The (new) LKB system

nn Copesta e

J

November 5, 1998

This document describes the new version of the LKB system: a grammar and lexicon development environment for use with constraint-based linguistic formalisms.

Contents

Chapter 1

Introduction

1.1 A brief introduction to the LKB system

The LKB system is a grammar and lexicon development environment for use with constraint-based formalisms.[1]

knowledge of some NLP concepts, such as chart parsing, will be helpful, but this should not

1.4 Acknowledgments

run in ‘tty’ mode from other Common Lisps. tty mode is also useful if you are accessing a

This will create the new directory and various subdirectories. It may take a few minutes

3. Once the loadup les have been loaded, enter the following into the Listener window:

This compiles and loads all the LKB les that are appropriate for your system.[3] You should now see LKB on the menu-bar.

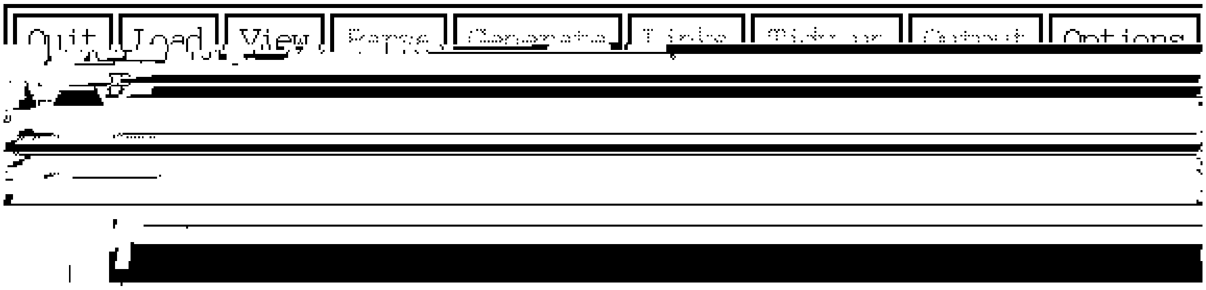

Figure 2.1: The LKB interaction window in ACL/CLIM

Process directory:
Executable image name:
Lisp image:
Image arguments:

You should then get a new buffer called with a running Lisp process.

4. At the Lisp prompt enter:

2.4.3 MCL

Chapter 3

A rst session

LKB menu to that. LKB error messages etc appear in the MCL Listener window. For the ACL/CLIM version, there is a distinct top level interaction window, with the menu displayed using buttons across the top of the window. From now on, we will ignore these di erences. We will use the term ‘LKB top menu’ for the menu and ‘LKB interaction window’ for the window in which messages appear for both the Mac and CLIM versions. The illustrations

Figure 3.2: Selecting the script le in ACL/CLIM

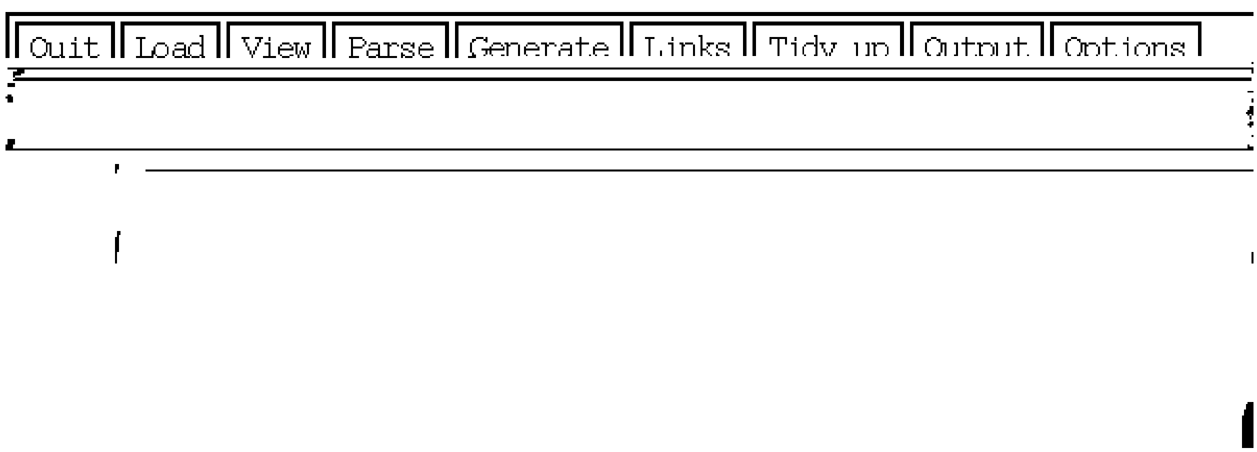

Figure 3.3: Loading a le

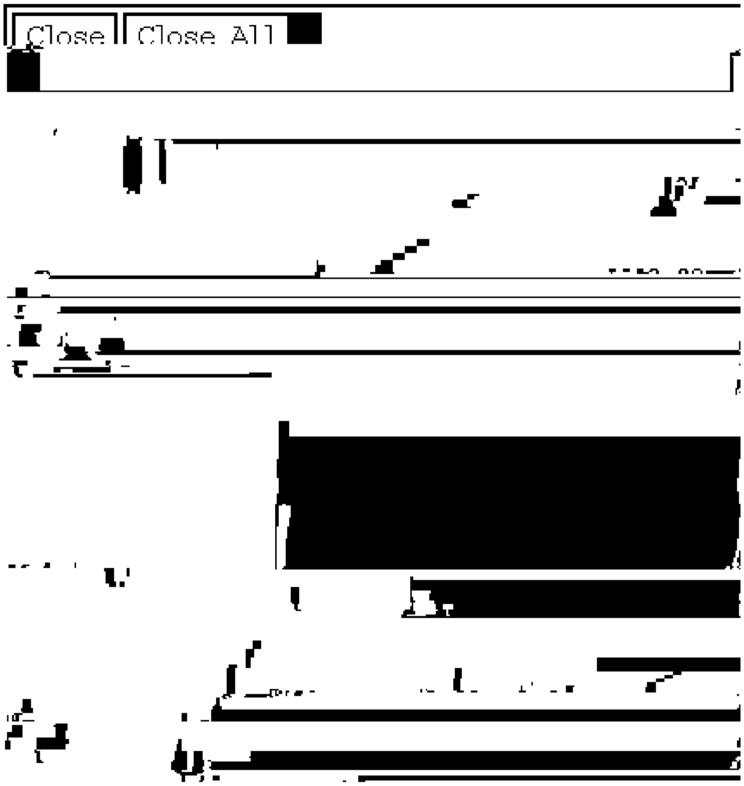

Figure 3.4: The type hierarchy window

Figure 3.5: An expanded type window

Figure 3.6: A complex type constraint

Look at the entry for the type ne-list in the actual source le (i.e., open

Figure 3.7: The head speci er rule

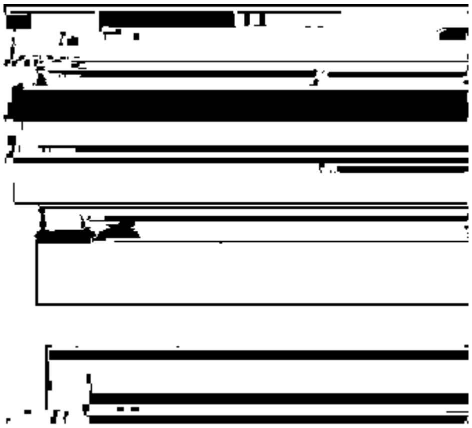

Figure 3.8: The parse result window in ACL/CLIM

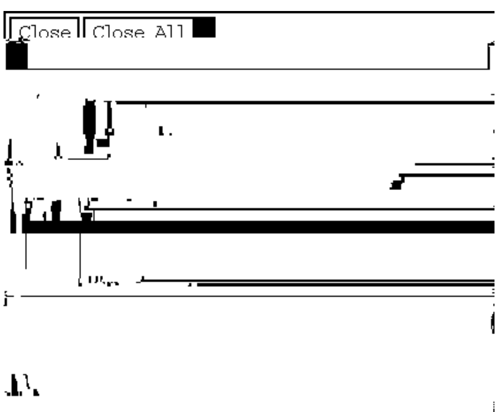

Figure 3.9: A parse tree window

grammar. A suitable le, , already exists in the toy directory. Select

Chapter 4

Feature structures and types in the LKB system

: $\mathsf{Feat} \times Q \to Q$: *the partial feature value function (transition function)*

: $Q \to \mathsf{Atom}$: *the partial atomic value function*

with the following constraints:

Connectedness *There must be some* path *from the root to every node.*

Paths are defined as sequences of zero or more features. Let $\mathsf{Path} = \mathsf{Feat}^*$ *and* *be the empty path. The definition of the transition function can be extended to paths by taking* $(\ ; q) = q$ *and* $(f\ ; q) = (\ ; (f, q))$

if $_F$ $'$ *then* $_{F'}$ $'$

if $P_F(\ ) =$ *then* $P_{F'}(\ ) =$

Thus F subsumes F' if and only if every piece of information in F is contained in

or not. We use $?$ as a notational convenience to indicate inconsistency, just as $?$ is used to indicate unification failure for feature structures.

4.2.2 Typed feature

Initially we will define the set F of typed feature structures to include both those which

(Path Value)
$P_F(\ ) = t$

2

6
4

would give rise to the following feature structure:

$$C(t_1) = \begin{bmatrix} t_1 \\ \mathrm{F}\ t_4 \end{bmatrix} \quad C(t_2) = \begin{bmatrix} t_2 \\ \mathrm{G}\ \top \end{bmatrix} \quad C(t_3) = \begin{bmatrix} t_3 \\ \mathrm{F}\ t_5 \\ \mathrm{G}\ \top \\ \mathrm{H}\ \top \end{bmatrix}$$

We then have: $C(t_1) \sqcap C(t_2) = \begin{bmatrix} t_3 \\ \mathrm{F}\ t_4 \\ \mathrm{G}\ \top \end{bmatrix}$

Then $t_3 = t_1 \sqcap t_2$ but $C(t_3) \sqsubset C(t_1) \sqcap C(t_2)$

Consider the following well-formed edature structures, F_1, F_2 and F_3:

$$F_1 = \begin{bmatrix} t_1 \\ \mathrm{F}\ t_4 \end{bmatrix} \quad F_2 = \begin{bmatrix} t_2 \\ \mathrm{G}\ \top \end{bmatrix} \quad F_3 = \begin{bmatrix} t_1 \\ \mathrm{F}\ t_6 \end{bmatrix}$$

$$F_1 \sqcap F_2 = \begin{bmatrix} t_3 \\ \mathrm{F}\ t_4 \\ \mathrm{G}\ \top \end{bmatrix}$$

$$F_2 \sqcap F_3 = \begin{bmatrix} t_3 \\ \mathrm{F}\ t_6 \\ \mathrm{G}\ \top \end{bmatrix}$$

$F_1 \sqcap F_2$ is not a well-formed fature structure oedtype t_3 as $F_1 \sqcap F_2 \not\sqsubseteq C(t_3)$. To extend itdto a well-formed fature structure invol

The type head-feature-principle has two parents:

Here the mother is given by the empty path, one daughter by the path `args.first` and

Finally, the syntax allows for *dotted-pairs*

Chapter 5

LKB source les

The previous chapter described the formalism used by the LKB. This chapter goes into the details of how the system evaluates grammar source les expressed in that formalism. Most of this chapter shou8d be equally relevant to people using the graphical or tty versions of the system. The source le organization is based on the assumption that a single le contains

classes over multiple les, for example to have di erent type les 341(t)-4(o)-316(h3erudl35(t)26(yp)-1(les

5.2.2 Conditions on the type hierarchy

After a syntactically valid type le (or series of type les) is read in, the hierarchy of types is constructed and checked to ensure it meets the conditions speci ed in x4.2.1.

All types must be de ned If a type is speci ed to have a parent which is not de ned anywhere in the loaded les, an error message such as the following is generated:

Although it is conventional to de ne parent types bef1(h)-148 -oitioer-408(b)0(n,)401(i)-2(o)he148

Redundant links This is the situation where a type is speci ed to be both an immediate and a non-immediate descendant of another type.

No Cycles Feature structures are required to be acyclic in the LKB system (see x4.1 and
x

With this definition, the previously impossible feature structure can be strongly typed without causing an infinite structure:

2
-
4

5.6 Irregular morphology

Irregular morphology may be speci ed in association with a particular lexical rule, but it

5.7 Rule application during parsing

At this point, it is useful to bring together a few details about how rules are applied during parsing. It's important to notice that there is considerable flexibility in the way rules are applied, since as far as possible we try not to build in any unnecessary assumptions about processing. A far as the parser i concerned, the primary distinction i between rules that a ect spelling changes (i.e. morphological rules) and those that don't. The system makes the assumption that all morphological processing happens at the level of an individual word (althoughee x7.4). Thus, when parsing, all xation operations apply before grammar rules.[3] This means that it is the grammar writer's responsibility totop rules applying in unwanted ways. On th whole, this is dn via th typ system, as was discussed abve in xThisel a5-323r9riw31(1.439(a)-31u8(disn,les)3(.)2Tj/F225 737f0.442 0 TD-0.ed)se1-c5(c)3-n3t5sineh yci1 instances can be stored in a separate le. Root instances may be viewed via View Lex Entry.

5.9 Parse tree node labels

The parse node le allows feature structures which control the labelling of the nodes in the parse tree to be de ned (for parse tree display, see 6.5). We will refer to these structures as templates, but this use shouldn't be confused with any other use of the term. In very general terms, a node will be labelled with a template, i the feature structure associated with the node is matched by the feature structure(s) associated with that template. The
parse node descriptions is identical to that of lexical entries. Par(s)4(e)-300(no)-34(des)-308(m)1(a)26(y)-2

To calculate the label for a node, the label templates are rst checked to nd a match. Matching is tested by uni cation of the template feature structure, excluding the label path, with the feature structure on the parse tree node. For instance, if a parse tree node had the following feature structure:

it would unify with

Chapter 6

LKB user interface

In this chapter, we explain the details of the menu commands and other aspects of the

6.1.1 Quit

ACL/CLIM only. This closes the LKB top window. It may be reopened by evaluating (clim-user::restart-lkb-window).

6.1.2 Load

Type definition Shows the definition of a type.

In the tty version of the system, this outputs a bracketed structure | otherwise the normal graphical parse output is produced.

Show parse Shows the tree(s) from the last parse again.

Show chart Shows the parse chart (see *x*6.6).

Batch parse This prompts for the name of a le which contains sentences on which you wish to check the operation of the parser, one sentence per line (see the le in

Save display settings Save shrunkenness of feature structures

Load display settings Load pre-saved display setting le.

6.1.8 Generation*

The generator is described in more detail in x7.12, but is currently in a fairly early stage of development. It operates in a very similar manner to the parser and relies on the use of an MRS style semantics. At the moment, there is no way of entering an MRS input other than by parsing a sentence which produces that MRS.

Generate ... Only usable if a sentence has been parsed. Prompts the user for the number

further undisplayed structure is indicated by a box round the type. Atomic feature structures menot be shrunk. Shrinking modi es the actual feature structure being displayed, so that if the window is closed, and subsequently a new window opened onto that feature structure, the shrunken status will be retained. Furthermore, any feature structures which are formed by uni cation with the shrunken feature structure, will also be displayed with equivalent parts hidden. Thus if the *expanded* constraint on a type has parts shrunk, any lexical entry which involves

6.3.1 Uni cation checks

The uni cation check mechanism operates on feature structures that are displayed in windows. One can temporarily select any feature structure or part of a feature structure by clicking on the relevant node in a displayed window and choosing **Select** from the menu. Then check whether this structure uni es with another, and get detailedessages if uni cation fails, nd the no corresponding the second structure, click on that, and choose **Unify**. the uni cation fails, failure messages will be shown in the top level LKB window. If it succeeds, a new feature structure window wille displayed. This can in turn e used tioncheck further uni cations.

Forxample, consi418der why \Kim sleep" dos note i418n the buggy grammar. In fact, the parse is bloked by the ro mechanism because we will want license the phrase \Kim sleep" in somontexts, e.g., *Sandy prefers that Kimep* . By looking at the parse chart for the9(s)4(en)31(tence,)-361(one)-3(9(c)3(an)-340(determine)-349(t)-2(hat)-333(t)-2(he)-3(9(hd-)-6(sp)-2
0(t)-2(he)-3feature structure9(n)1(amed)-330(b)31(y)]TJ/F2c[(.)-9330(tew3(lic)8.3463otUnify)Tj/F21 1 0412.938 Tcnif

The details of the parse tree display are discussed in the next section.

The parse output display is currently only available on the ACL/CLIM version of the

Chapter 7

Advanced features

instance, the textbook grammar is not a particularly good model for anyone concerned with

7.2.1 Loading functions

The following is a full list of available functions for loading in LKB source files written using the TDL syntax. All files are specified a full pathnames. Unless otherwise specified, details parent-directory

Returns the directory which is contains the directory containing the script le (only usable inside the script le).

temporary-directory

Returns the user's temporary directory, as set up in . By default, this is . (Note that it is important that each user have a distinct temporary directory.)

temporary- le *name*

Speci es a le *name* in the temporary directory.

lkb-pathname *directory name*

Takes a directory as speci ed by the commands above, and combines it with a le name to produce a valid full name for the other commands.

lkb-load-lisp *directory name* &optional *boolean*

Constructs a le name as with and loads it as a lisp le. If optional is (i.e., true), ignore the le if it is missing, otherwise signals a (continuable) error.

7.3 Check paths

The check

7.7 Cached lexicon

7.9 Emacs interface

FIX (though I'm not sure whether we need this section) - maybe Rob?

7.10 YADU

The structures used in the LKB system are not actually ordinary typed feature structures,

7.11 MRS

7.12 Gen702Gion

7.13 TSDB

Appendix A

The tty interface

show-type-tty *type*

Displays the expanded type constraint. For example,

A.7 Generation

edge-number

show-gen-result
print-gen-chart

A.8 Miscellaneous

Appendix B

Path syntax

Earlier versions of the LKB used a syntax for descriptions which was based on the path value

B.2 Instance descriptions

The syntactic description for lexical entries is as follows:

AppendiL C

Details of system parameters

independent. (Not all grammar independent variables are available under Options, since some are complex to set interactively, or rarely used.)

C.1 Grammar independent global variables

C.1.1 System les

sense-unif-fn, nil If set, this must correspond to a function. See make-sense-uni cations in xC.3, below.

maximum-number-of-edges, 500 A limit on the number of edges that can be created

C.1.4 Parse tree node labels

and ebisting grammar, in a context where multiple type inheritance was being used for

***deleted-daughter-features*, nil** A list of features which will not be passed from daughsince the value of the orthography string and the semantic relation name are predictable from the identifying material.

make-orth-fs A function used by the parser which takes a string and returns a feature

nd-infl-poss This function is called when a lexical item is read in, but only for lexical items which have more than one item in their orth value (i.e., multiword lexemes). It must be de ned to take three arguments: a set of uni cations (in the internal data structures), an orthography value (a list of strings) and a sense identi er. It should return an integer, indicating luerT92(e)2(lemen)30(t)-404(in)-381(a)-376(m)31(ulti-)-7(w)33(ord)-T92 left to right, leftmost is 1) or J

References

Briscoe, E.J., A. Copestake and V. de Paiva (1993) *Inheritance, defaults and the lexicon,* Cagbridge University Press.

Carpenter, R. (1992)