Gunter Gebhardi, Humboldt-Universität zu Berlin

Methods for Lexicon Maintenance

Abstract

The aim of the paper is to discuss the four basic methods for maintaining the content of lexicon entries. Thinking of the successful application of feature structure systems in computational linguistics, the paper suggests to introduce an additional operation into these systems to support lexicon maintenance. Our experience in using the lexicon formalism LeX4, which provides all the required operations, verifies the approach.

1. Introduction

The development of grammars and lexicons is very dynamic, especially in computational linguistics (CL) and it is a well known fact that it is not easy to update a lexicon while retaining and improving its quality. As a consequence, when developing a *lexicon formalism*, we do not merely seek a device for the adequate represention of linguistical data in a lexicon, but also aim to develop a unified and powerful tool which gives as much support as possible when *maintaining* a lexicon.

An analysis of the different ways of modification of lexicon *entries* gives four fundamental principles (using informal labels): *renaming and restructuring*, *views*, *extensions* and *variants*. This paper intends to discuss these four operations. Further analysis of these methods of modification of lexicon entries shows that we need only two basic operations and a mechanism of subselection to implement them. These operations are **removal** and **addition**. Taking into consideration that one of these operations is part of known CL formalisms, we can take these formalisms as a basis for the development of an extended formalism as a *unified* framework to support building-up and maintaining lexicons.

The outline of this paper: Firstly, we want to explain some basic assumptions of the lexicons we deal with, secondly, we consider the task of modifying the number of lexicon entries and the problem of selecting entries, thirdly, we discuss the four directions of modifying lexicon entries, and finally, we will give a conclusion with a brief look at our lexicon formalism LeX4.

2. Background and Motivation

The use of a lexicon as a part of a CL system presupposes a number of special properties of its content and structure. For example it has to be machine readable and sufficient for the purposes of information processing.

Currently *feature structures* or *typed feature structures* (for example Shieber (1986) and Carpenter (1992) form still the state of the art for representing and processing symbolic information in CL. The properties of feature structures and of the basic operations to deal with such structures are well known. The most important operation is *unification*. Unification joins the content of two data structures *if and only if* there is no contradiction in the content of both structures.

The contributions in Boguraev and Briscoe (1989) and in Briscoe, Copestake, and de Paiva (1993) as well as Emele and Heid (1993) show the usage of feature structure systems to encode lexical information.

Another idea is to use feature structures as basic data structures for linguistic databases. Ide, Maitre, and Veronis (1994) demonstrate how data of machine readable dictionaries can be stored using a device very similar to feature structures.

So we have a good motivation to use feature structures as basic structures for our lexicon, too, and to use unification as operation addition.

3. Some Basic Assumptions

A lexicon contains a set of entries. Each entry consists of a lexeme and a set of attached information. So we regard a lexicon as a two-dimensional system with the set of entries as one dimension and the set of information belonging to each entry as the other dimension.

From data base technology we know the two primitive operations to modify sets of entries: *insert* and *delete*. Very common is the combination of both operations to modify a set.

The operation of defining a subset of elements, maybe the set with only one element, is *selection*. Further we are able to define a (partial-) order over the set of elements and to arrange the elements corresponding to the order: *sorting*.

These considerations lead to the classification shown in table 1.

set	of entries	of information attached to an entry
order	sort	renaming and restructuring
insert	insert	extension
delete	delete	view
insert and delete	replace	variant
select	select	subselect

Table 1: Operations for modifying sets, sets of entries and sets of information attached to an entry

The operation *selection* identifies an entry on the basis of its content.

One problem is that we can make different use of one item within an entry. The item syntax for example may be the keyword, the domain of use or an identifier for the part of the entry's syntactic description. This is the reason why simple search tools are not suitable for solving the task of selection. Using more enhanced search and replacement programs with regular expression, we leave the basis of a unified framework.

The operation for selection has to take into account the structure of the entry as well as its content, i.e. the operation is data structure dependent. Feature structure unification is a data structure dependent operation and we can take this as a benefit.

Unification as a selectional operation has an interesting effect: it allows us to identify *underspecified* information. To select only non underspecified information we can use *subsumption* – another "classic" operation of feature structure formalisms – or the operation **removal**, which is part of the extended feature structure formalism we suggest here. Both operations are also data structure dependent. But it does not depend on any of these operations that selection will be always restricted to a test of unification, subsumption or removal.

4. The basic operations of maintaining single entries

4.1 Restructuring and Renaming

We consider *restructuring* and *renaming* as a bundle of two functions. These functions are similar in *not* changing the information content C of an entry – they neither add any information to, nor do they delete

information from the modified entry *C'*, i.e. $C = C'$. We can define a reverse functions to cancel the changes, i.e. $X = restructuring^{-1}(restructuring(X))$ and $X = renaming^{-1}(renaming(X))$.

We will see that the two basic operations removal and addition in combination with **subselection** will meet our requirements for operations for renaming and restructuring in a unified framework.

The function *renaming* serves to replace any occurrence of one name by another name. If, for example, the syntactic information is a value of an attribute syn, we may want to change the attribute's name to syntax; or we may want to spell out case names rather than using an abbreviation like gen for genitive.

Restructuring is somewhat more ambitious than renaming, because the *structure* of an entry will be changed. If, for instance, the description of case of an entry has to be modified from an old form case:(nom/acc) (case nominative or accusative) into nom:+,gen:–,dat:–,acc:+, the information content is not changed, but the *representation form.*

Two subtasks solve renaming or restructuring consistently for all entries of a lexicon. The first subtask selects all the entries in the lexicon which have to be modified. The second subtask identifies the information for renaming or restructuring, deleting this piece of information and inserting the corresponding new one *immediately.*

The first task was subject of section 3. The second task of restructuring has at first to identify that part of the information which has to be modified. Consider for example the information content of case:(nom/acc) which is identical with case:(acc/nom), but no problem arises if unification or one of the other operations are in use.

The next steps are removing the old pieces of information and adding the new one. We can describe all steps of the second subtask in a rule like

delete case: *and*
delete nom *and insert* nom:+ *or*
  by default insert nom:– *and*
delete / and insert, and
delete gen ...

Obviously, the function **removal** can fulfil the task of deleting the old information and **addition** that of inserting the new information. Subselection has an important role: because *renaming* and *restructuring* do not change the information content, the old piece of information can be deleted *if and only if* it can be replaced immediately by a renamed or restructured piece of information.

4.2 View

A *view* is a function which deletes information in an entry or in other words to restrict our view of the entry. As a consequence, the information content C of an entry is reduced, resulting in an entry with *less* content C'. So the relation $C' \subset C$ holds. It is impossible to define a function to cancel changes, i.e. we cannot define a function $X = view^{-1}(X')$ corresponding to $X' = view(X)$.

The operation **removal** in combination with **subselection** is an appropriate basis for implementing the function *view*.

The motivation for building up a new lexicon with entries containing *less* information is usually a task of configuration of a lexicon for a particular purpose. For example, information about certain readings of some lexemes in special domains can be removed. So the extent of the entries can be reduced and as a consequence, the application system can run more efficiently.

The task of calculating views of entries is to be split into two subtasks. The first is to select the relevant entries of the lexicon. The second is to subselect the information and to delete it. Obviously, we can use the same technology as the one used for renaming and restructuring, but without adding the modified information.

4.3 Extension

An *extension* is a function for adding information to an entry. The old entry contains less information C than the extended entry C'. So the relation $C \subset C'$ holds. We are able to define a reverse function $X = extension^{-1}(extension(X))$.

This reverse function for removing the additional information may be problematic if unification is the operation used, because it is impossible to define a reverse function for unification. An alternative way to implement the operation is to use unification in conjunction with subselection, which will give the possibility of defining a reverse function.

The operations **addition** and, if necessary, **subselection** are an appropriate basis for defining the function for extension.

The task of calculating extensions has three subtasks. The first subtask is again to select all the entries which are the basis for additional information. The second task is to add information. The third task which has to be solved *before* extension is to obtain the additional information. Unfortunately, we cannot discuss this aspect further here.

4.4 Variant

A *variant* is a function which deletes and adds information in an entry. It is impossible to decide whether the old or the new entry contains more or less information and the relation $C' \neq C$ with $\exists X(X \subset C \wedge X \subset C')$ *holds. Because of this property we call the new entry simply* a *variant* of the old one. A reverse function to cancel changes cannot be defined.

The function *variant* is the combination of the functions *view* and *extension*. The discussion has shown that the two basic operations **removal** and **addition** in combination with **subselection** meet the requirements for operations in a unified framework. The task of calculating a variant is the combination of the tasks of calculating views and extensions.

An example can show the usage of the function. The difference between two grammar versions we had to deal with was the way they handled prepositional objects. As a consequence, it was necessary to modify the entries of verbs. Some prepositional objects had the status of an obligatory object, but it was impossible to decide which only by looking at the preposition, because this depended on the verb *and* the preposition. So it was necessary to build up this part of the information from scratch again. The new variants of lexicon entries were calculated by removing the old information and adding new one. So the new lexicon was totally different with respect to this part of information, it was another *variant*.

5. Conclusion

The summary of our observations is shown in table 2.

set of information attached to an entry	set relation	basic operation(s)
renaming and restructuring	$C = C'$	removal and addition
view	$C \supset C'$	removal
extension	$C \subset C'$	addition
variant	$C \neq C'$ *with* $\exists X(X \subset C \wedge X \subset C')$	removal and addition

Table 2: Operations for modifying the information concerning to an entry

The four operations completely describe all possibilities of modifying entries and can be mapped onto two basic operations **addition** and **removal**.

Using the framework of feature structure systems, unification serves as operation for **addition**. The operation for removal is a special feature of our extended feature structure system. In practice we have achieved good results with two (slightly) different versions of this operation. **Subselection** makes use of both operations.

Also on the basis of this realization, we have implemented the lexicon formalism LeX4 ("LeXicon 4(fo(u)r)malism") (Gebhardi and Heinecke (1995)). This formalism is not restricted to its function as a tool for lexicon maintenance, but in our experience, it is well suited to this purpose. For our experiments, we use a lexicon with approximately 20,000 lexemes. To generate a particular version of the lexicon, we need on average only few hours (less than a working day): writing the rule system for modification and compiling the new lexicon. The time needed to edit the additional information depends on the content and varies widely. But even during periods of expensive expansion of the lexicon, it was always possible to deliver a consistent lexicon.

Acknowledgements

This work was funded by the German Federal Ministry of Education, Science, Research and Technology (BMBF) in the framework of the Verbmobil Project under Grant 01 IV 101 G. The responsibility for the content of this study lies with the author.

Thanks to Ines, Katherine, Chris, Johannes and Udo.

References

Boguraev, B. and T. Briscoe, eds. (1989): *Computational Lexicography for Natural Language Processing*. Harlow: Longman.

Briscoe, T., A. Copestake, and V. de Paiva, eds. (1993): *Inheritance, Defaults, and the Lexicon.* Cambridge: Cambridge University Press.

Carpenter, B. (1992): *The Logic of Typed Feature Structures.* Cambridge: Cambridge University Press.

Emele, M. and U. Heid. (1993): *Formal Specification of a Typed Feature Logic Based Lexical Representation Language.* Technical report, DELIS-Deliverable D-V-2, Universität Stuttgart.

Gebhardi G. and J. Heinecke (1995): *Lexikonformalismus LeX4.* Technical report, Verbmobil Technisches Dokument, Humboldt-Universität zu Berlin.

Ide, N., J. Le Maitre, and J. Véronis. (1994): *Outline of a Model for Lexical Databases.* In: Zampolli, Calzolari, and Palmer (1994).

Shieber, S.M. (1986): *An Introduction to Unification-Based Approaches to Grammar.* CSLI Lecture Notes, number 4. Stanford, CA: Center for the Study of Language and Information.

Zampolli, A., N. Calzolari, and M. Palmer, eds. (1994): *Current Issues in Computational Linguistics: In Honour of Don Walker.* Giardini editori e stampatori in Pisa, Kluwer Academic Publishers, Norwell, MA.