARGS

Outline

Parsing sentences

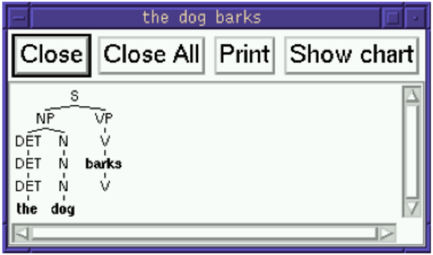


Figure: Parsing trees

Clicking on the tree provides a menu with the option `Show enlarged tree`

Click on the top node and choose `Feature structure - Edge 11`

The printed TFS is an instantiation of the `head-specifier-rule`

Parsing sentences

The top node = root node

The structure for the NP *the dog* is the value of the path ARGS.FIRST

The structure for the verb *barks* is the value of the path ARGS.FIRST.REST

More later on how the grammar rules works cf. `parse-nodes.tdl`

Morphological and lexical rules

The tree has 2 V nodes (above and below *barks*)

This is an application of a morphological rule

- ☞ Morphological rules are used for inflectional and derivational processes with affixation
- ☞ Lexical rules for processes with no affixation

In `lexicon.tdl` > the entry for *bark*

In `inflr.tdl`> the rule for 3SG generates the inflected form *barks*

Morphological and lexical rules

`View > lex entry` for *bark*

You will be prompted for `Lex-id>` *bark* > `bark - expanded`

Then choose `Apply all lex rule`

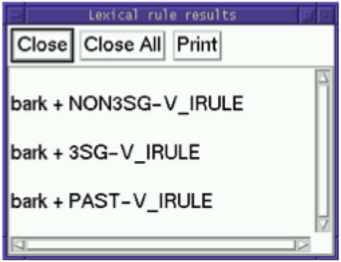


Figure: Lexical rules

Clicking on the nodes will display the feature structures corresponding to the nodes

Batch Parse

Try `Parse > Batch Parse`

Choose `test.items` when prompted to choose a file

The test suite file is basically the coverage of the grammar

Enter a name, for e.g `test.results` as the output file; the system will parse all the sentences in the file

Open your `test.results` file in emacs

Batch Parse

The data in your `test.results` file will show:

1. The sentence itself
2. The number of the sentence
3. The number of parses
4. the number of passive edges (a passive edge is a phrase that the system constructs while attemting to parse a sentence)

In the test suite file, ungrammatical sentences are marked with an asterisk but this is stripped out by the system when it is parsed

- Total parsing time is also reported for all sentences

Semantic Representation

Parse *The dog chased the cat*

Choose `Indexed MRS` > Semantic representation of the sentence

MRS (Minimal Recursion Semantics) is a semantic representation language that can be converted into for eg. predicate calculus

Figure: MRS representation

Semantic Representation

The semantic is in fact also a TFS; it is the value of SEMANTICS in the parsed sentence

The option `Indexed MRS` more readable

The option `MRS` has a representation closer to a TFS

Other options: `Prolog MRS` $>$ Suitable for Prolog systems

Other options: `Scoped MRS` $>$ Requires a full representation of quantifiers

Semantic Representation

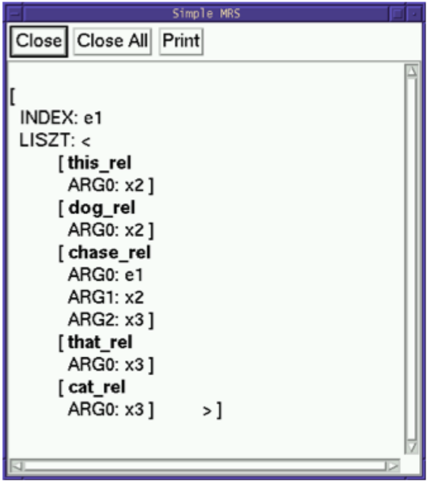


Figure: TFS MRS representation

Generating Sentences

The LKB allows generation of grammatical sentences

Click and choose the option `Generate` on the tree

It is also possible to generate form the MRS representation

- ☞ Input version of the sentences and other inflected forms of that same sentence

Try to parse an ambiguous sentence

- ☞ *The dog chased the cat near the aardvark*
- ☞ The grammar can sometimes generate ungrammatical sentences (*overgeneration*)

Outline

Adding a lexical entry

Open the file `lexicon.tdl`

Suppose you want to add another noun: *squirrel*

Change the orthography and value of the semantic feature

the "" around the values specify that these are proper values

Save the file, and then select `Load` followed by `Reload grammar`

Try to parse *The cat chased the squirrel*

Adding a type with a constraint description

Consider pair nouns like *scissors*, *binoculars*, *pants*, *trousers*

They always show plural agreement

Precision grammar: Attributing them the type `noun-lxm` would predict that they would have both singular and plural forms

Make a new type which says that the number agreement is always plural

Adding a type with a constraint description

Open `types.tdl` > look for the type description `noun-lxm`

Add a new type `pair-noun-lxm` that would inherit from `noun-lxm` that specifies a vlaue `pl` for the feature HEAD.NUMAGR

```
pair-noun-lxm := noun-lxm &
 [ HEAD [ NUMAGR pl ]].
```

Save and reload grammar > Check the type hierarchy!

Add a new entry to `lexicon.tdl`

```
scissor := pair-noun-lxm &
[ ORTH.LIST.FIRST "scissor",
  SEM.RELS.LIST.FIRST.PRED "scissor_rel" ].
```

try to parse new sentences and generate new ones!

Outline

A tour of the LKB system

TFS

An indepth examination

Extending the grammar

The Matrix

Readings

Grammar Customization

The LinGO Grammar Matrix aloow you to build up an implemented grammar for a language of your choice

Developed by Emily Bender et al. at the University of Washington

Provides a starter grammar with a language-independent core and customized support (your input)

http://www.delph-in.net/matrix/customize/matrix.cgi

Grammar Customization: tdl files

`matrix.tdl`: Supertypes for lex-rules, which handle the copying up of everything youre not changing

`my_language.tdl`: Position classes and lex rule types defined through the customization system; features for inside INFLECTED

`lrules.tdl`: Instances for non-spelling-changing lex rules (zero morphemes)

`irules.tdl`: Instances for spelling-changing lex rules

Copestake, A. (2002). *Implementing Typed Feature Structure Grammars*. CSLI lecture notes. C S L I Publications/Center for the Study of Language & Information.

Pollard, C. and Sag, I. (1994). *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. University of Chicago Press.

Sag, I. A., Wasow, T., and Bender, E. (2003). *Syntactic Theory: A Formal Introduction*. Stanford: Center for the Study of Language and Information, 2nd edn.