A TOOL FOR THE AUTOMATIC CREATION, EXTENSION AND UPDATING OF LEXICAL KNOWLEDGE BASES

Walter M.P. Daelemans
AI-LAB
Vrije Universiteit Brussels
Pleinlaan 2 Building K
B-1050 Brussels
Belgium
E-mail: walterd@arti.vub.uucp

ABSTRACT

A tool is described which helps in the creation, extension and updating of lexical knowledge bases (LKBs). Two levels of representation are distinguished: a static storage level and a dynamic knowledge level. The latter is an object-oriented environment containing linguistic and lexicographic knowledge. At the knowledge level, constructors and filters can be defined. Constructors are objects which extend the LKB both horizontally (new information) and vertically (new entries) using the linguistic knowledge. Filters are objects which derive new LKBs from existing ones thereby optionally changing the storage structure. The latter use lexicographic knowledge.

INTRODUCTION

Despite efforts in the development of tools for the collection, sorting and editing of lexical information (see Kipfer, 1985 for an overview), the compilation of lexical knowledge bases (LKBs, lexical databases, machine readable dictionaries) is still an expensive and time-intensive drudgery. In the worst case, a LKB has to be built up from scratch, and even if one is available, it often does not come up to the requirements of a particular application. In this paper we propose an architecture for a tool which helps both in the construction (extension and updating) of LKBs and in creating new LKBs on the basis of existing ones. Our work falls in with recent insights about the organisation of LKBs.

The main idea is to distinguish two representation levels: a static *storage level* and a dynamic *knowledge level*. At the storage level, lexical entries are represented simply as records (with fields for spelling, phonetic transcription, lexical representation, syntactic category, case frames, frequency counts, definitions etc.) stored in text files for easy portability. The knowledge level is an object-oriented environment, representing linguistic and lexicographic knowledge in a number of objects with attached information and procedures, organised in generalisation hierarchies. Records at the storage level are lexical objects in a 'frozen' state. When accessed from the knowledge level, these records 'come to life' as structured objects at some position in one or more generalisation hierarchies (record fields are interpreted as slot fillers). This way, a number of procedures becomes accessible (through inheritance) to these lexical objects.

For the creation and updating of dictionaries, *constructors* are defined: objects at the knowledge level which compute new lexical objects (corresponding to new records at the storage level) and new information attached to already existing lexical objects (corresponding to new fields of existing records). To achieve this, constructor objects make use of information already existing in the LKB and of the linguistic knowledge represented at the knowledge level. Few constructors can be developed which are complete, i.e. which can operate fully automatically without checking of the output by the user. Therefore, a central part in our system is a cooperative *user interface*, whose task it is to reduce initiative from the user to a minimum.

Filters are another category of objects. They use an existing LKB to create automatically a new one. During this transformation, specified fields and entries are kept, and others are omitted. The storage strategy used may be changed as well. E.g. an indexed-sequential file of phoneme representations could be derived from a dictionary containing this as well as other information, and stored in another way (e.g. as a sequential text file). The derived lexical knowledge base we call a *daughter dictionary* (DD) and the source LKB *mother dictionary* (MD). Filters use the lexicographic knowledge specified at the knowledge level. In principle, one MD for each language should be sufficient. It should contain as much information as possible (see Byrd, 1983 for a similar opinion). Constructors can be developed to assist in creating, extending and updating such an MD, thereby reducing its cost, while LKBs for specific applications or purposes could be derived from it by means of filters. The basic architecture of our system is given in Figure 1.

Current and forthcoming storage and search technology (optical disks, dictionary chips) allow us to store enormous amounts of lexical data in external memory, and retrieve them quickly. In view of this, the traditional storage versus computation debate (should linguistic information be retrieved or computed?) becomes irrelevant in the context of language technology. Natural Language

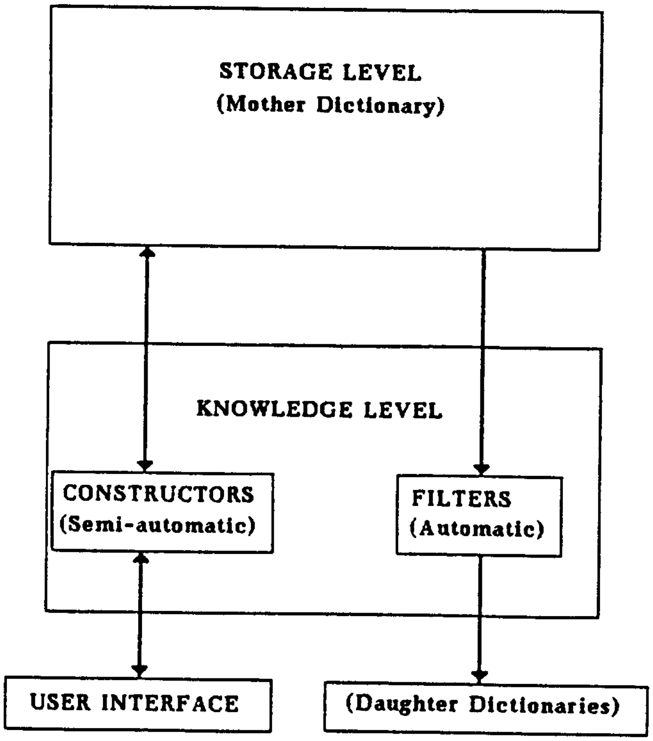


Figure 1. A System for Creating, Extending and Updating LKBs.

Processing systems should exhibit enough redundancy to have it both ways. For instance, at the level of morphology, derived and inflected forms should be stored, but at the same time enough linguistic knowledge should be available to compute them if necessary (e.g. for new entries). We think the proper place for this linguistic knowledge is the dictionary system.

There is some evidence that this redundancy is psychologically relevant as well. The duplication of information (co-existing rules and stored forms) could be part of the explanation for the fuzzy results in most psycholinguistic experiments aimed at resolving the *concrete* versus *abstract* controversy about the organisation of the mental lexicon (Henderson, 1985). The concrete hypothesis states that it is possible to produce and interpret word forms without resort to morphological rules while the abstract hypothesis claims that in production and comprehension rules are routinely used.

THE KNOWLEDGE LEVEL

We used the knowledge representation system KRS (Steels, 1986) to implement the linguistic and lexicographic knowledge. KRS can best be viewed as a glue for connecting and integrating different formalisms (functional, network, rules, frames, predicate logic etc.). New formalisms can also be defined on top of KRS. Its kernel is a frame-based object-oriented language embedded in Lisp, with several useful features. In KRS objects are called *concepts*. A concept has a name and a concept structure. A concept structure is a list of subjects (slots), used to associate declarative and procedural knowledge with a concept. Subjects are also implemented as concepts, which leads to a uniform representation of objects and their associated information.

KRS has an explicit notion of meaning: each concept has a *referent* (comparable to the notion of *extension*) and may have a *definition*, which is a Lisp form that can be used to compute the referent of the concept within a particular Lisp environment (comparable to the notion of *intension*). This explicit notion of meaning makes possible a clean interface between KRS and Lisp and between different formalisms.

Evaluation in KRS is *lazy*, which means that new objects can always be defined, but are only evaluated when they are accessed. *Caching* assures that slot fillers are computed only once, after which the result is stored. The built-in *consistency maintenance* system provides the automatic undoing of these stored results when changes which have an effect on them are made. Different *inheritance* strategies can be specified by the user.

At present, the linguistic knowledge pertains to aspects of Dutch morphology and phonology. Our word formation component consists of a number of morphological rules for affixation and compounding. These rules work on lexical representations (containing graphemes, phonemes, morphophonemes, boundary symbols, stress symbols etc.) A set of spelling rules transforms lexical representations into spelling representations, a set of phonological rules transforms lexical representations into phonetic transcriptions. We have implemented object hierarchies and procedures to compute inflections, internal word boundaries, morpheme boundaries syllable boundaries and phonetic representations (our linguistic model is fully described in Daelemans, 1987).

Lexicographic knowledge consists of a number of sorting routines and storage strategies. At present, the definition of filters can be based on the following primitive procedures: sequential organisation, (single-key) indexed-sequential organisation, letter tree organisation, alphabetic sorting (taking into account the alphabetic position of non-standard letters like phonetic symbols) and frequency sorting.

Constructors can be defined using primitive procedures attached to linguistic objects. E.g. when a new citation form of a verb is entered at the knowledge level, constructors exist to compute the inflected forms of this verb, the phonetic transcription, syllable and morphological boundaries of the citation form and the inflected forms, and of the forms derived from these inflected forms, and so on recursively. Our present understanding of Dutch morphophonology has not yet advanced to such a level of sophistication that fully automatic extension of this kind is possible. Therefore, the output of the constructors should be checked by the user. To this end, a cooperative user interface was built. After checking by the user, newly created or modified lexical objects can be transformed again into 'frozen' records at the storage level. This happens through a translation function which transforms concepts into records. Another translation function creates a KRS object on the basis of a record.

Figure 2 shows a KRS object and its corresponding record. This record contains the spelling, the lexical representation, the pronunciation, the citation form (lexeme) and some morpho-syntactic codes of the verb form *werkte* (worked). (Records for citation forms contain pointers to the different forms belonging to their paradigm, and information relevant to all forms of a paradigm; e.g. case frames and semantic information). The corresponding concept contains exactly the same information in its subjects, but through inheritance from concepts like *verb-form* and *werken-lexeme*, a large amount of additional information becomes accessible.

```
werkte  werk#D@  wɛrktə  werken-lexeme 11210


(defconcept werkte-form
     (a verb-form
        (spelling [string "werkte"])
        (lexical-representation [string "*werk#D@"])
        (pronunciation [string "wErkt@"])
        (lexeme werken-lexeme)
        (finiteness finite)
        (tense past)
        (grammatical-number singular)
        (grammatical-person 1-2-3)))
```

Figure 2. A static record and its corresponding KRS concept.

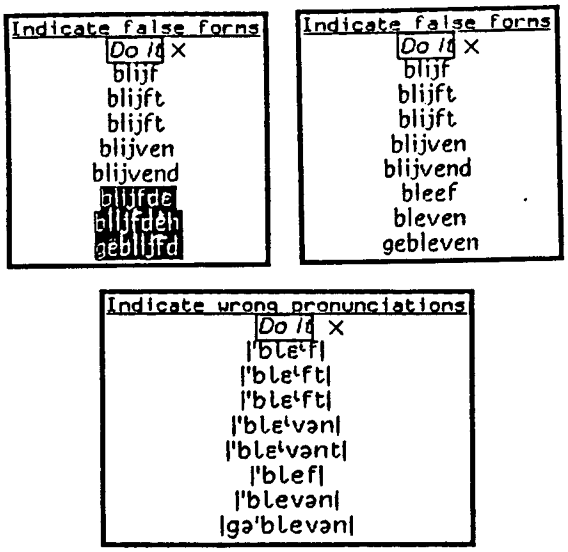
THE USER INTERFACE

We envision two categories of users of our architecture: *linguists*, who program the linguistic knowledge and provide primitive procedures which can be used as basic building blocks in constructors, and *lexicographers*, using predefined filters and constructors, creating new ones on the basis of existing ones and on the basis of primitive linguistic and lexicographic procedures, and checking the output of the constructors before it is added to the dictionary. The aim of the user interface is to reduce user intervention in this checking phase to a minimum. It fully uses the functionality of the mouse, menu and window system of the Symbolics Lisp Machine.

When due to the incompleteness of the linguistic knowledge new information cannot be computed with full certainty, the system nevertheless goes ahead, using heuristics to present an 'educated guess' and notifying the user of this. These heuristics are based on linguistic as well as probabilistic data. A user monitoring the output of the constructor only needs to click on incorrect items or parts of items in the output (which is mouse-sensitive). This activates diagnostic procedures associated with the relevant linguistic objects. These procedures can delete erroneous objects already created, recompute them or transfer control to other objects. If the system can diagnose its error, a correction is presented. Otherwise, a menu of possible corrections (again constrained by heuristics) is presented from which the user may choose, or in the worst case, the user has to enter the correct information himself.

Consider for example the conjugation of Dutch verbs. At some point, the citation form of an irregular verb (*blijven*, to stay) is added to the system, and we want to add all inflected forms (the paradigm of the verb) to the dictionary with their pronunciation. As a first hypothesis, the system assumes that the inflection is regular. It presents the computed forms to the user, who can indicate erroneous forms with a simple mouse click. Information about which and how many forms were objected to is returned to the *diagnosis procedure* associated with the object responsible for computing the regular paradigm, which analyses this information and transfers control to an object computing forms of verbs belonging to a particular category of irregular verbs. Again the forms are presented to the user. If this time no forms are refused, the pronunciation of each form is computed and presented to the user for correction, and so on. This sequence of events is illustrated in Figure 3.

Diagnostic procedures were developed for objects involved in morphological synthesis, morphological analysis, syllabification and phonemisation. At least for the linguistic procedures implemented so far a maximum of two corrective feedbacks by the user is necessary to compute the correct representations.



blijf
blijft
blijft
blijven
blijvend
blijfde
blijfden
geblijfd

blijf
blijft
blijft
blijven
blijvend
bleef
bleven
gebleven

|'blɛⁱf|
|'blɛⁱft|
|'blɛⁱft|
|'blɛⁱvən|
|'blɛⁱvənt|
|'blef|
|'blevən|
|gə'blevən|

Figure 3. Corrective feedback by the user: Erroneous forms are indicated (top left), second (and correct) try by the system (top right), presentation of the pronunciations of the accepted paradigm for checking by the user (down).

CONSTRUCTING A RHYME DICTIONARY

Automatic *dictionary construction* can be easily done by using a particular filter (e.g., a citation form dictionary can be filtered out from a word form dictionary). Other more complex constructions can be achieved by combining a particular constructor or set of constructors with a filter. For example, to generate a word form lexicon on the basis of a citation form lexicon, we first have to apply a constructor to it (morphological synthesis), and afterwards filter the result into a suitable format. In this section, we will describe how a *rhyme dictionary* can be constructed on the basis of a spelling word form lexicon in an attempt to point out how our architecture can be applied advantageously in lexicography.

First, a *constructor* must be defined for the computation of a broad phonetic transcription of the spelling forms if this information is not already present in the MD. Otherwise, it can be simply retrieved from the MD. Such a constructor can be defined by means of the primitive linguistic procedures *syllabification*, *phonemisation* and *stress assignment* The phonemisation algorithm should be adapted in this case by removing a number of irrelevant phonological rules (e.g. assimilation rules). This, too can be done interactively (each rule in the linguistic knowledge base can be easily turned on or off by the user). The result of applying this constructor to the MD is the extension of each entry in it with an additional field (or slot at the knowledge level) for the transcription. Next, a filter object is defined working in three steps:

(i) Take the broad phonetic transcription of each dictionary entry and reverse it (*reverse* is a primitive procedure available to the lexicographer).

(ii) Sort the reversed transcriptions first according to their rhyme determining part and then alphabetically. The rhyme determining part consists of the nucleus and coda of the last stressed syllable and the following weak syllables if any. For example, the rhyme determining part of *wérvelen* (to whirl) is *er-ve-len*, of *versnéllen* (to accelerate) *el-len*, and of *óverwérk* (overwork) *erk*.

(iii) Print the spelling associated with each transcription in the output file. The result is a spelling rhyme dictionary. If desirable, the spelling forms can be accompanied by their phonetic transcription.

Using the same information, we can easily develop an alternative filter which takes into account the *metre* of the words as well. Although two words rhyme even when their rhythm (defined as the succession of stressed and unstressed syllables) is different, it is common poetic practice to look for rhyme words with the same metre. The metre frame can be derived from the phonetic transcription. In this variant, step (ii) must be preceded by a step in which the (reversed) phonetic transcriptions are sorted according to their metre frame.

RELATED RESEARCH

The presence of both static information (morphemes and features) and dynamic information (morphological rules) in LKBs is also advocated by Domenig and Shann (1986). Their prototype includes a morphological 'shell' making possible real time word analysis when only stems are stored. This morphological knowledge is not used, however, to extend the dictionary and their system is committed to a particular formalism while ours is notation-neutral and unrestrictedly extensible due to the object-oriented implementation.

The LKB model outlined in Isoda, Aiso, Kamibayashi and Matsunaga (1986) shows some similarity to our filter concept. Virtual dictionaries can be created using base dictionaries (physically existing dictionaries) and user-defined Association Interpreters (AIPs). The latter are programs which combine primitive procedures (pattern matching, parsing, string manipulation) to modify the fields of the base dictionary and transfer control to other dictionaries. This way, for example, a virtual English-Japanese synonym dictionary can be created from English-English and English-Japanese base dictionaries. In our own approach, all information available is present in the same MD, and filters are used to create base dictionaries (physical, not virtual). Constructors are absent in the architecture of Isoda et al. (1986).

Johnson (1985) describes a program computing a reconstructed form on the basis of surface forms in different languages by undoing regular sound changes. The program, which is part of a system compiling a comparative dictionary (semi-)automatically, may be interpreted as related to the concept of a constructor in our own system, with construction limited to simple string manipulations, and not extensible unlike our own system.

CONCLUSION

We see three main advantages in our approach. First, the distinction between a *dynamic linguistic level* with a practical user-friendly interface and a *static storage level* allows us to construct, extend and maintain a large MD relatively quickly, conveniently and cost-effectively (at least for those linguistic data of which the rules are fairly well understood). Obviously, MDs of different languages will not contain the same information: while it may be feasible to incorporate inflected forms of nouns, verbs and adjectives in it for Dutch, this would not be the case for Finnish.

Second, the linguistic knowledge necessary to build constructor objects can be tested, optimised and experimented with by continuously applying it to large amounts of lexical material. This fact is of course more relevant to the linguist than to the lexicographer.

Third, efficient LKBs for specific applications (e.g. hyphenation, spelling error correction etc.) can be easily derived from the MD due to the introduction of *filters* which automatically derive DDs.

It may be the case that our approach cannot be easily extended to the domain of syntactic and semantic dictionary information. It is not immediately apparent how constructors could be built e.g. for the (semi-)automatic computation of case frames for verbs or semantic representations for compounds. Still, a heuristics-driven cooperative interface could be profitably used in these areas as well.

So far, we have invested most effort into the development of an object-oriented implementation of morphological and phonological knowledge for Dutch (i.e. in the definition of the primitive procedures which can be used by constructors), in the development of heuristics and diagnostic procedures, and in the design of the user interface. A prototype of the system (written in ZetaLisp and KRS, and running on a Symbolics Lisp Machine) has been built. Future efforts will be directed to the extension of the linguistic and lexicographic knowledge, the development of a suitable script language for the definition of constructors, and to the testing of our architecture on a large LKB. We think of using the Top10,000 dictionary which is being developed at the University of Nijmegen as a point of departure for the construction of a MD for Dutch. This LKB contains some 78,000 Dutch word forms with some morphological information.

ACKNOWLEDGEMENTS

This work was financially supported by the EC (ESPRIT project 82). My research on this topic started while I was working for the Language Technology Project at the University of Nijmegen. I am grateful to Gerard Kempen and Koen De Smedt for valuable comments on the text.

REFERENCES

Byrd, J.R. 1983 Word Formation in Natural Language Processing Systems. IJCAI-83, Karlsruhe, West Germany; 704-706.

Daelemans, W.M.P. 1987 *Studies in Language Technology. An Object-Oriented Computer Model of Morphophonological Aspects of Dutch.* Doctoral Dissertation, University of Leuven.

Domenig, M. and Shann P. 1986 Towards a Dedicated Database Management System for Dictionaries. COLING-86; 91-96.

Henderson, L. 1986 Toward a psychology of morphemes. In Ellis A.W. (Ed.) *Progress in the Psychology of Language, Vol. 1.* London: Erlbaum.

Isoda, M., Aiso, H., Kamibayashi N. and Matsunaga Y. 1986 Model for Lexical Knowledge Base. COLING-86; 451-453.

Johnson, M. 1985 Computer Aids for Comparative Dictionaries. *Linguistics* 23, 285-302.

Kipfer, B.A. 1985 Computer Applications in Lexicography — Summary of the State-Of-The-Art. *Papers in Linguistics* 18 (1); 139-184.

Steels, L. 1986 Tutorial on the KRS Concept System. Memo AI-LAB Brussels.