Lexeme-based Morphology: A Computationally Expensive Approach Intended for a Server-Architecture

Marc Domenig
Institut für Informatik der Universität Zürich
Winterthurerstr. 190, CH-8057 Zürich
domenig@ifi.unizh.ch

Abstract: This paper presents an approach to computational morphology which can be considered as being derived from the two-level model but differs from this substantially. *Lexemes* rather than *formatives* are the most important entities distinguished in this approach. The consequence is that a new formalism for the specification of morphological knowledge is required. A short description of a system called *Word Manager* will outline the characteristics of such a formalism, the most prominent of which is that different subformalisms for inflectional rules and word-formation rules are distinguished. These rules are applied separately though not independently and support the concept of lexicalization. The primary advantage of this is that the system can build up a network of knowledge on how formatives, lexemes, and rules depend on each other while individual lexemes are lexicalized. Thus, the system will know the inflectional forms of a lexeme, the destructuring of these forms into formatives, how the lexeme has been derived or composed if it is a word-formation, etc. This requires much memory, yet, the philosophy behind the approach is that the system runs as a *server* on a local area network, so that an entire machine can be dedicated to the task, if necessary.

1. Introduction

In recent years computational morphology has been dominated by the so-called finite-state approach. The discussion about this approach was reportedly started in 1981 by a presentation of Kaplan and Kay at an LSA meeting in New York. It gained momentum after the publication of Koskenniemi's thesis (Koskenniemi 1983), which introduced the widely acknowledged two-level model. This model had several advantages: one of them was that it could be implemented on relatively small machines, i.e. it was extremely economical and effective from a computational point of view. As a result of this, it could be used and tested by a large number of research groups. The original model was modified in different directions in the course of the following years. Bear, for instance, proposed to increase the model's expressiveness by replacing the finite-state framework of the two-level model's lexicon system by a more powerful, unification-based formalism (Bear 86, 88). A similar proposal was conceived by Russell, Pulman, Ritchie, and Black (1986). Kay (1986/87) proposed to increase the formalism's expressiveness in order to make it suitable for nonconcatenative morphology. These and many other efforts - also by Koskenniemi himself; see Kataja and Koskenniemi (1988) - were mainly directed towards an improvement of the model's capacity to handle different natural languages. An alternative approach was followed in the project which will be described in the following: here, the intention was not to maximize the system's capacity to handle different languages but to improve the original model's properties from the point of view of database theory. There were two reasons for this: firstly, our interest was limited to a restricted set of languages - primarily German, English, French, and Italian. Secondly, we felt that there was a great potential for improvements of the two-level model if it were redesigned on a somewhat 'larger' scale, i.e. for use on an environment of highly powered workstations linked by a local area network, and with design criteria derived from database theory.

According to our opinion, this latter claim proved to be correct in many respects; during the past few years, a cyclic prototyping process showed that the kind of system resulting from the application of database design criteria had indeed a number of advantages over the original two-level model. Naturally, these advantages had to be paid for, primarily in terms of size and complexity: the system which eventually emerged from this prototyping process, *Word Manager*, has therefore only remote affinity with the two-level model (as well as with most of the successor models that focussed on increasing the capability of handling different natural languages, for that matter). It is no longer 'small and beautiful', running on practically any personal computer, but complex and above all expensive as far as its memory requirements are concerned. The primary reason for this is that it follows what we like to call the *lexeme-based approach* to computational morphology, which we consider as an alternative to the so-called *formative-based approach* followed by the two-level model.

The distinction between these approaches will be the focus of this paper. We will argue that the lexeme-based approach is advantageous in many respects and should therefore be considered as an alternative to the formative-based approach under certain conditions. The argument will proceed as follows: first, we will give an explanation of the terminology chosen for the alternatives. Then, we will proceed with a short description of Word Manager - an exhaustive description will be published in (Domenig 1990). The conclusion will point out the main differences between the two approaches.

2. Terminology

The two-level model is a typical representative of what we call the formative-based approach to computational morphology. In this approach, *formatives* are the basic entities managed by the system. By formative we mean

a distributional segment of a word-form, independent of whether or not it is also a morph. Essentially, the formative-based approach considers computational morphology as a *parsing problem*, where the formatives are the terminals of some kind of grammar formalism. In contrast, the lexeme-based approach treats computational morphology not only as a parsing but also as a *database management* and a *knowledge representation* problem. Here, the basic entities managed by the system are *(morphological) lexemes*, though the notion of formative is known as well. A (morphological) lexeme roughly comprises the morphological knowledge represented by a traditional dictionary entry: firstly, the knowledge which is directly specified in the entry - which usually is an indication of an inflectional class as well as information about etymology and derivations based on the entry –, secondly, the knowledge that can be inferred from this information by the interpretation of rules specified elsewhere in the dictionary (usually in the introduction). The latter typically includes knowledge how the lexeme's inflectional paradigms can be generated, and how derivations and compounds can be built. The objective of the lexeme-based approach, then, is to provide a formalism which allows the formalization of the morphological knowledge represented by dictionary entries. This implies that the formalism has to be expressive enough to represent morphological lexemes in the above sense. More specifically, it means that the formalism must have the following capabilities:

- It must provide the means to formalize inflection in a way which records
  - for formatives how they *have been or can (potentially) be* combined into word-forms,
  - for word-forms how they *have been or can (potentially) be* structured into sequences, and how these sequences have been or can be associated with lexemes in such a way that the system will know the citation form(s) and inflectional paradigm(s) for each individual lexeme.
- It must provide the means to formalize word-formation in a way which records for formatives whether and how they
  - *have been created* by word-formations,
  - *have been used* in word-formations (in order to create other formatives/lexemes),
  - *can (potentially) be used* in word-formations.

Undoubtedly, formalisms following the formative-based approach do have some of these capabilities. Since they lack the notion of a lexeme in the sense we have defined it, however, they are not able to deal with all of them. How the full functionality proposed here can be provided will be outlined in the following description of Word Manager.

3. Word Manager

Word Manager distinguishes two phases in the knowledge specification process: a first phase where so-called *lexeme classes* are specified and a second phase where *instances of these classes* are specified. By convention, the class specifications have to be processed by the system (compiled) before the instances can be made. Therefore, the formalism supported by Word Manager can be considered to be split in two parts, a notion which is supported by the fact that the system distinguishes separate interfaces for the specification of the classes and the instances. The first of these interfaces is an interactive specification environment, the second is an interprocess communication protocol provided by an application which runs as a server on a local area network (compare with Figure 5 shown below). Both interfaces are described elsewhere in some detail (Domenig 1988, 1990). Here, we will focus the description on the formalism for the specification of the lexeme classes. This formalism is split into several subformalisms, where these serve for the specification of rules on the one hand, of formatives on the other hand. The fundamental types of rules are *spelling rules*, *inflectional rules*, and *word-formation rules*. The fundamental types of formatives are *inflectional formatives* and *word-formation formatives*.

Spelling rules

The function of the spelling rules is similar to that of the two-level rules in Koskenniemi's two-level model. They allow the formatives used for the construction of word-forms to be defined as linguistically motivated abstractions, so that the rules ensure that they are mapped onto correct orthographic representations of word-forms. The rules themselves are similar to those in the two-level model as well, though not identical. The main difference is that the rules in Word Manager are applied when entries are made into the database - at compile time, as it were - while the rules in the two-level model are applied at run-time (by a finite-state transducer). Thus, the analysis of word-forms is (at least conceptually) faster in Word Manager than in the two-level model, at the cost of increased space requirements for the storage of the surface strings corresponding to the lexical strings.

Inflectional rules

The inflectional rules serve the purpose of defining the way in which inflectional formatives may be combined into word-forms and how these word-forms may be grouped into sequences defining what we have called morphological lexemes. Each inflectional rule defines one lexeme class where each class is distinguished by a particular sequence of word-forms that will be defined for each lexeme belonging to this class (the sequence of word-forms of each lexeme class is partitioned into subsequences of citation forms and paradigms). Thus, the rule shown in Figure 1 will define the inflectional forms "Massage" and "Massagen" for the lexeme with the citation form "Massage", for instance.

Entries into the database may be considered as *instances* of lexeme classes, i.e. each entry is an instance of one particular lexeme class. Entries are made by the *firing* of inflectional rules in a particular mode. Notice that this is the *only way* how entries can be made, which means that the system controls that there are no individual formatives or word-forms 'floating around' in the database. Instead, each formative and word-form is associated with at least one lexeme.

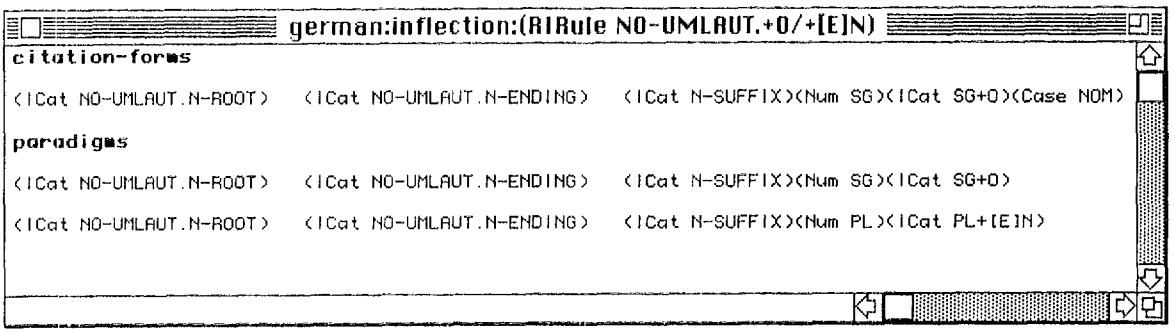

Figure 1: Inflectional rule for a German class of nouns

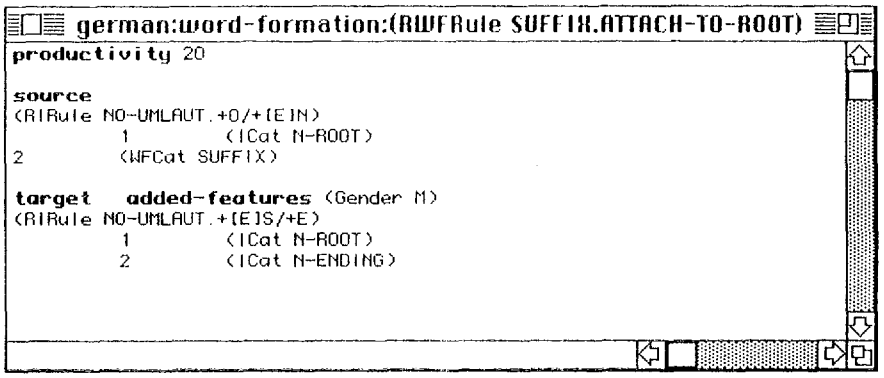

Figure 2: Word-formation rule for German noun derivations

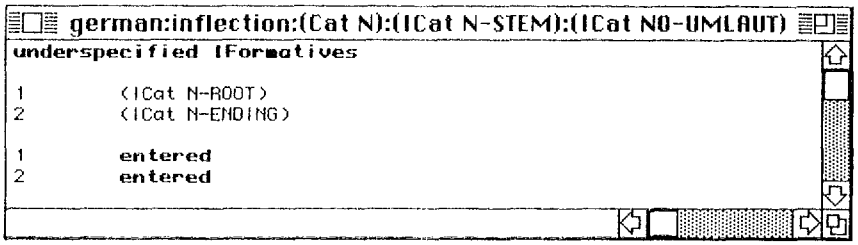

Figure 3: Underspecified formatives representing noun stems

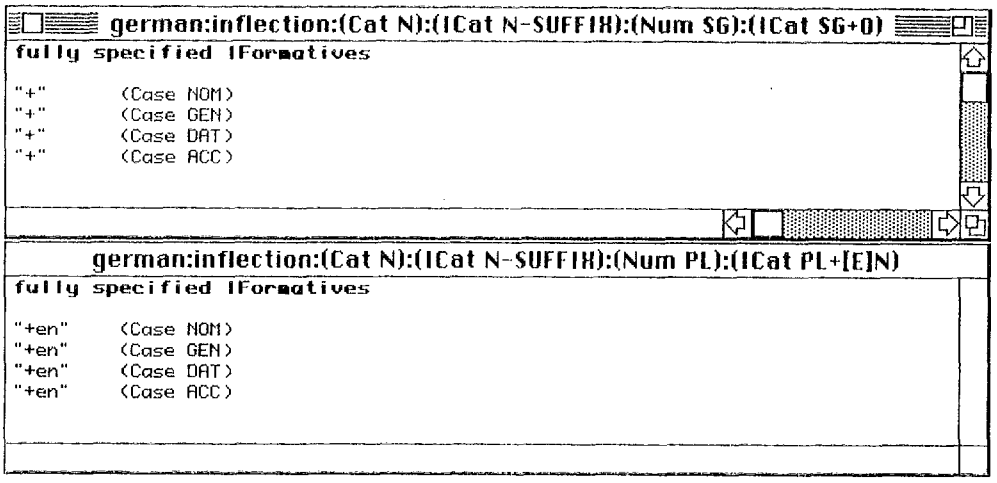

Figure 4: Fully specified formatives representing inflectional affixes

Word-formation rules

The word-formation rules serve the purpose of defining the way in which inflectional formatives of existing lexemes and word-formation formatives may be combined into inflectional formatives of new lexemes. When word-formation rules are fired, they will fire inflectional rules, thus instantiating new lexemes. Figure 2 shows a rule which could be used to derive the German noun "Masseur" from "Massage". The assumption made by this rule is that "Mass" is defined as root, "age" and "eur" as ending and derivational affix, respectively. A further, similar rule could be used for the derivation of the verb "massieren" from "Massage". Notice that the rule shown in Figure 2 is not realistic in the sense that it is too simplistic, i.e. it does not make use of the possibilities for generalizations provided by the formalism. A description of these features would require too much space, though.

Inflectional formatives

Inflectional formatives are lexical strings which are associated with feature sets (sets of attribute value pairs). They can be added to the database in two different ways: either they can be 'hard-coded' into the system as so-called *fully specified formatives*, or they can be added by the instantiation of so-called *underspecified formatives*. Underspecified formatives are formative specifications where the strings are missing. If the user wants to make an instance of such a formative, he has to provide a string. This he can do in either of two ways: by a direct specification or by the firing of a word-formation rule.

The underspecified formatives are the key to how lexemes can be entered into the database, because their instantiation is the prerequisite for the inflectional rules to be firable. More specifically: the strings needed for the instantiation of the underspecified formatives matched by an inflectional rule are the knowledge that has to be specified in order to make an instance of a lexeme class. Thus, it is evident that the underspecified formatives will typically be used for the representation of *stems* of lexemes (see Figure 3), while the fully specified formatives will typically be used for the representation of *inflectional affixes* (see Figure 4).

Word-formation formatives

Word-formation formatives are - like inflectional formatives - lexical strings which are associated with feature sets. Their typical role is to represent derivational affixes (like the suffix "eur" in the example above which is used for a noun to noun derivation).

Rule application

The application of the rules will take place at different times and some of the rules can be applied in different modes. The spelling rules are only fired when entries are added to the database, i.e. when instances of lexeme classes are created. This means that lexical strings will be converted into surface strings, so that the system will not have to apply the rules when entries are retrieved by the analysis of orthographic representations of word-forms; then, the system will parse on surface strings only.

The inflectional rules are applied both when lexemes are added to the database and when they - or parts thereof - are retrieved. When lexemes are added, the rules are applied in a generative mode which computes all the word-forms which belong to an individual lexeme (this is the point where the spelling rules are applied in order to compute the surface strings corresponding to the lexical strings). When lexemes are retrieved, the rules can be applied in an analytical mode which allows the identification of lexemes on the basis of the orthographic representation of individual word-forms. Notice that once a lexeme has been identified, the inflectional rule which is associated with it can be fired in generative mode again in order to compute all the word-forms which belong to the lexeme (this can be useful for illustrative purposes in a lexicographer interface, for instance).

Like the inflectional rules, word-formation rules are both applied when lexemes are added to the database and when they are retrieved. The modes for their application are more differentiated, though: Word Manager distinguishes two modes for their generative, and one mode for their analytical application. The first of the generative modes is used for what we call the *lexicalization* of complex lexemes. Such a lexicalization will instantiate a new lexeme, where this instantiation will have the effect that a future retrieval of the lexeme will not require the firing of the word-formation rule any more (but only the corresponding inflectional rule). The second of the generative modes is used for what we call the generation of *tentative entries*, i.e. the generation of lexemes which are *licensed* by the word-formation rules. Since the number of these tentative entries is typically infinite (because the rules will typically be defined in a way which allows them to be applied recursively), this mode is used rarely and primarily for illustrative purposes.

The firing of word-formation rules in analytical mode allows the analysis of word-forms of tentative entries. This is not only useful for the identification of 'unknown' (not lexicalized) lexemes but also for the construction of a database, because tentative entries are the ideal basis for the lexicalization process: once a tentative entry has been identified, the rule which licenses the entry is known. This means that all we have to do in order to lexicalize it is to fire that rule in the first of the generative modes mentioned. Thus, it is easy, for instance, to construct a dedicated terminological dictionary while scanning through a text corpus.

4. Word Manager databases

Knowledge represented in Word Manager databases is accessed via the so-called *database management system (DBMS)* over the local area network. The DBMS is a stand-alone application intended to run as a *server*. A typical usage of Word Manager in an NLP environment is shown in Figure 5. This illustration shows that the system is meant to be used by client applications of varied nature. The intention is that each of these clients will manage its own database, where this contains the knowledge the application requires *in addition* to the knowledge provided by Word Manager. Given this

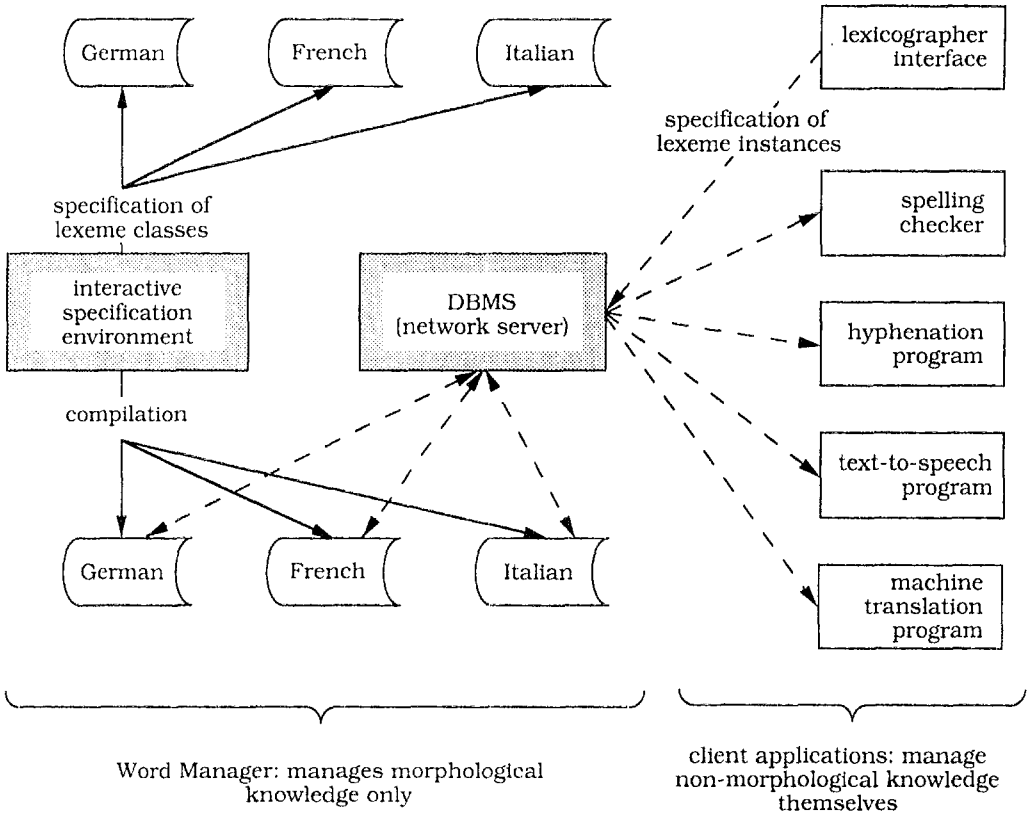

Figure 5: Word Manager in an NLP environment

framework, the importance of lexemes is evident, because the lexemes are the entities with which the clients will associate their application-specific knowledge: for every lexeme of a Word Manager database, a client will have a corresponding entry storing whatever it requires for its specific purposes. In order to make such associations possible, Word Manager provides so-called *lexeme identifiers* in the DBMS interface (where each of these identifies one lexeme unequivocally).

The actual knowledge of a Word Manager database comprises primarily the knowledge about a set of lexemes. Of each lexeme, the system knows

- all inflectional forms, whereas it can identify the lexeme on the basis of orthographic representations of its word-forms as well as generate the citation form(s) and inflectional paradigm(s) on the basis of an identified lexeme,
- the destructuring of word-forms into formatives,
- all word-formations which have been built on the basis of the lexeme and all word formations which can (potentially) be built on the basis of the lexeme. If the lexeme is itself a word-formation which has been instantiated by the firing of a rule (lexicalized), the system knows that rule and the lexeme(s) from which it has been derived.

This is not the whole story, of course. As pointed out at the beginning of section three, Word Manager distinguishes separate interfaces for the specification of lexeme classes and instances. In Figure 5, the interface for the specification of classes is represented by the box called 'interactive specification environment.' The interface for the specification of instances is represented by two boxes called 'DBMS' and 'lexicographer interface,' respectively. The former of these provides the interprocess communication protocol on the network, the latter is an end-user interface which technically is not a part of Word Manager, but a client application. This means that a Word Manager database not only knows a set of lexemes but also how new instances of lexemes can be created, and how existing lexemes can be modified or deleted. The interface provided by the DBMS must therefore not only offer functions for the analysis and generation of word-forms, etc., but also the possibility to view the formatives and rules, and the possibility to *fire* the rules for the creation of new entries.

The internal representation of a Word Manager database is a large network which links all entities that depend on each other - rules, lexeme class specifications, instances of lexeme classes, formatives, etc. Since entries are created by the firing of rules, the system can indeed keep track of all dependencies. This provides the possibility of creating an arbitrary number of *views* on the knowledge (the interprocess protocol allowing the computation of such views), because the entities linked by references can be combined in various ways. Thus, the knowledge can be presented to the user in different ways, in arbitrary levels of detail, for instance.

6. The prototype

The project which resulted in the design of Word Manager was started at the beginning of 1986. Since then, several prototypes were made, most of which implemented parsers for various parts and versions of the formalism, and the user interface for the 'interactive specification environment' shown in Figure 5. In the earlier phases of the project, three different machines were employed: the first was a Sun 3/50, which was primarily used in conjunction with the parser generator tools LEX and YACC. The second was a Lisp machine of the Texas Instruments Explorer family, which was used to prototype user interfaces. The third was an Apple Macintosh II. This machine became available towards the end of 1987, when the design of both the formalism and the 'specification environment' were in a state which suggested their full implementation. The following two years were spent on this task, which resulted in a prototype that includes about 16,000 lines of code (most of it written in Object Lisp (ALLEGRO 1987), an extension of Common Lisp). Notice that this prototype does not include the DBMS, though it provides the full functionality of the 'specification environment;' since the operating system did not provide interprocess communication, it was impossible to realize the network architecture proposed in Figure 5. In order to complete the system, we have therefore switched back to a UNIX environment, i.e. we are currently porting the prototype produced on Macintosh IIs to Sun SPARCstations.

6. Conclusion

Experience with the prototype of Word Manager has shown that the lexeme-based approach to morphology has advantages as well as disadvantages in comparison to the formative-based approach. On the side of the advantages, we claim that it is better suited for the implementation of application independent repositories of morphological knowledge, because a database realized with Word Manager knows more about its entries than a corresponding database realized with the two-level model, for instance. Moreover, the fact that entries are always made by the instantiation of lexeme classes has the effect that the system can execute a tight control over the consistency of the data. On the side of the disadvantages, we must admit that a system like Word Manager requires a much more powerful machinery than a system like the two-level model. In particular the storage requirements are quite formidable. A second disadvantage concerns the system's capability of handling different natural languages, though this is probably a shortcoming of Word Manager and not the lexeme-based approach in general. In any case, Word Manager's formalism is certainly not powerful enough to handle Finnish or Hungarian adequately - not to speak of Semitic languages. It might well be that some of the advantages of the approach must be sacrificed if we were to design a lexeme-based system which covers as many languages as the two-level model.

To sum up: we believe that the lexeme-based approach to computational morphology will be useful for many NLP applications. In view of the computers available today - and the future development to be expected on the hardware market -, the drawback concerning the powerful machinery required seems to be quite unimportant. Finally, further research might even lead to lexeme-based systems with formalisms that are powerful enough to handle as many natural languages as the systems following the formative-based approach.

7. References

ALLEGRO (1987): *Allegro Common Lisp for the Macintosh, User Manual.* Coral Software Corp., Cambridge, MA, 1987.

Bear John (1986): "A Morphological Recognizer with Syntactic and Phonological Rules." In *Proceedings of the 11th International Conference on Computational Linguistics, COLING-86,* Bonn, August 25-29.

Bear John (1988): "Morphology with Two-Level Rules and Negative Rule Features." In *Proceedings of the 12th International Conference on Computational Linguistics, COLING-88,* Budapest, August 22-27.

Domenig Marc (1988): 'Word Manager: A System for the Definition, Access and Maintenance of Lexical Databases.' In *Proceedings of the 12th International Conference on Computational Linguistics, COLING-88,* Budapest, August 22-27.

Domenig Marc (1990): *Word Manager: A System for the Specification, Use, and Maintenance of Morphological Knowledge.* To be published (probably by Springer).

Kataja Laura, Koskenniemi Kimmo (1988): "Finite-state Description of Semitic Morphology: A Case Study of Ancient Akkadian." In *Proceedings of the 12th International Conference on Computational Linguistics, COLING-88,* Budapest, August 22-27.

Kay Martin (1986): *Two-Level Morphology with Tiers.* Unpublished research paper, Xerox Palo Alto Research Center.

Kay Martin (1987): "Nonconcatenative Finite-State Morphology." In *Proceedings of the Third Conference of the European Chapter of the Association for Computational Linguistics,* Copenhagen, April 1-3.

Koskenniemi Kimmo (1983): *Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production.* Doctoral thesis at the University of Helsinki, Department of General Linguistics, Publications No. 11.

Russell G. J., Ritchie G. D., Pulman S. G., Black A. W. (1986): "A Dictionary and Morphological Analyser for English." In *Proceedings of the 11th International Conference on Computational Linguistics, COLING-86,* Bonn, August 25-29.