Two-Level Morphology : A General Computational Model for Word-Form Recognition and Production

Koskenniemi, Kimmo

1983

http://hdl.handle.net/10138/305218

Koskenniemi, K 1983, Two-Level Morphology : A General Computational Model for Word-Form Recognition and Production. Publications, no. 11, University of Helsinki. Department of General Linguistics, Helsinki. < http://www.ling.helsinki.fi/~koskenni/doc/Two-LevelMorphology.pdf >

TWO-LEVEL MORPHOLOGY: A General Computational Model for Word-Form Recognition and Production

Kimmo KOSKENNIEMI

University of Helsinki
Department of General Linguistics
Hallituskatu 11–13
SF-00100 HELSINKI 10
FINLAND

PUBLICATIONS
No. 11
1983

TWO-LEVEL MORPHOLOGY: A General Computational Model for Word-Form Recognition and Production

Kimmo KOSKENNIEMI

ISBN 951-45-3201-5
ISSN 0355-7170
Offset Oy 1983

ABSTRACT

Two-level morphology: A general computational model for word-form recognition and production.

Koskenniemi, Kimmo Matti
University of Helsinki, SF

This dissertation presents a new computationally implemented linguistic model for morphological analysis and synthesis. The model incorporates a general formalism for making morphological descriptions of particular languages, and a language-independent program implementing the model. The two-level formalism and the structure of the program are formally defined. The program can utilize descriptions of various languages, including highly inflected ones such as Finnish, Russian, or Sanskrit. The new model is unrestricted in scope and it is capable of handling the entire language system as well as ordinary running text. A full description of Finnish inflectional morphology is presented in order to validate the model.

The two-level model is based on a lexicon system and a set of two-level rules. It differs from generative phonology in the following respects. The rules are parallel, as opposed to being sequentially ordered, as is the case with the rewriting rules of generative phonology. The two-level model is fully bidirectional both conceptually and processually. It can also be interpreted as a morphological model of the performance processes of word-form recognition and production. The model and the descriptions are based on computationally simple machinery, mostly on small finite state automata. The computational complexity of the model is discussed, and the description of Finnish is evaluated with respect to external evidence from child language acquisition.

PREFACE

This work is a part of the project 593 sponsored by the Academy of Finland and directed by professor Fred Karlsson. The four year project started in 1981 at the Department of General Linguistics at the University of Helsinki. The aim is to investigate automatic recognition of Finnish from morphological, syntactic and semantic aspects. This dissertation belongs to the morphological part of the project.

I am deeply endebted to my instructor, professor Fred Karlsson for encouragement, an inspiring research atmosphere, and constructive comments on the manuscript, to Lauri Karttunen and Martin Kay for fruitful ideas and for acquainting me with their research, and to Benny Brodda whose BETA program has been a valuable tool in the preliminary steps of this work.

My sincere thanks are also due to the Cultural Foundation of Finland and the Wäinö Aaltonen Foundation for scholarships, to my former employer, the Computing Centre of the University of Helsinki, to its director Lars Backström for enabling the preliminary studies which led to this thesis, and to my instructor in computer science, professor Martti Tienari. I am grateful to Eugene Holman for improving the English of the manuscript.

I am most grateful to my wife Pirjo for her encouragement and support, and I apologize to Anni, Ilkka, and Kerttu for the time for which they had better plans. I owe my parents, Matti and Sirkku a debt of gratitude for their continuous support.

Helsinki, December 1983 K.K.

CONTENTS

CHAPTER 1
INTRODUCTION

This dissertation proposes a two-level model as a framework for computational morphological analysis and synthesis. It incorporates an explicit new formalism for describing morphological and morphophonological phenomena. In addition to being motivated by the goal of the automatic analysis of Finnish, this new model is also motivated by more general linguistic and computational considerations. The work consists of a presentation of the model, a description of Finnish morphology and morphophonology according to the new formalism, and a computer program implementing the model.

1.1 A process model

There is great variation between the inflectional systems of different languages. Since the 1960's, linguists have widely used the formalism of generative phonology for the description of inflectional morphology and morphophonology. The generative notation is powerful and thus widely applicable, but its descriptive power is also the source of problems. The formalism does not provide a realistic framework for describing the dynamic processes of word recognition and production. The two-level model incorporates an alternative formalism, and it is better suited to the processual aspects than are standard generative models.

The two-level formalism is based partly on the same concepts as the formalism of generative phonology. The main difference is that the two-level model uses parallel rules where generative phonology uses rewriting rules applied one after another in a predetermined order. Parallel rules have the benefit of being conceptually and computationally simpler, and they avoid problematic rule interactions. The two-level formalism is presented in chapter 2.

1.2 Bidirectionality

Most models of morphology are structurally and/or processually unidirectional. With structurally unidirectional formalisms, such as rewriting systems, it is easier to discuss one direction, in particular the production of word forms, than the reverse one. Structural unidirectionality of the rewriting grammars is connected to "opaque" rules which cannot be reversed (cf. Kiparsky 1973, Eliasson 1975, Joshi & Kiparsky 1979). The two-level formalism is neutral with respect to production and analysis because it describes morphological phenomena as relations between lexical and surface representations. The relations are seen as correspondences, not as segments being transformed into other segments.

In processually unidirectional frameworks the processes of recognition and production are distinct (cf. Schlesinger 1977, chapter 6). Transformational psycholinguists seem to accept the separation of these two processes, which is more or less a consequence of the formalism rather than an evident goal as such (cf. Fodor, Bever & Garrett 1974). The two-level model succeeds in unifying the processes of word form production and recognition. This is possible because each two-level rule ultimately corresponds to a finite state automaton, and there exists an efficient algorithm using these automata for both production and analysis of word forms. These and various other aspects of the interpretation of the two-level model are discussed in chapter 5.

1.3 Simplicity and reality

The two-level model shares many of the goals of concrete morphology but uses a different formalism, thus enabling a more detailed commitment to the dynamic processes of recognition and production. The restricted power of the two-level rules encourages the linguist to build descriptions with realistic internal representations (as opposed to the representations in abstract phonology). The representations of the two-level model are open to external evaluation. The structure of the two-level lexicon avoids the need of storing multiple entries of inflected words without resorting to phonologically invariant morphemes. Furthermore, the two-level lexicon permits real time incremental access.

The computational complexity of the two-level model is minimal, it relies mainly on small finite state automata. In this way it pro-

vides an estimate of the complexity of the "real" morphological and morphophonological processes.

1.4 Describing Finnish

Finnish is not among the most investigated languages in computational linguistics. Nevertheless, it is a good testing ground because it is well documented and quite different from most better known Indo-European languages. It differs from the Germanic languages i.a. by having a rich inflectional and derivational morphology, and a loose word order. Inflection is rich with regard both to the number of forms and the morphophonological variations. Thousands of distinct word forms can be produced from a single lexical entry, and the morphophonology is intricate. Thus it is reasonable to claim that if a morphological model works when tested on Finnish, it should work for most other languages as well.

Lexicons in natural language understanding systems have usually been small because the systems have been designed for narrow domains, microworlds, or the like. Our goal has been different: to analyze unrestricted running text. This task stresses the need for capturing all the regularities of the inflectional and derivational systems. Because even the exceptional words have thousands of forms, they must be described within the same framework as the regular ones. Exceptional words are not utterly deviant, in all except one or two respects each of them conforms to the regular patterns.

Finnish morphology has been thoroughly described and interpreted, and this work does not attempt to make significant revisions of existing studies. Chapter 3 presents a two-level description of Finnish inflectional morphology covering practically all inflectional types of Finnish verbs, nouns, and adjectives. The main purpose of the description is to show that the two-level formalism is feasible for the description of complex morphologies. The entries of the Reverse Dictionary of Modern Standard Finnish (Tuomi 1972) have been made operational by converting them to the format of the two-level model.

In addition to being the target of the two-level description, Finnish is used throughout this book in examples of the formalism and its concepts. The morphological examples are briefly explicated and English glosses are given. Readers not familiar with Finnish morphology and wishing a more comprehensive description should consult F. Karlsson (1979, 1983a).

1.5 Computational goals

The program implementing the two-level model is interpreted as being part of the model. There is no standard way to present a program as a model and this is, in fact, a problematic venture. Substantial effort must be spent in relating the program to the definitions of the formalism. A bridge between mathematical and linguistic formalisms, and operational computer programs is, however, necessary for making proper use of computational models.

The separation of the description from the program code is generally considered desirable. In this case the same program is intended to be used for processing several distinct languages and therefore the need for separation is obvious. The unification of the recognition and production processes requires the use of the same description and even the same program for both tasks.

Efficiency is one important goal. It is needed in processing large amounts of text as well as in interactive applications. Efficiency in processing and in memory utilization is, however, a secondary goal which should not compromise the others. Given the rapid development of computing hardware, it is sufficient if the computation is efficient enough not to be prohibitive. The program implementing the two-level model is described in chapter 4.

1.6 Previous work on computational morphology

The simplicity of English word inflection has restricted the interest in theoretical research in computational morphology. In English systems there are usually very few true morphological rules, and most of the material is simply listed in the lexicon as such (cf. Winograd 1983, pp. 544-549). This disregard is reflected in natural language processing systems where there may be language independent formalisms and modules for syntax and semantics while the morphological modules are entirely language specific and sometimes even ad hoc.

Most early approaches to morphological analysis can be characterized as "stripping off the endings" or "stemming" (cf. Lovins 1969, Chapin & Norton 1968). Information retrieval was aided by a few crude rules reducing the shape of word forms in order to eliminate the variation due to inflection. Such algorithms performed a "many-to-one" mapping, i.e. distinct words might be reduced into identical "stems".

These methods still have practical use in some applications. The morphological structure of many languages (such as Finnish) may, however, result in unacceptable levels of overlapping "stems".

The inflectional analysis in the Swedish frequency dictionary (Allén 1970, 1971) was done by Hellberg (1971, 1972). The deduction of the base form and the association of inflected forms to it were both accomplished without use of a dictionary. The word forms to be analyzed were first alphabetized, after which groups of successive forms were compared. Maximal sets with consistent endings were deduced to be inflected forms of a single entry word. This approach led to a reasonable percentage of correct groupings and deduced base forms (95 % and 85 %, respectively, of the distinct word forms). Some forms remained unanalyzed and some received an incorrect analysis which had to be detected by proofreading and corrected manually. Similar methods have been used for French texts by Meunier, Boisvert, and Denis (1976), and for German morphology by Hann (1974, 1975). These methods work best in languages where there are few stem alternations. In favourable circumstances, all inflected word forms of an entry word occur consecutively in the alphabetic order.

Brodda and Karlsson (1980) have designed a system for segmenting inflectional affixes in running Finnish text. It is based on the exploitation of redundancy in the graphemic shapes of word forms. The word forms are analyzed in isolation and the algorithm gives a correct segmentation in about 89 % of the instances. The algorithm finds the boundaries between various endings but does not attempt to deduce the underlying base form in instances where the stem is subject to variations.

Such schemes of ending removal may be combined with a dictionary lookup. If the dictionary contains morphological information, further rules may check whether the word forms are correctly inflected forms of existing word entries (Kay & Martins 1970). Hand compiled inverted inflectional systems may be quite accurate, but they tend to be difficult to master. Small changes in the rules intended as improvements may have unanticipated negative side effects.

It is easier to approach inflection if one focuses on word entries and describes how the words in question are inflected. The affixes can be presented in a more familiar shape and the alternations within stems can be described independently without constant reference to other substrings which happen to be similar. One early system of this type was that of Wolfart and Pardo (1973). The system was designed for the compilation of a dictionary for a corpus of Cree texts. The inflectional rules were revised and checked in parallel with the

establishing of the word entries. Later, Latin inflection was implemented on the same system (Wolfart 1979). The formalism was, however, not general enough to handle e.g. Finnish morphology (especially phenomena such as consonant gradation). Sågwall presented a similar system in her dissertation (1973) but her program was more language-specific. It had routines for the morphophonological alternations in Russian inflection and a dictionary of word entries with sufficient morphological information for checking the correctness of possible analyses.

Recently, there has been a growing interest in more general approaches to morphological analysis. Sågwall (1978, 1980) has investigated the applicability of Kay's general chart parsing methods (Kay 1975) to Finnish inflection. L. Karttunen (1981) has designed a system called TEXFIN for Finnish morphological analysis. The program is capable both of analyzing and of producing word forms. It is based on general principles applicable to other languages as well, although the morphotactic structure is presently part of the program code. TEXFIN relies on the concept of linked minilexicons, an idea also used in the present two-level model. TEXFIN has only a minimal rule component.

Källgren has applied Brodda's general purpose rewriting system BETA to Finnish consonant gradation (1981) and to the morphological analysis and synthesis of Finnish nouns (1983). The latter piece of work resembles TEXFIN in showing that the complexity of Finnish morphology is not beyond the power of regular grammars or finite-state automata. Källgren's system is furthermore bidirectional in the sense that although she has distinct programs for analysis and synthesis, the programs are very similar and systematically related. Fleck (1982) has studied the problem of morphological analysis including the segmentation of words from a continous input, which is very difficult or impossible for most morphological systems. As the basis for a design of a general algorithm she has used the morphologically complex Mayan language Tzotzil.

The most interesting recent approach has been taken by Kaplan and Kay (1981). They have designed a compiler for the formalism of generative phonology. The morphological description of a language is written in this formalism and the compiler compiles each rule into a finite state automaton. The automata form a chain, and if they all are combined into a single automaton, it will execute all phonological variations by comparing the lexical and the surface forms. The same unified automaton would both analyze and produce word forms. The only open question concerns the actual size of the resulting unified automata which may grow prohibitively large.

The two-level model has been greatly influenced by the Kaplan and Kay model. Many good qualities have been inherited from their model, especially the bidirectional functioning and real-time execution. Although the two-level model resembles their system by using finite-state automata for comparing lexical and surface representations, the two approaches have significant differences. Especially the use of parallel rules is new. It relieves us from the necessity of combining all automata into a single one, and thus leads to an implementation which is computationally very efficient. The most significant impact of the parallellism is, however, that it forces one to reject the generative formalism and adopt a different theoretical framework.

1.7 Two-level rules

Morphology is just one linguistic subsystem. A full model of a language would contain several other subsystems, in particular semantics, syntax, and phonetics. The two-level model deals only with two representations of the whole system: the lexicon and the phonemic surface representations. The lexicon contains the morphophonological representations of word entries and endings. The phonemic surface level consists of phonemes, or letters of a phonemic alphabet.

The essential difference between the two-level model and generative phonology is that in the former, there are no intermediate stages between the lexical and the phonemic representations. Instead, the two representations are directly related to each other. The relation is formulated with parallel rules which may refer to both of these two representations. Mathematically speaking, individual rules may be regarded as equations, and all rules together as a set of equations. The rules "do" nothing as such, they only test whether the correspondence is correct. The actual production or analysis is a separate driver mechanism guided by the rules.

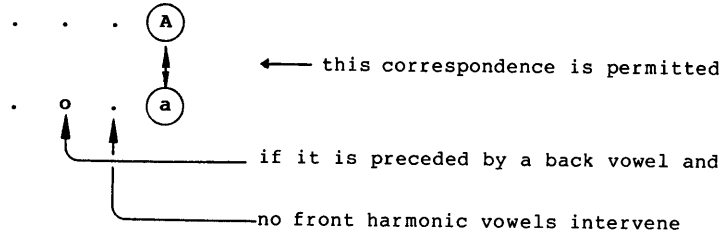
The two levels are considered to be written above each other as two tiers. The upper row consists of lexical representations: word stems and morphotactically permitted endings. The lower row consists of the phonemes of the surface form. The upper and the lower characters correspond to each other pairwise. The most trivial case occurs when the characters of upper and the lowel levels are identical, e.g. (nominative singular of the noun **talo** 'house'):

Lexical level:	**t**	**a**	**l**	**o**
Surface level:	**t**	**a**	**l**	**o**

The primary mechanism of inflection consists of appending the lexical representation of an ending after the stem. The following slightly simplified configuration represents the inflected form **talona** 'as a house' (where **nA** is the representation of the essive singular ending):

Lexical level:	**t a l o n A**
Surface level:	**t a l o n a**

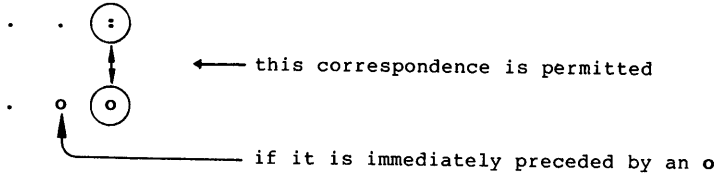
This example shows a correspondence **A** ⟷ **a** which is an instance of vowel harmony. The front/back opposition of the low vowels, **ä** and **a**, is neutralized in Finnish endings. One interpretation is that there are two parallel ending morphs, here **na** and **nä**, one of which is selected to satisfy the requirements of vowel harmony. The above example follows the other possibility of positing an archiphonemic representation **nA**. The two-level rule defines the correspondence and its context condition, schematically:



The illative singular form **taloon** 'to the house' demonstrates vowel doubling. The illative ending is represented as **:n**, where the colon marks the repetition of the preceding vowel. The configuration is:

Lexical level:	**t a l o : n**
Surface level:	**t a l o o n**

The vowel doubling rule might be visualized as:



The two-level model allows context conditions to refer to either or both of the levels. Phonological conditioning is preferably defined by reference to the surface. The surface context has the obvious advantage of being less subject to the difficulties encountered in

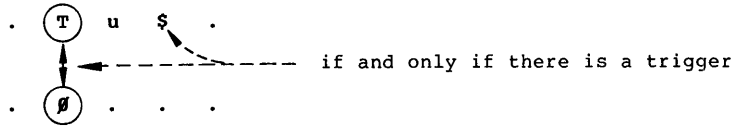
selecting more abstract lexical representations.

Some alternations cannot, however, be accounted for by the phonological surface context alone. These are described by using morphological features, which are special characters with no phonological substance. Features always correspond to zero (ø) on the surface and they denote grammatical boundaries or identify (sub)classes of endings which need to be distinguished for morphological reasons. They are used as explicit triggers for morphologically conditioned alternations.

E.g. Finnish consonant gradation is easiest to describe with a triggering feature (from the synchronic point of view) (see section 3.2.8). This is an alternation between strong grades (e.g. -tt-) and weak grades (e.g. -t-). The choice of the grade depends mainly on the structure and morphological class of the ending. Thus, endings requiring the weak grade may be marked with a feature ($). One of these endings is the genitive singular. Its lexical representation is **$n**. The configurations for the nominative and genitive singular forms of **kettu** 'fox' are:

Lexical level:	**k**	**e**	**t**	**T**	**u**		
Surface level:	**k**	**e**	**t**	**t**	**u**		
Lexical level:	**k**	**e**	**t**	**T**	**u**	**$**	**n**
Surface level:	**k**	**e**	**t**	**ø**	**u**	**ø**	**n**

The absence of the trigger permits only the correspondence **T** ⟷ **t** and its presence only **T** ⟷ **ø**. The triggering can be represented schematically as:



On the basis of this example it is seen that many phonological and morphophonological phenomena can be described within the framework of two representations, one above the other. The availability of both levels enables us to describe fairly complex morphophonological phenomena. Even instances where two adjacent segments both vary are usually easy to describe because either one or both of the decisive environments can be stated in terms of the surface. The descriptive power of this framework is probably greater than that of a phonological rewriting system without rule ordering.

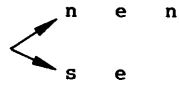
1.8 Alternation patterns

All morphological alternations are, of course, not simple enough to be characterised as correspondences of single segments. Some may have a phonological origin, but in a synchronic description they are no more decomposable into active variations of single segments (cf. the indivisible morphological operations of Linell 1976, 1979).

Complex alternations need not be isolated exceptions. They may be parts of quite common and productive inflectional types. E.g. all Finnish nouns ending in **-nen** are subject to a suppletion-like **nen-se** alternation. This word type is both common (more than 7000 entries in the reverse dictionary) and productive e.g. by being a part of several derivational suffixes such as **-lainen** 'inhabitant of' and **-mainen** 'like'. The nominative singular of these words ends in **-nen** and the inflected singular stem in **-se**, e.g. **hevonen** 'horse':

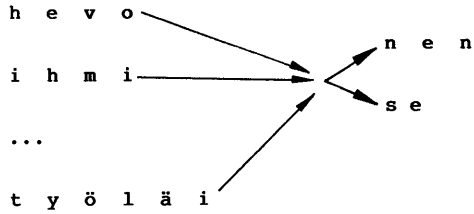
hevonen	nominative singular
hevosen	genitive singular 'of a horse'

There are no rules in present-day Finnish relating the independent segments **n** and **s** in such an environment. To avoid nonnatural ad hoc rules for transforming sequences of phonemes into other phonemes we use so called alternation patterns. An alternation pattern is essentially a mini-lexicon which contains the representations of the alternatives, schematically (and simplified, cf. 3.1.8):



An alternation pattern gives the alternatives as such, without relating them morphophonologically. This mechanism can thus represent even those alternations where the alternatives have different origins and there is no phonological relation. Most often such patterns are, however, used where the relation is recognizable but no more systematic.

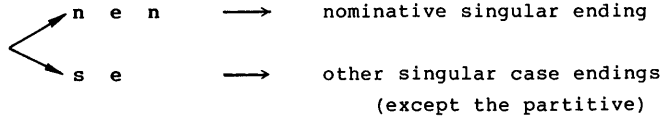
An alternation pattern is usually common to a whole inflectional type of words, e.g.:



The root entries of the above words contain only the constant part of the nominative form, the variable part at the end is truncated. The lexicon system of the two-level model needs only one instance of each alternation pattern. All nouns of the same inflectional type refer to the same pattern. This mechanism captures a generalization just as rules or operations do, but conceptually it is a part of the lexicon. No actions are implied, only references.

The reference mechanism combined with the alternation patterns takes a position in the debate whether invariant base forms, selections of stems, or full series of endings are stored in the lexicon. In a sense, all inflected forms <u>are</u> in the two-level lexicon because they can be retrieved by following the linkages. On the other hand, the linked structure avoids the excessive repetition which arises from the simpler view that the lexicon is a <u>list</u> of entries. (Cf. Vennemann 1974, Linell 1976.)

The linkage mechanism is extensively used in the lexicon system. Each entry, whether it is a word root or an alternative in a pattern, has a specific continuation pattern. Simple word entries like **talo** and **ketTu** refer to the whole set of nominal case endings as their continuation. Alternatives in patterns refer to subsets of case endings, e.g. (again, simplified):



Word roots, pattern alternatives, and various classes of endings are grouped into lexicons. Continuation linkages may be defined by referring to these (mini)lexicons.

1.9 A classification of alternations

There are several kinds of morphological alternations, which may need different kinds of methods for their description. The following two-way classification will be useful when one compiles a two-level description and selects the proper formalism for each variation:

Variation

(1) (Fairly natural) single segment modifications: alternations between two or more phonologically closely related single phonemes. E.g. "plural **i**" is manifested in surface forms usually as **i**, but as **j** between vowels.

(2) Suppletion-like nonnatural alternations: alternations between either longer sequences of phonemes (e.g. the **nen-se** alternation pattern) or phonologically unrelated (or remotely related) single segments (e.g. **a-t** in the stem formation of contracted verbs such as **hakkaa - hakatkaamme** 'he beats - let's beat!').

Triggering

(A) Phonologically conditioned: the variation is triggered by the plain phonological context. The context may be local (referring to the neighbouring phonemes) or non-local (referring e.g. to the syllable structure).

(B) Morphologically conditioned: The phonological surface representation does not account for the variation. Alternants occur e.g. in connection with certain classes of endings (junctures or features are referred to).

It is easy to find examples of all four combinations of these two criteria in Finnish morphology:

(1&A) Vowel doubling is a typical, local, phonologically conditioned variation. The variation is limited to a single segment, the nature of which is determined by the immediately preceding vowel. Vowel harmony is an example of a similar variation between phonologically

related segments, but here the triggering phonological context is nonlocal. Assimilations typically belong to type (1&A).

(1&B) This constitutes a large group in Finnish morphology. Most vowel alternations at the end of the stem affect single closely related vowels, but the alternations occur only in this position. The trigger is e.g. the plural or past tense **i** (other **i**'s do not necessarily have the same effect). Examples: **i-e** in **risti-ristejä** 'cross sg.- crosses partitive pl.', **a-o** in **kala-kaloja** 'fish sg. - fish partitive pl.'. Consonant gradation is an alternation affecting single consonants, but the context condition is only partially phonological (and is here described with an explicit trigger). Examples: **t-d** in **keitaan-keidas** 'of an oasis - an oasis', and in **jahtaat-jahdataan** 'you hunt - someone hunts'.

(2&A) The selection of allomorphs for the endings of the partitive and illative singular and plural, and the genitive plural cases involves a choice between phonologically unrelated morphs, e.g. **maitten** 'of countries', **lasien** 'of glasses', **kielten** 'of languages'. The alternatives are given in the lexicon as separate formally unrelated entries. The choice is made by rules referring to the syllable structure of the word stem.

(2&B) The alternation patterns at the end of stems consist, by definition, of phonologically unrelated alternatives. The choice among them is not based exclusively on phonological criteria or on morphological features, e.g. **vapaus - vapauden - vapauksien** 'freedom - of the freedom - of freedoms'. Rather, it depends on the combinatorics of various subclasses of endings, here **s** ⟷ nominative, **de** or **te** ⟷ other singular endings, **ks** ⟷ plural endings.

This classification has a direct connection to the two-level formalism. The method of describing an alternation is selected according to these distinctions. Types (1&A) and (1&B) are usually handled with rules (cf. section 3.2). The context conditions for rules of type (1&A) refer to the surface only, whereas (1&B) rules refer also to the

features in the lexical representations. In both types the alternation is handled with a single lexical representation. Variations of type (2&A) and (2&B) need multiple entries in the lexicon. (2&B) is described in the lexicon with the continuation linkages. Alternations (2&A) are accompanied by rules for checking the proper syllable structure of the word stem.

The classification necessarily has some tolerance. Although many alternations clearly belong only to one class, there are others where two classifications would be possible. This kind of redundancy is characteristic of the two-level model. Being a model of dynamic processes of word form recognition and production, it does not try to cover all redundancies.

CHAPTER 2
A FORMALISM FOR TWO-LEVEL DESCRIPTIONS

The morphological two-level model consists of two major components, a lexicon system and two-level rules. The components are based on a common alphabet. They are interdependent and together they form a complete description of word inflection. The lexicon system lists, of course, the morphemes of the language, but in this model it also restricts the feasible sequences of morphemes within a word. The two-level rules specify the permissible (or obligatory) discrepancies between lexically stored representations and surface forms. The rules have the form of conditions, but the rule component as a whole can also be regarded as a filter through which the real word "sees" the lexical representations.

The two-level formalism is bidirectional in the sense that morphological descriptions may be read both in the direction of analysis and production, since the rules are conditions rather than actions. The two-level model resembles concrete morphology in paying great attention to the surface forms, which, in this model, are as important as the lexical representations. Compared to generative models, the two-level formalism assigns rules a more limited role, while making wider use of the lexicon system (for a fairly recent discussion on the roles of the lexicon see e.g. Farkas & al. (eds.) 1978).

2.1 THE ALPHABET

Both the lexical and the surface levels in the two-level model are strings of characters in an alphabet S. The upper string, which is a concatenation of lexical entries (e.g. roots and endings), may contain some characters which never occur on the surface. To emphasize the special roles of some characters, the alphabet is partitioned into three subsets:

S_S The set of surface characters, which are the only ones allowed in real surface word forms. By default, all surface characters may also occur in the lexical representations.

S_m The set of archiphonemes and morphophonemes. These segments are used for describing (morpho)phonological alternations. They are only used in the lexical representations and they never occur on the surface as such. Rules realize them as surface characters (or as zero).

S_f The set of morphological features, which represent morphological properties and boundaries. They act as triggering conditions for morphophonological rules. These symbols always correspond to zero on the surface.

The definition of the alphabet depends on the scope of the study. If our perspective is phonetic, the surface consists of phones and we are bound to have a large alphabet. In phonological studies the surface consists of phonemes, and if the target is plain written language the surface characters are letters (or graphemes).

In the following we shall take examples of these sets from a description of written Finnish (details will be given in chapter 3). We use the ordinary set theoretical notation for expressing these sets. E.g. the surface characters in the Finnish description are the following:

$$S_s = \{ \text{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o,}$$
$$\text{p, q, r, s, t, u, v, w, x, y, z, å, ä, ö, ' } \}$$

The letters **b, c, f, q, w, x, z** and å are "foreign" and do not occur in native Finnish words. They are included in the surface characters because loan words and proper names contain them. The apostrophe is used in the orthography for certain zero realizations of gradating **k** (cf. rule 8b in section 3.2.8).

Archiphonemes and morphophonemes are used as variants of surface characters in the description of certain variations involving one segment. Technically, all of these are very similar to surface characters, with the exception that archiphonemes and morphophonemes do not occur on the surface. The term "archiphoneme" is used here for those alternations that represent phonologically conditioned neutralizations of oppositions (in the sense of Trubetzkoy 1939, p. 71). The term "morphophoneme" is here used in conjunction with morphologically conditioned alternations involving a single segment.

It turns out that it is more useful to classify alternations as morphophonemic or phonotactic rather than to stress the status of individual segments participating in such alternations. An alternation is considered morphophonological if it is morphologically conditioned

or more radical than a mere phonological neutralization. The lexical segment corresponding to a morphophonemic alternation can be either a surface character or a morphophoneme. E.g. the productive type of Finnish nouns ending in **i** has a morphophonemic **i-i-e** alternation:

lasi	'glass'
lasissa	'in (a) glass'
laseissa	'in glasses'

Because of the productivity of this type, a lexical representation **lasi** is used. There is, however, a smaller, closed type of nouns with a different, but also morphophonemic alternation **i-e-ø**:

lovi	'hole'
lovessa	'in (a) hole'
lovøissa	'in holes'

A different lexical representation must be used for these word, e.g. **lovE**. The difference between **i** and **E** in the lexical representations serves thus only to distinguish between these two (morphophonological) alternation types.

When we use archiphonemes and morphophonemes in lexical representations they correspond to the interpretation that we have a single lexical form with an isolated alternation where the rest of the form is constant. This approach should, however, be taken only in clear cases, and when it results in an overall simplification. Single invariant underlying lexical representations should not be forced if the alternatives are remote (one should then use the means of the lexicon system).

It is customary to use capital letters for archiphonemes and morphophonemes, and this practice is followed here, except for the colon (:) which is used for vowel doubling (to distinguish it from the set of vowels). In the description of Finnish we use the following set:

$$S_m = \{ \mathbf{A}, \mathbf{O}, \mathbf{U}, \mathbf{:}, \mathbf{E}, \mathbf{Ä}, \mathbf{Ä}\!\!\!\!\!\!\cdot, \mathbf{K}, \mathbf{P}, \mathbf{T}, \mathbf{D}, \mathbf{Z} \}$$

The first three segments represent archiphonemes in endings, where the front/back opposition between vowel harmony pairs is neutralized. In the lexical representation of endings, **A**, **O**, **U** are used where there is an **a** or **ä**, **o** or **ö**, **y** or **u** on the surface in accordance with vowel harmony. The vowel morphophonemes **E**, **Ä** and **Ä**˙ are used for certain stem final vowel alternations. The morphophonemes **K**, **P** and **T** are used for

consonant gradation. The morphophonemes **D** and **Z** are used in endings for infinitives and passives. (See the corresponding sections in chapter 3.)

Morphological features differ from the other types of characters: they do not represent any phonetic substance at all. Technically they are treated like other segments, but their interpretation and use is different. They represent certain morphological properties of lexical units or denote junctures. The characters for features appear as parts of lexical representations but they correspond only to zero on the surface. Boundary features mark the presence of certain classes of endings. These and other features may be used as triggers for certain variations, where the phonological environment alone is not enough, and explicit morpholexical conditioning is needed.

Features may also be used as distributional selectors. There are instances where the feasibility of affixing an ending allomorph cannot be accounted for by phonotactic criteria alone. In Finnish at least, the distribution of some ending allomorphs can be defined according to the phonological 'gestalt' of the preceding stem, but not according to phonotactic constraints proper. E.g. for the plural illative, there are three ending allomorphs: **iin, ihin, isiin** (cf. 3.2.9). The only acceptable illative plural of **valo** 'light' is **valoihin**. The form **valoisiin** is not understood as a form of **valo**. In fact, it is the plural illative of another word **valoisa**, and the form is segmented **valois**<u>**iin**</u>. Thus the selectional criteria cannot be purely phonotactic.

Sensitive endings may be marked with a selector feature. A rule corresponding to this feature may refer to the 'gestalt' of the stem and discard unacceptable combinations (cf. section 3.2.9). The description for Finnish uses the following set of features:

$$S_f = \{ +, /, \_, (,), @, *, \#, \$, ., 1, 2, 3, 4, 5, 6, 7, 8, 9, !, \%, \& \}$$

The first eight (**+** through **#**) signal morpheme boundaries and mark certain classes of endings: **+** is a boundary in front of case endings, **/** in front of possessive suffixes, and **_** in front of clitics. **#** denotes a word boundary, ***** marks passive endings, and **@** comparatives. The period (**.**) is used as an explicit syllable boundary between vowels. **$** denotes a triggering morphological feature for consonant gradation, and the twelve last ones (**1** through **&**) are selector features for certain endings.

We may define any useful subsets of the alphabet for our convenience. The Finnish description uses the following subsets (vowels,

consonants, back harmonizing vowels, front harmonizing vowels and liquids):

V = {a, e, i, o, u, y, å, a, ö, A, O, U, E, Å, Ä}
C = {b, c, d, f, g, h, j, k, l, m, n, p, q, r, s, t, v, w, x, z, K, P, T}
Vb = {a, o, u, Å}
Vf = {ä, ö, y, Ä}
Cl = {l, r}

We define a special symbol, the null, to be used when some character on the lexical level corresponds to nothing on the surface. We have chosen the character zero (0) to symbolize this null. (The reader should take care to distinguish the various nulls and zeroes used in this work: null character, the empty set {}, the null sequence <>, and the null string.)

2.2 THE LEXICON SYSTEM

The task of the lexicon system is to define the lexical representations of word-forms. These consist of the representations for the stems and various endings in combinations defined by morphotactic structure. The approach taken here is a restricted one: it is based on the concept of simple continuation linkages. It has a minimal expressive power, but it seems powerful enough to cover the morphotactic structure of many languages. Only a small residue of structures needs apparently unnecessary duplicate descriptions or forces one to resort to the use of rules and features for morphotactics (cf. Karttunen & Wittenburg 1983, Alam 1983, and Khan 1983). Only restricted infixation and reduplication can be handled adequately with the present system. Some extensions or revisions will be necessary for an adequate description of languages possessing extensive infixation or reduplication.

The present lexicon system of the two-level model consists of a set of (sub)lexicons and a set of continuation classes. All lexicons are formally similar, although they are used for storing various kinds of entries:

(1) The actual entries for all root lexemes in the lexicon. Each entry contains both the phonological representation of the

word stem and syntactic or semantic information that has been stored with the lexeme.

(2) Suppletion-like alternation patterns at the end of words in certain inflectional classes, e.g. the **nen-se** alternation in **hevonen - hevosen**, which is common to all Finnish nouns ending in **-nen**.

(3) Inflectional and derivational endings, e.g. for number, case, voice, tense, mood, and person. Fully analyzed endings consisting of a single morph as well as portmanteau morphs may be used.

All lexical entries must be stored in some of the sublexicons. The grouping of entries into the lexicons is usually determined by the combinatory characteristics of the endings. Repetition of entries is here considered more harmful than partitioning of series of endings into smaller sublexicons. Continuation classes (which are described below) refer to collections of sublexicons, and these are sometimes linguistically more meaningful than the individual sublexicons.

2.2.1 Lexical entries

All lexicons have an identical format. The lexicon has a name and a list of entries, e.g.:

```
LEXICON S3
    1+tA      P    "PTV SG";
    2+ten     P    "GEN PL"
```

This (sub)lexicon has a name "S3" and it consists of two entries, here endings that follow certain stems of Finnish nouns. Each entry consists of three items.

(1) The lexical representation of the entry, e.g. **1+tA**, which is a sequence of four characters in our alphabet. The first character is a selector feature, and the second is a juncture. The third segment is an ordinary surface character, and the fourth is an archiphoneme.

(2) A reference to a continuation class. P is defined elsewhere as a class which states that the entry may be followed either by a possessive suffix, a clitic, or a word boundary.

(3) The information stored in the entry. It will be retrieved when the entry is recognized as part of a word form. The kind of information needed depends on the environment where the morphological analyzer works. In a syntactic or semantic parser appropriate information must be stored. Here we shall be satisfied with the minimum of information which shows the correctness of the analyses by giving crude glosses and mnemonic morphosyntactic features.

The lexical representations contain the substance which may appear on the lexical level of our model, whereas the continuation class mechanism defines the morphotactic structure of word-forms. In the present lexicon system the information associated with the lexical entries does not affect the function of the two-level model. The morphosyntactic (and other) features might be quite useful in defining certain morphotactic structures, which are difficult to handle with the linkage method alone (cf. Alam 1983).

2.2.2 Continuation classes

Each continuation class has a name and it gives the list of sublexicons, the members of which may follow. There is a special continuation symbol # for allowing the entry to be at the very end of a word form. The following are examples of continuation classes in the description of Finnish.

```
( S123 = S1 S2 S3 )
( P = P K0 # )
```

The symbols "S123" and "P" are the names of the classes "S1", "S2" and "S3" and, correspondingly, the second "P" and "K0" are names of sublexicons the entries of which are the possible continuations. The S's are lexicons for subsets of case endings, the lexicon "P" contains possessive suffixes and "K0" clitics. (The position determines which "P" is which in these definitions, so the same names may be used for both, if it improves the readability of the description.)

2.3 TWO-LEVEL RULES

The task of the two-level rules is to account for all discrepancies between lexical representations and the surface word forms. Some phonemes in the lexical representation may not be realized as such on the surface, and for each morphophoneme in the lexical representation a choice must be made between its possible realizations. The variations may be phonotactic or morphophonological in nature, but they all pertain to single segments in the lexical representations.

In addition to defining the correspondence between the levels, the rules may be used for forbidding certain combinations of lexical units. The task of the lexicon system was to define morphotactically feasible sequences. Some combinatory rules may, however, be more phonological in their nature than what can be neatly expressed by lexical means. In such instances one may use rules for excluding phonotactically or otherwise invalid combinations.

2.3.1 Two-level configurations

The two-level rules differ fundamentally from generative rewriting rules. Instead of rewriting, they restrict the possible correspondences between lexical representations and surface forms. The presence of the two-level rules guides the production and the analysis of word forms, but the rules are not capable of analyzing or producing as such. The rules can be best understood if we think that we have both the lexical representation and the surface form available simultaneously. These two forms must be aligned by adding null characters, where necessary, so that the corresponding characters are at the same positions, e.g.:

Lexical level:	t	a	l	o	+	i	A
Surface level:	t	a	l	o	Ø	j	a

Synchronized strings like this are called configurations, and the two-level model interpretes them as strings of corresponding character pairs.

The rule component may also be interpreted as a slightly distorting filter between the upper and lower strings of a configuration. Lexical representations seen through the filter would look like suface forms. If we go inside and look through the same filter in the oppo-

site direction, the surface forms would look like the lexical representations. (There might be multiple images through the filter but one of them is identical to the string in front of the filter.) Although this is a useful metaphor, we shall, in the following, operate mostly with strings of pairs and various expressions based on them.

2.3.2 Character pair expressions

The two-level rules are defined by sets of permitted configurations. When all rules are simultaneously satisfied they leave very little room for possible configurations, and this set actually defines our "distorting filter". To define the permitted configurations for the rules, we have to elaborate a formalism for character pair expressions, which we call regular pair expression notation (RPE). This closely resembles the well-known concept of regular expressions (see e.g. Booth 1967, chapter VI, or Kain 1972, pp. 49 ff). Deviations from regular expressions are motivated by two concerns: to abbreviate the pairs with single symbols where possible, and to maintain some similarity to standard linguistic notations.

In the following we start from elementary concepts and proceed to more complex ones. First we define concrete pairs as pairs of lexical and surface characters (including ∅) which may correspond to each other in at least some environment, e.g.:

s	i	K	+
s	j	k	∅

In these both components are single characters, the lower ones must be surface characters or the null character, and the upper one may be any character of our alphabet (including morphophonemes and morphological features).

The rules in the two-level description are not quite independent of each other. All rules together list the set of all possible concrete pairs as they occur in some parts of the rules. We shall need this concrete pair set (or CPS) in the definition of more general pair expressions.

With the CPS we may define more general pairs, so called abstract pairs, where we use set symbols instead of concrete characters. Below, X and Y stand for arbitrary subsets of the alphabet (which includes the null character). The abstract pair is defined as:

$$\begin{matrix} X \\ Y \end{matrix} = \left\{ \begin{matrix} x \\ y \end{matrix} \text{ IN CPS } . \; x \text{ IN } X \text{ and } y \text{ IN } Y \right\}$$

The abstract pair is thus a subset of the concrete pair set. It consists of those pairs where the upper symbol belongs to X and the lower symbol to Y.

We extend the notion of abstract pairs to cases where either X or Y is a single character q by letting the character represent the set {q}. In this way we may use semiabstract pairs, where one of the sets is replaced by a single character, e.g.:

$$\begin{matrix} : \\ \mathbf{V} \end{matrix}$$

We introduce an alternative symbol (=) for the set of all characters, i.e. the set consisting of S plus the null character. Thus we may denote a surface **a** by:

$$\begin{matrix} = \\ \mathbf{a} \end{matrix}$$

This (semi)abstract pair covers any surface **a** no matter what it corresponds to on the lexical level. It must be stressed that abstract pairs do not introduce any new possibilities into the correspondences. They are just subsets of the CPS. Thus the above pair refers only to those pairs of vowels corresponding to each other which are explicitly mentioned elsewhere. In any reasonable description it would <u>not</u> cover such arbitrary pairs as $\begin{matrix} \mathbf{f} \\ \mathbf{s} \end{matrix}$ or $\begin{matrix} \mathbf{s} \\ \mathbf{a} \end{matrix}$.

Often the upper and the lower symbols in a pair are identical. Then we may omit one symbol and write the other halfway between the upper and the lower line. Furthermore, the signs for morphological features always correspond to zero, and we write the sign as such to represent the pair. With these notations we gain more readable formulas like:

$$\begin{matrix} \mathbf{V} & \mathbf{s} & + \end{matrix}$$

instead of

$$\begin{matrix} \mathbf{V} & \mathbf{s} & + \\ \mathbf{V} & \mathbf{s} & \emptyset \end{matrix}$$

We denote the complement of a set X (with respect to the alphabet including zero), by -X. The difference of two sets is denoted by X - Y as usual. Complements and differences may be used in denoting sets and abstract pairs, as well as in the actual RPE formalism. E.g. an abstract pair $\begin{matrix} \mathbf{K} \\ \mathbf{k} \end{matrix}$ stands for all concrete pairs in the CPS, where the lexical character is **K**, and the surface character something other than **k**.

2.3.3 Pair strings

Context expressions often need to refer to environments wider than just a single pair. Therefore we must be able to concatenate elements and move over from sets of pairs to strings of pairs and sets of such strings. If we take the CPS as the alphabet for strings, we can use the common concept of strings consisting of zero, one, or more elements of CPS in a sequence. In order to make full use of the previous notion of abstract pairs as subsets of the CPS, we have to proceed to the concept of sets of pair strings, i.e. regular pair expressions (RPE).

Each concrete pair q in the CPS is a RPE, and it represents a set consisting of one string whose length is one and the only symbol is the concrete pair q. An abstract pair is also a RPE and it represents the union of all its individual pairs. Thus e.g. $\overset{s}{s}$ is a RPE consisting of single string (which consists of one pair): $\overset{s}{s}$.

To construct sets containing strings longer than a single element, we simply write constructs of the above type one after another, e.g.:

$$\overset{=}{\mathbf{v}}\ \overset{=}{\mathbf{v}}$$

This refers to a diphthong or double vowel on the surface. As a RPE it denotes a set of pair strings where each of the strings is two pairs long. Both elements in these strings are pairs in CPS, where the lower component belongs to the set V (the vowels). Generally, if we concatenate two sets of strings, the result is the set containing all combinations of concatenations where the first part is from the first set and the other part from the second set. Thus if we have two RPEs P and Q, the concatenation (or product) is defined as:

$$P\ Q = \{\ p\ q\ .\ p\ \text{IN}\ P\ \ \&\ \ q\ \text{IN}\ Q\ \}$$

Starting with strings of one symbol, the concatenation produces strings of any length by recursive application.

2.3.4 Alternative expressions

So far we have defined sets of pair strings, where the members are one element long, and concatenations of such sets. All useful sets of pairs cannot be expressed like this, because we may want to leave out some cross combinations. We denote an alternative expression by

separating the alternatives with vertical bars and by enclosing the whole expression in square brackets, e.g.:

$$[\ \overset{=}{\mathbf{V}}\ |\ \mathbf{h}\ ]$$

Such expressions are just set theoretical unions of the alternatives, thus containing those strings that belong to at least one of the alternative sets. The square brackets are not necessary around the alternatives, they are used there and elsewhere for clarity and disambiguation.

2.3.5 Optional parts

According to standard conventions in generative linguistics, we enclose optional parts in the RPEs in parentheses. E.g. the expression:

$$\overset{=}{\mathbf{a}}\quad (\mathbf{h})$$

denotes strings of pairs consisting of either a single surface **a** or an **a** followed by a **h**, and is equivalent to the following RPE which uses the notations already defined:

$$[\ \overset{=}{\mathbf{a}}\ \mathbf{h}\ |\ \overset{=}{\mathbf{a}}\ ]$$

2.3.6 Iterative expressions

In some cases we have to refer to the presence of segments at an arbitrary distance from the correspondence pair. In Finnish morphology such references are needed only when we describe vowel harmony and the syllable structure of stems. The frontness or backness of harmonizing ending vowels may be resolved by a vowel anywhere in the preceding part of the word. We use the Kleene star (*) to denote iteration, i.e. zero, one, or more repeated subexpressions. The use of iteration in two-level rules is probably justified only for denoting segments that are ignored by the rule. Vowel harmony rules use e.g. the following kind of expressions:

$$\left(\overset{=}{\mathbf{-Vb}}\right)^{*}$$

This represents the expression in parentheses repeated zero, one or

more times. Note that any expression repeated zero times gives the null string, which is the string of length zero. (The set consisting of only the null string is neutral in concatenation: the result is the same set we start from.)

2.3.7 The components of two-level rules

Two-level rules formally resemble generative rules, although their interpretation and function is quite different. In contrast to the rewriting rules of generative phonology, two-level rules are like filters. The counterpart of a traditional rewriting rule:

$$\mathbf{i} \longrightarrow \mathbf{j} \;/\; \mathbf{V}+ \_\_ \;\mathbf{V}$$

would be the following two-level rule (to be defined in detail in the next section):

$$\begin{matrix}\mathbf{i}\\ \mathbf{j}\end{matrix} \;\langle\Rightarrow\rangle\; \mathbf{V}+ \;\text{—}\; \mathbf{V}$$

The basic structure of two-level rules is:

CP "op" LC — RC

They thus consist of three parts:

(1) The correspondence part (CP), which is a concrete or an abstract character pair whose occurrence is restricted by the rule. In our example **i** above **j** to the left of the operator indicates that the rule deals with (the plural) **i** realizing as a **j** on the surface.

(2) An operator indicating what kind of a relation is stated between the pair in the left part and the context. The operator in the above example states two things simultaneously: $\begin{matrix}\mathbf{i}\\ \mathbf{j}\end{matrix}$ may occur only in this environment; additionally, in this environment lexical **i** may correspond only to **j**.

(3) The context of the correspondence stating the phonological or morphological conditions for the correspondence. The left

(LC) and right (RC) contexts are RPEs, which together with the CP are used in constructing the expression defining the permitted configurations. In the above example the (positive) context condition is that the **i** is in a plural ending and it is between vowels

2.3.8 Elementary rules

Rules are expressed using RPEs, but individual rules can be best interpreted as conditions which focus on correspondences CP and their environments LC — RC. Elementary rules have a single pair as their CP and a single LC — RC. More complex rules will be defined by reducing them into elementary rules. The basic relation between a CP and the context is an implication in either direction. These two directions of implication will define the elementary rule types and their combination a third type.

In the following we use a notation (..) which stands for the expression CPS^*, which denotes the set of all possible configurations, i.e. all sequences consisting of members of CPS. They include even those that will be excluded by the rules, and have lengths 0, 1, 2, etc. This notation will be used especially to indicate that an expression starts or ends with a subexpression, (e.g.: L = .. LC).

The first basic rule type is called context restriction rule. It lists an environment where a given correspondence may occur. In other words, the CP may occur only if it is enclosed in the context LC—RC.

Context restriction rule:

$$CP \;\Rightarrow\; LC \;\_\_\; RC$$

In general, there may be occurrences of CP in the context expressions LC and RC, but they must, in turn, satisfy the same condition. The "only if" condition for each occurrence of CP in a configuration CF is the following:

$$\left[CF = L\ q\ R \;\&\; q \text{ IN } CP\right] \Longrightarrow \left[L \text{ IN } .. \text{ LC} \;\&\; R \text{ IN } RC\ ..\right]$$

The permission given by context restriction rules is tentative. Any rule of the following type may override it.

Rules with the converse implication are called surface coercion

rules. These rules state that for pairs q, which have the same lexical character as CP, the presence of the context LC -- RC implies that there is no other choice for the surface character than the same as in CP:

Surface coercion rule:

$$CP \ <= \ LC \ — \ RC$$

In elementary rules CP consists of a single pair $\frac{x}{Y}$ where x is a single character and Y either a set or a single character. This rule deals with occurrences of pairs $q = \frac{x}{z}$ where z might or might not belong to Y. The rule condition dictates that if q is enclosed in context LC — RC then q must one of those represented by $\frac{x}{Y}$:

$$\left[CF = L \ q \ R \ \ \& \ \ q = \frac{x}{z} \right] \ ==>$$

$$\left[[\ L \ IN \ .. LC \ \ \& \ \ R \ IN \ RC \ ..] \ \ ==> \ q \ IN \ CP \right]$$

Now we have defined individual elementary rules. Full descriptions are, in principle, collections of elementary rules: $Rule_1$, ..., $Rule_n$. Each elementary rule is based on a condition:

$$CTX_i(L,q,R)$$

which tests one instance of q within a configuration CF = L q R according to the corresponding $Rule_i$. The interpretation of the whole rule component when applied to a configuration CF is the following. If q is a pair in CF, i.e. CF = L q R then:

1) If there is more than one coercion rule "<=" for the same q, (q IN CP_j, ..., q IN CP_m), they must all be satisfied: $CTX_j(L,q,R)$ & ... & $CTX_m(L,q,R)$.

2) If there are several context restriction rules "=>" for q, (where q IN CP_k, .., q IN CP_n), one of them must be satisfied $CTX_k(L,q,R)$ or ... or $CTX_n(L,q,R)$.

In other words, context restriction rules "=>" referring to the same pair are taken disjunctively. These bunches and all coercion rules "<=" are taken conjunctively. This principle seems reasonable because supplementary necessary conditions in "=>" rules may arise from different phonological phenomena, and they seem quite justified, whereas

simultaneous necessary context conditions would more likely result in contradictions. The conjunction of elementary coercion rules "<=" means (e.g. by de Morgan's laws) that the disjunction of the sufficient context is effectively used. The need for indicating distinct areas for the rule is more obvious than the definitions of domains as intersections.

2.3.9 Complex rules

It is quite common that we can combine the effects of context restriction (only if) rules and surface coercion (if) rules because at the same time we know that certain alternations occur only in a certain environment and that then there is no other choice. The if and only if combination is called:

Composite rule:

CP <=> LC — RC

This is defined as the combination of "only if" and "if" rules:

CP => LC — RC
CP <= LC — RC

All rules may have a list of CPs instead of a single CP:

CP_1, CP_2, ..., CP_n "op" LC — RC

This is interpreted simply as the corresponding separate elementary (or composite) rules:

CP_1 "op" LC — RC
CP_2 "op" LC — RC
. . .
CP_n "op" LC — RC

All types of rules may have multiple right parts. The additional context parts are written below the first one in the following way:

$$
\begin{array}{llll}
CP & \text{"op"} & LC_1 & — & RC_1 \\
 & & LC_2 & — & RC_2 \\
 & & \multicolumn{3}{c}{\cdot\ \cdot\ \cdot} \\
 & & LC_n & — & RC_n
\end{array}
$$

In all rules this is interpreted as the n separate rules with CP (or a list of CPs) repeated as their correspondence part:

$$
\begin{array}{llll}
CP & \text{"op"} & LC_1 & — & RC_1 \\
CP & \text{"op"} & LC_2 & — & RC_2 \\
\multicolumn{4}{c}{\cdot\ \cdot\ \cdot} \\
CP & \text{"op"} & LC_n & — & RC_n
\end{array}
$$

2.3.10 Angle brackets in rules

Common subexpressions (sets or regular pair expressions) may be defined when necessary and the name of the subexpression used instead of the full expression, e.g.:

$$\text{GRAD} = \binom{\mathbf{V}}{=} \left(+ \quad \mathbf{i} \quad \text{")"}\right) \quad \$$$

$$\mathbf{Cn} = \{\mathbf{l},\ \mathbf{r},\ \mathbf{n}\}$$

Angle brackets will be used in standard generative fashion for abbreviating several rules in a single rule. We may enclose lists of RPEs in the angle brackets in the ordinary way, e.g.:

<l, r, n>

If we have a set which is previously or separately defined, we may enclose its name in angle brackets as well. E.g. <Cl> is identical to:

<l, r>

In all rules angle brackets may be used for abbreviating a series of rules, where only one component varies. Such a notation always corresponds to separate rules with respective elements from the angle brackets, e.g.:

$$\overset{\mathbf{T}}{\mathbf{<Cn>}} \quad => \quad \mathbf{V} \quad \mathbf{<Cn>} \quad —$$

stands for rules:

$$\begin{matrix} T \\ l \end{matrix} \quad => \quad V \; l \; —$$

$$\begin{matrix} T \\ r \end{matrix} \quad => \quad V \; r \; —$$

$$\begin{matrix} T \\ n \end{matrix} \quad => \quad V \; n \; —$$

Similarly:

$$\begin{matrix} K \\ v \end{matrix} \quad => \quad C \; \langle u, y \rangle \; — \; \langle u, y \rangle$$

stands for:

$$\begin{matrix} K \\ v \end{matrix} \quad => \quad C \; u \; — \; u$$

$$\begin{matrix} K \\ v \end{matrix} \quad => \quad C \; y \; — \; y$$

Rules using the above abbreviatory conventions (<=>, multiple CPs, multiple contexts, <>) may be reduced into simpler rules in any order, the result will be the same.

2.3.11 Further rule conventions

We have discussed the special role of the feature characters. In order to stress their nonphonological substance, we introduce a convention that one or more features, which are not mentioned in the rule, may optionally occur between any symbols in the contexts LC and RC. Thus, the following configuration is also captured by the context (V + — V) of the rule in 2.3.7:

t a l o & + i e n
t a l o ø ø j e n

Especially the left context (V +) is satisfied although there is an extra "&" between the "V" and the "+".

We have now defined a formalism which is powerful enough for full scale morphological descriptions. Most of the rules to be written will be context restriction or composite rules. The use of coercion rules will be less frequent. The most important role for that rule type is its being part of composite rules. It will be used in the Finnish description for defining some "holes" in otherwise regular contexts. E.g. the following rule excludes a special case of the distribution of

-ten ending for plural genitives (they may not follow a t:

$$\underset{\{\ \}}{2} \quad <= \quad t \quad --$$

This is to be read: "If we have a 2 in a context t — it may correspond to nothing". This forbids such an occurrence of 2 altogether.

There is another kind of an exception: under special conditions another correspondence is used instead of the normal one. In a limited context the ordinary correspondence is forbidden, and for another it is the only permissible context. Thus e.g. simultaneuosly:

$$\underset{'}{K} \quad => \quad V \ <V> \ -- \ <V>$$

$$\underset{-\emptyset}{K} \quad <= \quad V \ <V> \ -- \ <V>$$

The latter rule effectively forbids the correspondence $\underset{\emptyset}{K}$. Because these have a common right part they may be abbreviated as:

$$\underset{'}{K} \quad => \quad V \ <V> \ -- \ <V> \quad => \quad \underset{-\emptyset}{K}$$

2.3.12 Simulating rules with automata

We have planned to implement a compiler for the rule formalism described above. It would convert the rules into a set of finite-state automata. At this stage the model has been implemented so that it uses automata which are hand-compiled. Because of the intimate connection between rules and automata, the hand-coding has proven to be quite easy. The representation of the rule component as a set of finite-state automata provides some insight into the logical structure of the rule formalism.

Each automaton to be written will be in a conjunctive relation to all other automata, i.e. the configurations must pass all automata in order to be acceptable. Quite often several rules are coded into a single automaton. This is necessary in case of context restriction rules referring to the same CP, because there is no other way to achieve the disjunctive relation. The other reason for sharing an automaton between several rules is that the rules often have identical (or similar) contexts, and thus most of the automaton will be common to the rules. It would be feasible to use separate automata, but much of the automata should be repeated. The hand-compilation of the finite state automata is further discussed in section 4.2.

CHAPTER 3
A TWO-LEVEL DESCRIPTION OF FINNISH MORPHOLOGY

This chapter has two parts: one describes the lexicon, the other the two-level rules for Finnish. The parts are highly interdependent, because for many phenomena there exists the choice of describing them within the lexicon system or with rules (cf. Jackendoff 1975). I have been fairly cautious with the rules, and described some variations lexically that are often handled with rules. This is partly a consequence of the structure of the two-level model, and partly of personal choice. I have tried to achieve overall simplicity, which I consider to be a combination of consistency, lack of complicated and unexpected interferences between rules, and brevity of description (I do not attempt to give an evaluation measure for it.) In addition to simplicity, I have used criteria such as the productivity of variations and the frequency in the lexicon.

This description of Finnish morphology is different from its predecessors. The differences arise mostly from the different scope and formalism of this work rather than from disagreements in the interpretation of the morphological structure. The main goal of the present work has been to build a general and operational model and apply it to Finnish morphology. It is an operational and thus dynamic model which is also interpreted as a model of the morphological performance of a Finnish speaker-listener.

The description is purely synchronic and concentrates on Standard Finnish ignoring all types of linguistic variation. Even as a synchronic description it leaves out some important areas such as the deduction of lexical representations from the most frequent actually occurring word forms, and the description of lexical redundancy. These areas have been covered thoroughly by F. Karlsson (1983b).

Most of the description below follows the framework outlined in standard grammars (cf. Setälä 1963, Penttilä 1963, F. Karlsson 1979, 1983a, 1983b). Perhaps the main difference between this and previous descriptions is the treatment of alternative endings.

The present description is comprehensive. It covers the full inflection of nouns, adjectives and verbs, including affixation of possessive suffixes and clitics, and compounding. I have furthermore written a small program which produces two-level entries from the entries in the Reverse Dictionary of Modern Standard Finnish (Tuomi 1972). Some peripheral areas have been omitted in the present descrip-

tion because of their unimportance rather than because of any serious difficulties in describing them:

(1) Pronouns and numerals. Many of these have idiosyncratic inflection which can be handled in the present framework either by including new alternation patterns or by inserting parallel stems into the root lexicon. These classes are closed and they represent a negligible portion of the lexicon.

(2) Certain adverbs with restricted inflection (only the local cases). These can be handled with a sublexicon consisting of the appropriate subset of case endings. These entries are not marked in the Nykysuomen Sanakirja and I have delayed handling them until the lexicon is checked and revised in this respect. The textual frequency of unanalyzed word forms of this kind is small.

(3) Some foreign words retain their original orthography and pronunciation, for which reason they do not follow the Finnish rules, e.g. the illative of the proper name **Bordeaux** repeats the last vowel that is pronounced: **Bordeaux'hon.** These are rare, however.

The alphabet on which the two-level description is based was already presented in chapter 2.

3.1 A TWO-LEVEL LEXICON SYSTEM FOR FINNISH

The lexical aspect of Finnish word inflection consists of two major parts: stem formation and morphotactic structure. The stem formation is, however, not a purely lexical phenomenon. A part of it is handled by rules to be given in section 3.2. The lexical part of the stem formation consists of identifying basic stems for nouns and verbs, and of classifying entry words into appropriate inflectional types. Each of the more complicated types will be assigned a characteristic alternation pattern.

The morphotactic structure is defined by establishing representations for endings and by grouping them into appropriate sublexicons. Correct combinations are defined by using proper continuation classes

for each root, alternation entry, and ending in the lexicons.

The stem formation mechanisms and the morphotactic structures in the nominal and verbal inflection are formally similar. The maximal nominal paradigm contains some 2000 distinct word forms, and the full verbal paradigm some 12,000 forms (see e.g. F. Karlsson 1983b, p. 356). In compensation for the larger number of forms, the verbal conjugation is more regular and it needs less lexical marking: stem formation and endings can be deduced from the strong vowel stem of each verb. Nouns, on the contrary, need some lexical markers to distinguish between various stem formation patterns. These redundancies are not fully reflected in the present description, which is a performance model of how the words in the lexicon are processed rather than of how words are entered into the lexicon. Thus a more detailed classification of word root entries is used which reflects the view that certain alternations are not produced from scratch, but are rather coded as associations to the alternation patterns.

Finnish is a fairly agglutinating language because the roots and endings are normally segmentable in the inflected surface forms. However, some morphemes have quite different shapes in certain combinations of endings (e.g. genitive singular vs. plural). The connection between such shapes can be accounted for diachronically only, synchronically the relation is irregular. Furthermore, there are substantial asymmetries in the combinations of (perhaps otherwise well segmentable) endings (e.g. imperative vs. present tense forms). In the following description, I have used portmanteau representations in such instances for the sake of simplicity and because of my concern for psychological reality of the description (cf. 3.1.5). There is nothing in the two-level model to prevent the use of fully segmented endings, if one wishes to use them.

3.1.1 The stems of Finnish nouns

Finnish nouns have one or more distinct stems. Many words like **talo** have only one invariant stem. Other words like **katto** 'roof' have variation within the stem: **katolla** 'on the roof'. This particular variation is due to consonant gradation which is easiest to describe with rules. Lexically, there is thus only one stem for such words. Some words have several distinct stems, e.g. **rakkaus** 'love' represents a fairly common type:

rakkaus	nominative singular
rakkauteen	illative singular
rakkautta	partitive singular
rakkauksiin	illative plural

The stems in the second and the third forms are related through a rule of **e-ø** alternation at the end of the stem. Three distinct stems remain. A word representing a different type, **saapas** 'boot' shows a slightly different pattern:

saapas	nominative singular
saappaaseen	illative singular
saapasta	partitive singular
saappaisiin	illative plural

Here the first and the third stems are identical, and the second and the fourth are related through the rule of double vowel simplification. Two distinct stems remain but they have a different distribution than the stems in the previous type. Thus we cannot make one partitioning of the case endings that would reflect the existence of distinct stems for all nouns. The problem is, however, easy to resolve by making a four-way partitioning of the endings which is too detailed for most nouns but general enough for all. The stem formation for individual noun types is then described by using these smaller partitions as building blocks. The partition into four stems is the following:

Nominative stem (S0)

This is linguistically and psycholinguistically the most important one. It is identical to the nominative singular form of the word, e.g. **talo** 'house', **saapas** 'boot'.

Singular inflectional stem (S1)

This stem is manifested in most of the inflected singular forms, e.g. in the inessive: **saappaassa** 'in (a) boot'. (This stem is also called the "singular stem" later on.)

Plural inflectional stem (S2)

This stem is manifested in plural forms based on the ending **i**, e.g. inessive plural **vapauksissa** 'in liberties'. (This stem is also called the "plural stem" later on.)

Consonant stem (S3)

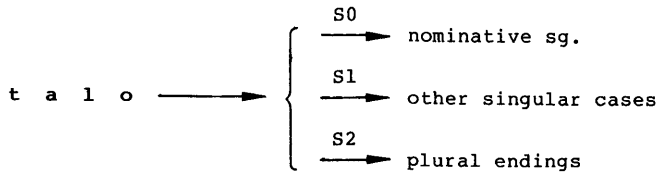
This stem exists only for a part of the nominals, and is then to be found in the partitive singular and genitive plural, e.g. **saapasta** 'boot (partitive singular)'.

Few nominals have four lexically distinct stems, e.g. **kahdeksas** 'eighth'. The lexical representations of the singular and the plural stems are identical for most nouns. Many nouns do not have inflectional stems distinct from the nominative stem. Neither do most nouns have a consonant stem. The various combinations of nominal stems, including the idiosyncratic ones, can easily be handled within this four stem framework.

Noun endings are grouped into a few sublexicons. Each of the above stem types has its own sublexicon consisting of those endings that are normally affixed to them.

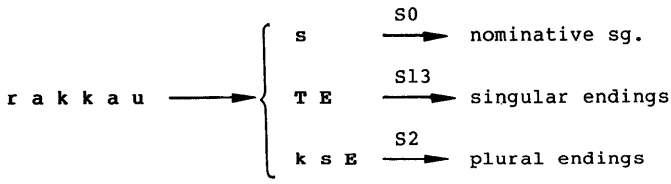
3.1.2 Continuation classes for nouns

Many words have only (fairly) natural one-segment alternations (or no alternations at all) and they need lexically only one common stem. Any differences in the surface stems are accounted for by the two-level rules. Inflectional types with suppletion-like alternations need an alternation pattern, which is a sublexicon that lists the lexical representations of nonnaturally related alternatives. Each of these alternatives corresponds to one or more of the four stems and can thus accommodate only some of the possible endings. The endings for each of the stems form separate sublexicons S0, S1, S2 and S3. The association of corresponding series of endings to the stems is defined by the continuation classes. Simple noun entries may contain the representation of the only stem in full and refer directly to the ending sublexicons, schematically e.g.:



Nouns with nonnatural alternations refer first to the appropriate

alternation pattern. The entries there, in turn, refer selectively to the appropriate S-sublexicons, schematically e.g.:



The following are the continuation classes for nouns. The names of the classes have a mnemonic convention for encoding the ending lexicon numbers.

```
( /S = S0 S1 S2 S3 )
( /S* = S0 S1 S2 S3 S5 )
( S01 = S0 S1 )
( S03 = S0 S3 )
( S04 = S0 S4 )
( S12 = S1 S2 )
( S123 = S1 S2 S3 )
( S13 = S1 S3 )
```

The continuation class /S is the most common one, and it is used in word entries where there is only one stem. The consonant stem, insofar as one exists, is formed by realizing the last vowel of the stem as zero. The class /S* is reserved for trisyllabic nouns which are subject to so called "special consonant gradation" (cf. section 3.2.9). Others are used in those alternation patterns where a stem alternant can adapt only a subset of the possible endings, e.g. S12 is a continuation class to be used in alternation pattern entries which stand for both singular and plural stems. The continuation may then be any ending in the singular lexicon S1 or the plural lexicon S2. Continuation classes consisting of a single minilexicon have been omitted, the name of the sublexicon serves as the name of the continuation class.

3.1.3 Endings for nominative stems

The nominative singular has no surface ending. I have, however, assigned a sublexicon for nominative stems, which contains empty phonological representations. The first entry is a null entry providing the appropriate morpholexical information. It lets the word either stop here or have a clitic at the end (cf. section 3.1.10). The second entry provides a continuation for compound words and is discussed in section 3.1.15:

```
LEXICON S0
     0          K     "NOM SG";
     #          R     "NOM SG"
```

3.1.4 Endings for singular stems

Most singular case endings are affixed to the singular inflectional stem. It should be emphasized that the term "singular" stem is technical. It contains a few plural forms along with the singular case endings. Singular and plural stems are defined according to the absence or presence of the (underlying) plural **i**, not according to the morphosyntactic "number" feature.

```
LEXICON S1
     $+n        K     "GEN SG";
     $+n#       R     "GEN SG";
     +n         Pl    "NOM SG/ NOM PL/ GEN SG";
     +nA        P     "ESS SG";
     !+A        P     "PTV SG";
     $+ksi      K     "TRA SG";
     $+kse      Pl    "TRA SG";
     $+ssA      P     "INE SG";
     $+stA      P     "ELA SG";
     3+:n       P     "ILL SG";
     4+h:n      P     "ILL SG";
     5+seen     P     "ILL SG";
     $+llA      P     "ADE SG";
     $+ltA      P     "ABL SG";
     $+lle      P     "ALL SG";
```

```
$+ttA      P    "ABE SG";
$+t        K    "NOM PL";
%+en       P    "GEN PL"
```

In the lexical representations we should pay attention to:

(1) The trigger **$** for consonant gradation (cf. 3.2.8).

(2) The juncture **+** denoting the boundary and the presence of a case ending.

(3) The archiphoneme **A** for vowel harmony (cf. 3.2.5), and **:** for vowel doubling (this is the archiphoneme usage of :, cf. 3.2.1).

(4) The two parallel entries **ksi** and **kse** for the translative. This alternation is restricted to the translative only and it is therefore handled within the lexicon and not by rules.

(5) The three alternative endings for the illative. These are distinct quasimorphemes each marked with a feature (**3**, **4** or **5**) for selecting the proper endings according to the gestalt of the stem (cf. 3.2.9).

The second column in the entries gives the continuations. The labels P, Pl and K refer to optional and obligatory possessive suffixes and clitic particles, respectively (cf. section 3.1.10). The genitive singular and nominative plural trigger consonant gradation except when followed by a possessive suffix. This irregularity is handled by using a separate entry for the possessive genitive singular. (This form is always homographic with the possessive nominative singular and plural.)

Some grammars use the term "accusative case" for nominatives and genitives occurring as syntactic objects. The accusative has an independent morphological manifestation only in the personal pronouns, e.g. **minu<u>t</u>** 'me'. I have given no separate entry for the accusative case in nouns, because it is always homographic with the nominative and genitive forms. The decision whether or not to introduce separate entries in such instances depends on syntactic considerations. The disambiguation is no easier if a systematic homography is added here.

3.1.5 Endings for plural stems

The plural lexicon makes use of portmanteau representations for the plural ending i in combination with case endings proper. I have chosen this approach, although I am aware that it is against the more common practice of treating the plural i and the case endings as distinct morphemes. My arguments are partly linguistic and partly technical. The singular and plural paradigms are linguistically far from isomorphic. Six of the fourteen cases have discrepancies. All three of the most important cases (nominative, genitive, partitive) are deviant. The differences include: (1) the presence of plural i (nominative, genitive), (2) the shape of the case ending (nominative, genitive, illative), (3) the distribution of allomorphs (partitive, genitive, illative) and (4) morphosyntactic asymmetry (comitative, instructive). (Cf. also the discussion in F. Karlsson 1983b, p. 294.)

The more technical reason for this choice is that the series of singular and plural case endings only partially overlap. It would be quite possible to establish a separate minilexicon for the plural i, and split both S1 and S2. Correct combinations would be guaranteed by introducing two new continuation classes. The portmanteau solution seems to be slightly simpler. These plural endings are not the only ones where similar arguments apply, the verbal paradigm in particular contains nonsymmetric areas where endings should be treated as wholes, at least in the context of performance processes (cf. 3.1.13).

The plural lexicon consists of elements that are familiar from the singular lexicon:

```
LEXICON S2
     &+ien      P    "GEN PL";
     $6+iden    P    "GEN PL";
     $6+itten   P    "GEN PL";
     +inA       P    "ESS PL";
     7+iA       P    "PTV PL";
     $6+itA     P    "PTV PL";
     $+iksi     K    "TRA PL";
     $+ikse     Pl   "TRA PL";
     $+issA     P    "INE PL";
     $+istA     P    "ELA PL";
     2+iin      P    "ILL PL";
     8+ihin     P    "ILL PL";
     9+isiin    P    "ILL PL";
     $+illA     P    "ADE PL";
```

```
$+iltA      P     "ABL PL";
$+ille      P     "ALL PL";
$+ittA      P     "ABE PL";
+ine        P     "CMT";
$+in        K     "INS PL"
```

3.1.6 Other case endings

The lexicon S3 contains endings which typically occur after a consonant stem. It has only two entries, both of which are tagged with a selector feature. These features make them almost self-sufficient in selecting their correct contexts (cf. 3.2.9). They form a separate lexicon in order to reserve the possibility of explicit exclusion when necessary. Some nominative stems do not permit these endings, although they look misleadingly like other stems which permit them (e.g. ***hevonenta** 'horse (partitive)' and ***kolmasta** 'third (partitive)' are impossible, but **poljinta** 'pedal (partitive)' and **lammasta** 'sheep (partitive)' are acceptable).

```
LEXICON S3
     1+tA       P     "PTV SG";
     2+ten      P     "GEN PL"
```

The sublexicon S4 is used only in the common inflectional type **vene** 'boat', where **ttA** is to be interpreted as a partitive ending.

```
LEXICON S4
     +ttA       P     "PTV SG"
```

S5 is reserved for "special consonant gradation" in trisyllabic stems. E.g. **mellakka** 'riot' has two equally acceptable illative plurals: **mellakoihin, mellakkoihin.** The latter corresponds to the illative plural entry in S2, and the former to an entry in S5. The entries in S5 are otherwise similar to the normal endings except that they trigger the weakening in the three forms subject to special gradation (although an open syllable follows) (cf. 3.2.9).

```
LEXICON S5
     $+inA      P     "ESS PL";
     $8+ihin    P     "ILL PL";
     $+ine      P     "CMT"
```

3.1.7 Combining nominal stems and case endings

It was mentioned above that the morphotactic structure would allow endings in any of the lexicons S0, S1, S2 and S3 to follow the only stem of many words. (Entries in S3 are often rejected by rules.) Such word entries have a simple continuation /S. The following examples represent inflectional types described without alternation patterns for stem formation.

```
kaTo        /S   "Loss (failure of crops) S";
katTo       /S   "Roof S";
liuKu       /S   "Glide S";
puKu        /S   "Dress S";
risti       /S   "Cross S";
kivE        /S   "Stone S";
läPE        /S   "Hole S";
kielE       /S   "Language S";
koira       /S   "Dog S";
kalÅ        /S   "Fish S";
ma:         /S   "Earth S";
sukla:      /S   "Chocolate S";
```

The corresponding surface forms of the four stems are not necessarily identical. Any variation in them is accounted for by two-level rules. In the current description all natural one-segment alternations in the stem formation are morphophonological, the phonological alternations are restricted to the archiphonemes in the endings (cf. Linell 1979). These morphophonological alternations include consonant gradation (**K**, **P**, **T**), double vowels (**:**), and certain vowel alternations (**a**, **i**, and morphophonemes **E**, **Å** in stem final position). No particular phonetic values are assigned to the morphophonemes, they are best thought of as relations or labels for alternations (cf. Anttila 1980).

There are two kinds of consonant stems. One arises when the final morphophoneme **E** of the singular stem corresponds to zero on surface, e.g. in the only lexical stem of **kielE** 'language' the final **E** corresponds to zero before the partitive singular ending **-tA:** **kieltä** (cf. 3.2.4). The other type of consonant stems is identical to the nominative stem ending in a consonant, e.g. **saapasta** corresponding to **saapas** 'boot'.

I have used the same stem as singular and plural stem, and I have related them with rules where it has been possible. There would have

been another possibility of using alternation patterns giving the "ready-made" plural stems where the effects of the plural **i** would be present already in the lexical representation. The choice between these two approaches is not just a matter of taste. The distribution of e.g. the plural illative **isiin** depends on the relation between these two stems. The distribution is easily described by reference to the simplification of a double vowel, whereas the phonological appearance of a ready made plural stem does not have sufficient information (cf. section 3.2.9).

The following are two examples of word entries with "special consonant gradation" (cf. 3.2.9). They only need a wider continuation class:

```
harakKÅ        /S*  "Magpie S";
yksikKÖ        /S*  "Unit S";
```

3.1.8 Alternation patterns for nouns

One of the basic principles of the two-level model is that all alternations need not be handled by rules. If the relation between the alternatives is nonnatural or suppletion-like, they are given as such in the form of a minilexicon. There are many nominal inflectional types that have a characteristic pattern at the end of the stem. Word entries in those types refer to the case endings only indirectly via the minilexicons. The minilexicons have two tasks: (1) they list the alternatives in the stem formation, and (2) they assign the correct subset of case endings to the alternatives.

Inflectional types are discussed by giving representative word entries and the associated alternation pattern minilexicons:

```
kalsium 0-i/S       "Calcium S";

LEXICON 0-i/S       0   S0 "";
                    i   S12 ""
```

This is the productive inflection scheme for foreign nouns ending in a consonant. The nominative remains unchanged, the inflectional stem has an **i** epenthetically added (all singular stems must end in a vowel). I describe this alternation lexically. Pure epenthesis would complicate the interactions between rules because certain native types overlap with the foreign ones. A morphophoneme "I" could also be used by including rules for realizing it as zero before word boundary and

clitics, as **e** before plural **i**, and as **i** elsewhere. I have chosen to use an alternation pattern because of rule consistency: the inflectional stems are related via normal rules, and single **i**'s do not disappear elsewhere. Anyway, this group is not frequent in ordinary running text.

```
lu mi/S              "Snow S";
kä si/S              "Hand S";
la psi/S             "Child S";
vei tsi/S            "Knife S";

LEXICON mi/S         mE  S012 "";
                     n   S3   ""
LEXICON si/S         sE  S02  "";
                     TE S13 ""
LEXICON psi/S        psE S012 "";
                     s   S3 ""
LEXICON tsi/S        tsE S012 "";
                     s   S3 ""
```

This group contains consonant alternations: assimilation of **m**, **t-s** alternation, and consonant cluster simplification. These are usually handled with rules (Wiik 1967, F. Karlsson 1974). I have used the lexical approach mainly because of the unproductivity of these alternations. Some generality is also achieved because there is no guarantee that the consonant clusters to be simplified would not show up in foreign names (where they remain constant).

The use of an alternation pattern corresponds to the interpretation that all word entries of the type "käsi" have an alternation in common. This alternation is stored as a pattern which is referred to, not constructed each time, when word forms are produced or recognized.

```
ka ksi               "Two NUM";

LEXICON ksi          ksE S02  "";
                     hTe S1   "";
                     htA P "PTV SG"
```

This is an example of handling exceptions. There are three things unique to two common numerals, **yksi** 'one' and **kaksi** 'two': (1) there is a **ks-hT** alternation, (2) the consonant stem is formed irregularly, and (3) after the consonant stem the **ten** genitive plural is not possi-

ble, although the partitive **ta** is. The irregular stem alternants are given as such in an idiosyncratic minilexicon. (Of course full stems could have been entered in the root lexicon: kaksE S02 "Two NUM", etc.)

```
sisa r/S            "Sister S";
runotTa r/S         "Muse S";
polKi n-me/S        "Pedal S";
työtTö n-mA/S       "Unemployed (person) S";

LEXICON r/S         $r S03 "";
                    rE  S12 ""
LEXICON n-me/S      $n S03 "";
                    mE  S12 ""
LEXICON n-mA/S      $n S03 "";
                    mA  A12 ""
```

These inflectional types have a nominative consonant stem and they are subject to "inverted consonant gradation", i.e. the nominative stem has the weak grade and the inflectional stems the strong grade. The gradation is here conditioned by the stem alternant rather than by the endings (or whether the following syllable is open or closed). The S3 endings are appended to the nominative stem rather than to the inflectional stems with zero realization of **E** (e.g. **runotarta** from the nominative stem **runotar** 'muse' instead of ***runottarØta** from the inflectional stem).

```
hevo nen/S          "Horse S";

LEXICON nen/S       nen K     "";
                    sE  S123  "";
                    s#  R     "NOM SG"
```

This is the full descripion of the common and productive **nen-se** alternation which has been repeatedly used as an example. Note that the singular, plural, and consonant stem are all represented by the **-sE** alternative. In plural and consonant stem forms, the **E** corresponds to zero according to the rules. Words of this particular type have a so called casus componens, which is used when they occur as a nonfinal part of a compound word (it is the third alternative: **s#**). This is the main type of Finnish words where such a form is used instead of the nominative singular. The description allows **hevosmies** 'horseman' but not ***hevonenmies.**

```
kato s-kse/S          "Shed S";
vastau s-kse/S        "Answer S";
rakkau s-Te/S         "Love S";

LEXICON s-kse/S       s  S03 "";
                      ksE S12 ""
LEXICON s-Te/S        s  S0  "";
                      TE  S13 "";
                      ksE S2 ""
```

These two overlapping types are native, frequent, and productive. Lexical marking (here: a distinct alternation pattern) is needed to distinguish them. The alternation s-kse/S undergoes a similar consonant cluster reduction as e.g. psi/S above. The alternation s-Te/S shows a more drastic variation **s-TE-ksE** which typically deserves an alternation pattern.

```
kevä t-:/S            "Spring (season) S";
hankKe 0-:/S          "Project S";
saapPa s-:/S          "Boot S";

LEXICON t-:/S         $t S03 "";
                      :  S12 ""
LEXICON 0-:/S         $  S04 "";
                      :  S12 ""
LEXICON s-:/S         $s S03 "";
                      :  S12 ""
```

These types are further examples of the "inverted consonant gradation". The alternation t-:/S is rare, 0-:/S is increasingly common due to the **-e** deverbal derivational suffix. The nonproductive alternation s-:/S is yielding to the more common and productive s-kse/S alternation.

3.1.9 Degrees of comparison

Adjectives have three degrees of comparison: positive, comparative, and superlative. All grades are inflected like nouns. The comparative and superlative grades are first formed from the singular and plural stems, respectively, and then inflected like nouns. The continuation classes for adjectives are very similar to those for nouns.

```
( /A = C1 C2 S0 S1 S2 S3 )
( /A* = C1 C2 S0 S1 S2 S5 )
( A1 = C1 S1 )
( A12 = C1 C2 S1 S2 )
( A123 = C1 C2 S1 S2 S3 )
( A13 = C1 S1 S3 )
( A12C1 = S1 S2 C1 )
( A2 = C2 S2 )
```

The main difference is the inclusion of two minilexicons: C1 for the comparative grade, and C2 for the superlative. The comparative and superlative endings are appended in the same way as alternation patterns:

```
LEXICON C1
     $@mpi      S0    "CMP";
     $@mPA      S12   "CMP";
     $@mmin     K     "CMP MAN";
     $+sti      K     "POS MAN"

LEXICON C2
     $+in       S03   "SUP";
     $+imPA     S12   "SUP";
     $+immin    K     "SUP MAN"
```

All these endings close the syllable and trigger consonant gradation (if other conditions are satisfied). The comparative endings are marked with the feature **@** because they trigger the **a-e** alternation in stems ending in **a** (e.g. **paha** 'bad' **pahempi** 'worse'). This mechanism avoids the need for a fifth stem for comparatives. The superlative **i** acts almost like the plural **i**.

As in nouns, some adjectives need no alternation patterns. All ending series S0, S1, S2, S3, C1 and C2 may be affixed to them (S3 if the rules allow the zero realization of the final vowel):

```
aiTo          /A   "Genuine A";
tunnetTu      /A   "Well-known A";
auti.o        /A   "Desert A";
marKä         /A   "Wet A";
pahÅ          /A   "Bad A";
raaKÅ         /A   "Raw A";
rohke.a       /A   "Brave A";
```

Note the period marking the syllable boundary in **rohke.a**, it is used by the rules referring to the syllable structure of the stem.

Below are examples of representative entries making use of alternation patterns and the associated minilexicons:

```
onnetTo n-mA/A      "Unhappy A";
kuoll Ut/A          "Dead A";

LEXICON n-mA/A      $n S03 "";
                    mA A12 ""
LEXICON Ut/A        Ut S03 "";
                    e: A12 ""
```

These types represent two stem alternation patterns similar to noun patterns but normally present in adjectives. The Ut/A pattern is used in the past participle forms of verbs.

```
rauhalli nen/A      "Calm A";
viera s-:/A         "Guest/Strange A";

LEXICON nen/A       nen  K "";
                    sE A123 "";
                    s# R "NOM SG"
LEXICON s-:/A       $s S03 "";
                    :  A12 ""
```

These types are obvious parallels to the corresponding noun patterns: C1 is only added to alternatives where nouns have S1, C2 to those that have S2. (In fact, the adjectival pattern could be used for nouns if we could impose conditions on the combinations of morphological features for excluding CMP and SUP if the part of speech is S.)

```
kevy t-e/A          "Light-weight A";

LEXICON t-e/A       t  S03 "";
                    .e A1  "";
                    E  A2  ""
```

These adjectives have a complication in the relation between their singular and plural stems. The singular stem ends in **ue** or **ye**, and there is a syllable boundary between the vowels. In the plural, **E** is realized as zero, and there is no syllable boundary because the plural

i forms a diphthong with any vowel. The present two-level description needs a correct syllable structure for the distribution of ending allomorphs. Therefore I have to give a distinct singular with the syllable boundary mark, and a plural stem without it. (It is possible that this could be avoided by using more elaborate patterns for syllable structure.)

3.1.10 Possessives and clitics

Word forms need not end with a case ending. Most inflected forms may have a possessive suffix and/or a clitic particle. The entries for endings given above have referred to the following continuation classes:

```
( P = P K0 # )
( P1 = P )
( K = K0 # )
( K01 = K0 K1 # )
( R = Root )
```

The first class gives the common continuation when a word may continue with a possessive suffix (P), or a clitic (K0), or end without either of these (#). Class P1 is for obligatory possessives. (This class is needed anyway because of the translatives of first infinitives, e.g. **saadakseni** 'in order that I would get' where the possessive suffix is obligatory.) I have omitted the rule for turning the **i** in the translative **ksi** into an **e** before possessive suffixes. This class is also used in the genitive singular with possessive suffixes, because then we may deactivate consonant gradation by using a separate entry without $. (Because the cancelling occurs only with a possessive the alternative ending must force one to be present).

The lexicon for possessives is straightforward, each entry is marked with a feature (/) for the rules to be able to recognize possessives:

```
LEXICON P
     /ni        K     "SG1";
     /si        K     "SG2";
     /nsA       K     "SG3/PL3";
     /:n        K     "SG3/PL3";
     /mme       K     "PL1";
     /nne       K     "PL2"
```

The lexicons for clitic particles are equally straightforward. There are two such lexicons, one is the common set that can be affixed to nearly any word form. K1 contains the entry **s** which is possible only in familiar imperatives.

```
LEXICON K0
     _hAn       #    "han";
     _kA:n      #    "kaan";
     _kin       #    "kin";
     _kO        #    "ko";
     _pA        #    "pa";
     _kinkO     #    "kin ko";
     _kA:nkO    #    "kaaan ko";
     _kOhAn     #    "ko han";
     _kOs       #    "ko s";
     _pAhAn     #    "pa han";
     _pAs       #    "pa s"

LEXICON K1  _s  #    "s"
```

All clitics are marked with a feature (_) so that the rules can identify them (this boundary is equivalent to the end of a word form except for vowel harmony).

3.1.11 Verb stems

Verb stem formation is similar to that in nominal inflection. Many verbs have only one distinct stem, and this is used throughout the paradigm, e.g. **kiehu** 'boil (intransitive)':

kiehuu	"PRES SG3"
kiehui	"PAST SG3"
kiehuisi	"COND SG3"
kiehukoon	"IMPV ACT SG3"

Many verbs have more complex stems. An example of the common and productive type usually called contracted verbs is **varaa** 'reserve':

varaa	"PRES SG3"
varasi	"PAST SG3"
varaisi	"COND SG3"
varatkoon	"IMPV ACT SG3"

In some conjugational types the past tense and the conditional have identical lexical stems. In others they are distinct, but the conditional and the present tense share an identical stem. E.g. the stems in **huutaa** '(he) shouts' and **huutaisi** '(he) would shout' are identical but distinct from **huusi** '(he) shouted'.

Thus the technique used in the nominal stem formation is also applicable here. Four stem types are defined. For most verbs some of the stems are identical or related through the two-level rules. There is, however, no parallel to the consonant stem formation in nouns. All four stems are used, though some of them are not independent. The stem types control which endings may be affixed to each stem. The four stem types are:

Present tense stem (V0)
: This is manifested e.g. in present indicative tense (e.g. **huutaa** 'he shouts' and **hakkaa** 'he beats'), and in the second person singular imperative (e.g **kudo** 'knit!'). The form in the 3rd person present tense stem is also called the "strong vowel stem".

Past tense stem (V1)
: This occurs only in the past tense (e.g. **huusi** 'he shouted' and **hakkasi** 'he beat').

Conditional stem (V2)
: This occurs only in the conditional (e.g. **hakkaisi** 'he would beat')

Infinitive stem (V3)
: This stem is manifested in several forms including the potential, all passive forms, several infinitives and imperatives (e.g. **hakatakseen** 'in order to beat').

A separate past tense stem is used because of the idiosyncrasies within the **t-s** alternation in past tense stems. The boundary between the stem and the ending may be defined in different ways. The **s** in **hakkasi** is part of the ending in the descriptions presented by F. Karlsson (1974, 1983b). This reflects the fact that the speakers understand **si** as a good past tense schema in the sense of Bybee and Slobin (1982). I have followed the more traditional line of using only one past tense ending. The **si** interpretation would still require idiosyncratic modifications in the past tense stem formation. The condi-

tional stem is only marginally independent. In most verbs, the present tense stem could be used by adding some rules. The separate conditional stem saves us from these and enables a slightly more accurate description. The infinitive stem is distinct from the other stems e.g. in contracted verbs.

As in nouns, there is a sublexicon corresponding to each of these stems called V0, ..., V3. The following list of continuation classes shows that several combinations of these are used:

```
( /V = V0 V1 V2 V3 )
( V012 = V0 V1 V2 )
( V02 = V0 V2 )
( V023 = V0 V2 V3 )
( V03 = V0 V3 )
( V12 = V1 V2 )
( V123 = V1 V2 V3 )
( V23 = V2 V3 )
```

The continuation /V is used for verbs with only one stem on the lexical level. Others name combinations with a coding similar to that used in the nominal continuation classes. The following are examples of one-stem verbs:

```
kuTo        /V        "Knit V";
muista      /V        "Remember V";
venytTA     /V        "Stretch V";
kanTÅ       /V        "Carry V";
haKE        /V        "Fetch V";
lähTE       /V        "Depart V";
sa:         /V        "Get V";
```

Here the present tense and the infinitive stems are identical even on the surface level. The past tense and conditional stems are either identical with or in a regular relation to these stems according to the two-level rules for vowel alternations or suppression of vowel doubling. The past tense i has the same effect as the plural i. The conditional isi has a slightly more restricted scope but otherwise similar effects (cf. sections 3.2.1-4).

3.1.12 Alternation patterns for verbs

The mechanism for handling complex stem formation for verbs is similar to that for nouns. The various inflectional types are discussed below by giving representative entries and the corresponding alternation pattern minilexicons.

The following rather small and unproductive group of verbs has a **t-s** alternation. Many of these verbs are, however, quite common and there is at least one productive denominal derivational suffix **-nta** with this inflection.

```
huu         tA-s/V     "Shout V";
kiel        tA-s/V     "Deny V";
sou         tA-t,s/V   "Row V";
saar        tÅ-s/V     "Surround V";
tun         te-s/V     "Feel V";

LEXICON tA-s/V        TA  V023 "";
                      s   Vl   ""
LEXICON tA-t,s/V      TA  V023 "";
                      T   Vl   "";
                      s   Vl   ""
LEXICON tÅ-s/V        TÅ  /V   "";
                      s   Vl   ""
LEXICON te-s/V        TE  V023 "";
                      s   Vl   ""
```

The past tense stem is simply given separately with the **s**. A few verbs like **souta** 'row', where the **s** stem and the **T** stems are in free variation, have the past tense ending in two parallel entries.

If the vowel stem of a verb ends in **i**, this segment disappears in the past tense. The past tense **i** otherwise behaves like the plural **i** with respect to causing vowel mutations; this is the only deviation. For this reason I have maintained the overall regularity and assigned those verbs an alternation pattern for the affixation of past tense and conditional endings to an alternant without the **i**:

```
vaaT        i-0/V      "Demand V";

LEXICON     i-0/V      i   V03  "";
                       0   Vl2  ""
```

There is a small group of common verbs otherwise similar to **lähtE** 'depart' except that the infinitive stem ends in a consonant (e.g. **tulkaamme** 'let's come'):

```
tul         E-0/V       "Come V";

LEXICON     E-0/V       E    V012 "";
                        0    V3   ""
```

The following two verb types are productive but idiosyncratic with regard to consonant gradation. The gradating stop is not the last consonant of the stem, and gradation is conditioned by the consonant stem rather than by the ending (here the grade is selected according to whether the following syllable is open or closed, cf. 3.2.8). Otherwise these resemble E-0/V type verbs.

```
jaKe        (e)le/V     "Distribute V";
rohKe       (e)ne/V     "Dare V";

LEXICON     (e)le/V     lE   V012 "";
                        $1   V3   ""
LEXICON     (e)ne/V     nE   V012 "";
                        $t   V3   ""
```

The following types show nontrivial consonant cluster simplifications:

```
valit       sE-0/V      "Choose V";
juo         ksE-s/V     "Run V";
LEXICON     sE-0/V      sE   V012 "";
                        0    V3   ""
LEXICON     ksE-s/V     ksE V012 "";
                        s    V3   ""
```

The verb **rankaise** 'punish' is unique. In addition to the (e)le/V conditioning, it has optional gradation. This idiosyncrasy is handled by entering two parallel stem alternants, one with and the other without the trigger:

```
ranKa       (a)ise/V    "Punish V";
```

```
LEXICON    (a)ise/V  isE V012 "";
                     $is V3  "";
                     is  V3  ""
```

There is a common class of verbs where the vowel stem ends in **oi** or **öi** (the corresponding denominal derivational suffix is productive). These verbs have a mixed paradigm: in parallel to the regular stems, there is an infrequent **tsE** stem in the present and past tense and in the conditional. Alternation patterns handle such complications quite easily.

```
luenno     (o)i/V    "Lecture V";

LEXICON    (o)i/V    i    V03  "";
                     0    V12  "";
                     itsE V012 ""
```

The following type has a suppletive alternation which is also easily represented by a pattern:

```
vär        ise/V     "Vibrate V"

LEXICON    ise/V     isE V012 "";
                     is  V3   "";
                     AjA V02  ""
```

Contracted verbs are common and productive. Their stem alternations appear fairly complicated, partly because of the choice of **kuto**-type verbs as prototypical and their endings as the standard. Parallel series of ending for contracted verbs would, of course, reduce their stem alternations.

```
hakKa      (A):/V    "Beat V";

LEXICON    (A):/V    :   V02  "";
                     s   V1   "";
                     $t  V3   ""
```

The other contracted verbs have similar alternations. If the vowel preceding the final **a** or **ä** is not an **i**, it is optionally realized as zero in the conditional, e.g. **kiipeäisin - kiipeisin** 'I would climb'. (The period in the **.A** alternatives of the patterns A/V, (i)A/V below

denote the intervocalic syllable boundary for the sake of consistency.)

```
kiiPe      A/V       "Climb V";
koKo       A/V       "Assemble V";
silpPu     A/V       "Chop V";
helTi      (i)A/V    "Loosen V";

LEXICON    A/V       .A   V02   "";
                     s    V1    "";
                     0    V2    "";
                     $t   V3    ""
LEXICON    (i)A/V    .A   V02   "";
                     s    V1    "";
                     $t   V3    ""
```

3.1.13 Verbal endings

The four stems have associated ending lexicons. The present tense stem may be followed by the entries in V0:

```
LEXICON V0
     (          1-3-N      "PRES ACT";
     ($         K01        "IMPV ACT SG2";
     (mi        nen/S      "DVMI";
     (mA        /S         "DVMA ACT";
     (vA        /A         "PCP1 ACT";
     (mAtTO     n-mA/A     "PCP3"
```

The left parenthesis is a boundary feature for verbal endings (except for the past tense and the passives). The continuations in all these except the first are already known from the nominal paradigm. The first (1-3-N) refers to the minilexicon of personal endings explained below.

The past tense minilexicon contains i with appropriate boundary features:

```
LEXICON V1
     +i)        1-3        "PAST ACT"
```

The right parenthesis is a boundary feature used for preventing the doubling of the i in third person singular. The conditional lexicon is also simple:

```
LEXICON V2
     (isi)      1-3-N      "COND ACT"
```

The infinitive stem lexicon is larger, and it contains most of the imperatives and participles, as well as all the passives and infinitives.

```
LEXICON V3
     (ne        1-3-N      "POTN ACT";
     (kOOn      K          "IMPV ACT SG3";
     (kAAmme    K          "IMPV ACT PL1";
     (kAA       K01        "IMPV ACT PL2";
     (kAAtte    #          "IMPV ACT PL2";
     (kOOt      #          "IMPV ACT PL3";
     (kO        #          "IMPV ACT NEG";
     *$DA       4-N        "PRES PSS";
     *$Ztiin    K          "PAST PSS PE4";
     *$ZtAisi   4-N        "COND PSS";
     *$ZtAne    4-N        "POTN PSS";
     *$ZtAkO    4-N        "IMPV PSS";
     (DA        K          "INF1 NOM";
     (DA+kse    Pl         "INF1 TRA";
     +De+ssA    P          "INF2 ACT INE";
     +Den       K          "INF2 ACT MAN";
     *$ZtAessA  K          "INF2 PSS INE";
     *$ZtAmAn   K          "DVMA PSS INS";
     *$ZtAvA    /A         "PCP1 PSS";
     (n         Ut/A       "PCP2 ACT";
     *$ZTU      /A*        "PCP2 PSS"
```

Again, the continuations are mostly known from the nouns. Note the feature (*) marking the passive endings. It is used as a trigger for the **a-e** alternation rule (cf. 3.2.2). Portmanteau representations are used e.g. in imperatives and infinitives in order to simplify the combinatorics and avoid unnecessary continuation classes and duplicated subsets of endings.

The verbal paradigm has many forms. Most of them are, however, inflected forms of participles which have the full nominal paradigm: two thousand forms for each nominal ending with a noun reference ../S and six thousand for each nominal ending with an adjective reference.

3.1.14 Personal endings

The personal endings (Sg1-Pl3) are the most important characteristic of the verbal paradigm. In addition to the three singular and plural persons, there is also the negative form, (Neg), and a "fourth" person (Pe4). The negative (Neg) is used in conjunction with the negative auxiliary verb which contains the personal ending. The fourth person occurs in passives, which are impersonal. The following three subsets of personal endings are used:

```
( 1-3-N = Sg1-Pl3 Neg )
( 1-3 = Sg1-Pl3 )
( 4-N = Pe4 Neg )
```

The lexicon for the three singular and plural persons is:

```
LEXICON Sg1-Pl3
     $n        K    "SG1";
     $t        K    "SG2";
     :         K    "SG3";
     $mme      K    "PL1";
     $tte      K    "PL2";
     vAt       K    "PL3"
```

Note the third person singular represented by a vowel doubling. This doubling is sometimes suppressed by rules (cf. 3.2.1). The minilexicons for the negative and for the impersonal passive form are trivial:

```
LEXICON   Neg       $    K    "NEG"

LEXICON   Pe4       :n   K    "PE4"
```

3.1.15 Compound words

Compounding is a productive process in Finnish. The Dictionary of Modern Standard Finnish contains some 200,000 entry words. Although two thirds of them are compound words, the dictionary covers only a fraction of the actually occurring compounds. According to orthographic conventions no spaces or hyphens are inserted between the

component words. This makes the analysis of compounds even more important for the goals of the present work. The solution adopted here is a general one. It permits all kinds of morphologically feasible compounds, even though they might be semantically bizarre or even impossible.

The basic scheme for compounding is that only the last root may be freely inflected; the preceding roots must be nominals in the nominative or genitive singular, e.g.:

tietokonekeskuksessamme	'in our computer centre'
tieto	'data' NOM SG
kone	'machine' NOM SG
keskuksessamme	'center' INE SG PL1

Compounding, like derivation, is easy to implement within the two-level model. The continuation class (R) in the endings for the nominative and genitive singular just include the "Root" lexicon as one alternative. The representation of these endings contain the word boundary juncture (#) for the correct recognition of syllable structure and vowel harmony values of the next component of the compound word.

3.2 TWO-LEVEL RULES FOR FINNISH MORPHOLOGY

The alphabet for the Finnish description and several subsets of it were already given in the preceding chapter as examples of the general formalism and these sets are not repeated here. The roles of surface characters and morphophonemes might need some clarification. Classification of a character as a surface character is a technical statement equivalent to mentioning each surface character corresponding to itself in some redundant rule. The definition of surface characters brings pairs:

a b c d
a b c d ...

into the concrete pair set CPS. Whether some characters (or phonemes) actually participate in morphophonemic alternations or not is an entirely different matter. Among our surface characters e.g. **i, a, ä, n, t** are subject to morphophonemic alternations at the end of the

stem, as will be shown below.

In the following we present the morphological rules according to the rule formalism explained in chapter 2. As it stands now the two-level program only accepts finite state automata. This deficiency will be resolved when a rule compiler has been constructed. The automata used and tested in the actual computer runs are given in appendix A.

3.2.1 Vowel doubling and double vowels

Vowel duplication (or gemination) occurs in some endings, e.g. illatives **taloon** 'into the house' and **kotiin** '(into) home'. Some stems end in a double vowel which alternates with a short one. Because it is necessary to distinguish between stems ending in a double vowel from those ending in a sequence of any two vowels, I use the colon (:) in the lexical representation for stem ends and some ending morphemes. The term 'doubling' is used for the phenomenon in the endings rather than the term 'vowel lengthening' because 'duplication' suits written Finnish better as two consequtive vowels are used. One of the allomorphs of the illative endings, **h:n**, has an intervening **h** between the head vowel and its repetition, e.g. **maahan** 'to the earth' and **puuhun** 'to the tree'. This instance does not accord with the concept "lengthening".

Doubling is by definition a repetition of the preceding vowel. This can be stated as the following two-level context restriction rule:

$$\overset{:}{\langle Vs \rangle} \quad \Rightarrow \quad \overset{=}{\langle Vs \rangle} \quad \left(h\right) \quad \_ \qquad (1a)$$

where

$$Vs = \left\{a,\ e,\ i,\ o,\ u,\ y,\ ä,\ ö\right\}$$

Remember that this is an abbreviation of eight separate rules, one for each of the vowels (cf. section 2.3.10). The first of these is:

$$\overset{:}{a} \quad \Rightarrow \quad \overset{=}{a} \quad \left(h\right) \quad \_$$

The context restriction rule (1a) stands for all these subrules and it dictates that the colon corresponds to a specific vowel only if it is preceded by the same vowel on the surface. There may, however, be one optional intervening **h**, which is a reservation for the **h:n** form of the illative. From this point of view the lexical duplication segment

could be classified as an archiphoneme because in such positions the opposition between vowels is neutralized and the immediately preceding phonological context determines its surface value (cf. Trubetzkoy 1958, p.71 ff.).

The basic use of this rule is exemplified below. The illative ending **3+:n** duplicates the final vowel of the stem, e.g. **talo** 'house':

t	a	l	o	3	+	:	n
t	a	l	o	ø	ø	o	n

We have an instance of $\begin{matrix}:\\o\end{matrix}$ in this configuration. Rule (1a) says that because it is one of the correspondence pairs (CP) mentioned in a "=>" rule, the context condition must be satisfied. In this case it means that a string consisting of some consecutive pairs to the left must be a part of the left context (LC) expression of the rule. If we select the three preceding pairs $\begin{matrix}o&3&+\\o&ø&ø\end{matrix}$ as this sequence, we see that it belongs to LC, which is the following (written in full, cf. 2.3.11):

$$\begin{matrix}=\\o\end{matrix}\ X^*\ \left(h\ X^*\right)$$

where X denotes the set of features. Because the rule has no right context, there is nothing more to be checked and the rule leaves a possibility for the correspondence $\begin{matrix}:\\o\end{matrix}$. The other possibilities (":" realized as some other vowel) are, however, excluded by the rule, because their LC conditions are not satisfied. The possibility of suppression ($\begin{matrix}:\\ø\end{matrix}$) is left open here, and will be resolved by rules (1b,c,d) below which exclude it.

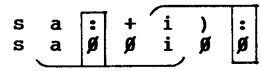
I also use the colon in stems ending in double vowels. In such positions the second vowel (i.e. the colon) is suppressed in certain morpholexically conditioned environments. In this use, the colon is a morphophoneme rather than a archiphoneme. There are three different environments where the colon corresponds to zero: (1b) as the second component of a stem final double vowel in front of plural, past tense or conditional **i** (1c) in third person personal ending in verbs if they are already preceded by a double vowel, or (1d) in the same personal ending if the preceding vowel belongs to the past tense or conditional ending **+i)** or **(isi)**. The following composite rule describes the suppression:

$$\begin{matrix}:\\ø\end{matrix} \quad <=> \quad V\ —\ \left["+" \mid "("\right]\ \begin{matrix}i\\=\end{matrix} \qquad (1b)$$

$$V\ \begin{matrix}:\\V\end{matrix}\ — \qquad (1c)$$

$$V\ ")"\ — \qquad (1d)$$

The word form **sai** is an example of the application of (1b) and (1d). The representation of the verb stem 'get' is "**sa:**", the past tense affix is "**+i)**" and the third person singular ending is "**:**". There is only one correct configuration:



According to rule (1a) the first colon might correspond to an "**a**" and the latter to "**i**". For the first occurrence, this possibility is, however, overridden by the "<=" part of (1b). The colon is mentioned as the lexical character in this coercion rule, the left context "**a**" is a part of the LC and the right context "**+ i**" is a part of the RC of the rule. Therefore, the if-part, or "<=", forbids occurrences other than $\begin{smallmatrix}:\\ø\end{smallmatrix}$ here. Because the "only if" part "=>" refers to the same context, this correspondence is here permitted. (This permission is, of course conditional, but in this case there are no other rules in the description to forbid it.) Similar reasoning for the latter occurrence of "**:**" leads to the same result. Note that the right context of the former correspondence overlaps with the left context of the latter.

There is a complication with the third person possessive suffix **/:n** when the partitive ending **+A** forms a bimorphemic double vowel. If the partitive **A** is preceded by any other vowel than **a** or **ä** it is normally duplicated, e.g. **taloa** 'house (partitive)' with the possessive suffix becomes **taloaan** 'his house (partitive)'. The exception concerns stems ending in **a** or **ä** where the partitive has a double vowel e.g. **kauppaa** 'shop (partitive)'. The **/:n** allomorph of the third person possessive does not occur in such an environment but rather the **/nsA** allomorph: **kauppaansa** 'his shop (partitive)'. This exclusion rule has a net effect similar to that of the distributional selection rules (section 3.2.9).

$$\begin{smallmatrix}:\\ \{\}\end{smallmatrix} \quad <= \quad \begin{smallmatrix}=\\ \mathbf{Va}\end{smallmatrix} \quad \text{"+"} \quad \begin{smallmatrix}\mathbf{A}\\ \mathbf{Va}\end{smallmatrix} \quad \text{"/"} \quad \_\_ \qquad (1e)$$

where

$$\mathbf{Va} = \{\mathbf{a}, \mathbf{ä}\}$$

3.2.2 Alternation of stem final a and ä

A great number of surface base forms end in a short **a** or **ä**. Whatever the underlying segment is, it always corresponds to surface characters other than **a** and **ä** in plural and past tense stems. There are two kinds of variations (nominative vs. plural surface): **a-ø** and **a-o**. In bisyllabic stems the alternation is determined by the quality of the first vowel. If it is round the **a-ø** alternation is in effect. In trisyllabic nouns and adjectives the part of speech affects the choice (G. Karlsson 1978), but often the alternation depends on certain derivational suffixes and the pattern similarity to such suffixes.

At this stage I prefer to mark the type of the alternation in the lexical representation. Because deletion of a vowel is a more common alternation in Finnish morphology than rounding I use the surface **a** and **ä** in lexical representations where the zero alternation is in effect. The morphophonemes **Å** and **Ä** are used where the alternations **a-o** or **ä-ö** occur. It should be stressed that both alternations are morphophonological and grammatically conditioned (in the sense of Linell 1979). The alternation for plural and past tense is described by the following composite rule:

$$\begin{matrix} a & A & Å & ä & Ä \\ ø & ø & o & ø & ö \end{matrix} \quad <=> \quad \_\_ \quad "+" \quad \begin{matrix} i \\ = \end{matrix} \qquad (2a)$$

The "only if" or "=>" part of this rule limits the zero realization of e.g. **a** at the end of **koira** 'dog' to plural forms. In other words, if we have an instance of zero realization of **a**, then it must be followed by the plural "+i". This direction of the rule does not say anything about the default correspondence $\begin{matrix} a \\ a \end{matrix}$ or other possibilities. The exclusion of them is done by the "if" or "<=" part of the rule. In the following configuration the stem **koira** is followed by the partitive plural ending **&+ien**:

k	o	i	r	a	&	+	i	e	n
k	o	i	r	ø	ø	ø	i	e	n

According to rule (2a) this is the only permitted correspondence of the stem final **a**. (Other rules might even forbid that, although this is not true of any rule in this description.)

Stems ending in **a** or **ä** have a special alternation in comparatives or passives: the final vowel alternates with **e**. Passive endings are marked with a morphological feature denoted by the asterisk * which

triggers this alternation, e.g. **muista-** is the present tense stem of 'remember' and its passive is **muiste-taan** where the lexical representation of the passive ending is ***$DA:n**, see (2b). Comparative endings have a similar feature denoted by @. This triggers the **a-e** alternation, but only in bisyllabic adjectives, e.g. **kuiva** 'dry', **kuivempi** 'drier', see context (2c) of the composite rule below:

$$\begin{matrix} a \\ e \end{matrix} , \begin{matrix} ä \\ e \end{matrix} , \begin{matrix} A \\ e \end{matrix} , \begin{matrix} Å \\ e \end{matrix} , \begin{matrix} Ä \\ e \end{matrix} \quad <=> \quad \_\_ \quad "*" \qquad (2b)$$

$$2C \quad \_\_ \quad "@" \qquad (2c)$$

The environment "2C" covers the bisyllabic stem (ending in a single vowel) up to the consonant preceding the final vowel (for the definition of 2C see the section for the selection of alternative endings 3.2.9 below).

3.2.3 Alternation of productive stem final i

The productive and lexically most common type of nouns ending in **i** has a grammatically conditioned alternation in the plural stem, e.g **risti** 'cross', **risteissä** 'in the crosses'. I use the surface **i** as the final segment in the lexical representations of these words to reflect the productivity of this type. According to rule (3) this **i** corresponds to **e** in front of plural **i**. The rule restricts the correspondence exactly to this instance. The **i-e** alternation is not a phonotactic necessity and therefore it is classified as morphophonemic. Its motivation is more likely to provide plural forms that conform better to the general schema of plural forms (in the sense of Bybee and Slobin 1982).

$$\begin{matrix} i \\ e \end{matrix} \quad <=> \quad \_\_ \quad "+" \quad \begin{matrix} i \\ = \end{matrix} \qquad (3)$$

3.2.4 The morphophoneme E

There is a closed class of nominals having an **i-e-Ø** alternation, e.g. **kivi** 'a stone', **kiven** 'of a stone', **kivØissä** 'in stones'. The **i** occurs only in the nominative, **e** in the inflected singular stem and zero in the plural stem. I use a morphophoneme **E** in the lexical representation for this alternation. Many verbs have a similar **e-Ø**

variation. In these **e** occurs in the present tense stem and zero in past tense and conditional stems, e.g. **tul_en** 'I come', **tulØin** 'I came'. I use the same morphophoneme **E** in such verbs and in some alternation patterns as well (e.g. **nen-sE**).

The default correspondence id **E** <--> **e**, which occurs in the inflectional singular stems and in most verb forms other than the past tense and conditional. This is expressed as a dummy context restriction rule:

$$\begin{matrix}\mathbf{E}\\ \mathbf{e}\end{matrix} \quad \Rightarrow \quad — \tag{4a}$$

In two environments **E** corresponds to zero: (4b) when followed by a plural, past tense, or conditional **i** (cf. rule 2a), and (4c,d) in the consonant stems of nouns:

$$\begin{matrix}\mathbf{E}\\ \mathbf{Ø}\end{matrix} \quad <\!\!=\!\!> \quad — \quad \left[\text{"+"} \mid \text{"("}\right] \quad \begin{matrix}\mathbf{i}\\ =\end{matrix} \tag{4b}$$

$$\begin{matrix}\mathbf{E}\\ \mathbf{Ø}\end{matrix} \quad <\!\!= \quad — \quad \left[\mathbf{1} \mid \mathbf{2}\right] \tag{4c}$$

$$\begin{matrix}\mathbf{E}\\ \mathbf{Ø}\end{matrix} \quad \Rightarrow \quad \mathbf{V} \quad \left[\mathbf{Cn} \quad \begin{matrix}=\\ \mathbf{t}\end{matrix} \mid \mathbf{Cnhs}\right] \quad — \quad \left[\mathbf{1} \mid \mathbf{2}\right] \tag{4d}$$

where:

Cn = {**l, r, n**}
Cnhs = {**l, r, n, h, s**}

The symbols **1** and **2** are the distributional selectors for the partitive singular ending **1+tA** and the genitive plural ending **2+ten.** The formation of consonant stems occurs in cooperation between the ending and the stem:

(1) If these endings are used, the morphophoneme **E** must correspond to zero on the surface, otherwise the forms would be misinterpreted as imperatives or first infinitives (if interpreted at all), e.g. **pienetä** could not possibly be a partitive of **pienE** 'small' (it is the first infinitive of the verb **pienene** 'become smaller'). This effect is accomplished by the coercion rule (4c).

(2) Not all nouns ending in **E** require (or permit) a consonant stem, e.g. **kieltä** 'language (partitive sg.)' is the mandatory partitive, but ***lovta** 'hole (partitive sg.)' is unacceptable. The permission for consonant stem formation is phono-

tactically motivated, and it is formulated as restriction rule (4d). The class of words affected is quite small, and could also be interpreted as exceptional (with rule 4d being replaced by the use a distinct alternation pattern for the corresponding entries).

(3) The successful correspondence $\frac{E}{\emptyset}$ is in many cases a necessary condition for the selection of allomorphs **1+tA** and **2+ten**, otherwise rules (12b,c,d) will discard these allomorphs (in which case other alternatives would be used).

The presence of an **i** in the nominative of the **E**-nouns could be described either with an alternation pattern or with a rule. Both approaches have their advantages. In order to simplify the lexical representations I have chosen to use a rule that realizes **E** as i in front of the word boundary or the clitics. The following rule also accounts for second infitive forms like **luk<u>i</u>essa** 'while reading' from stem representation **lukE**:

$$\frac{E}{i} \quad <=> \quad \_\_ \quad \left[\# \mid \text{"\_"} \mid \left(\frac{=}{\emptyset}\right)^* \; e \right] \qquad (4e)$$

3.2.5 Vowel harmony

The opposition between harmonizing front (**ä, ö, y**) and back vowels (**a, o, u**) is neutralized in inflectional (and most derivational) suffixes. The harmony value of the suffix is determined by the last harmonizing stem vowel. Stems with only neutral vowels (**i, e**) receive front vowels in suffixes.

$$\frac{A}{a}, \frac{O}{o}, \frac{U}{u} \quad => \quad \frac{=}{Vb} \quad G^* \quad \_\_ \qquad (5a)$$

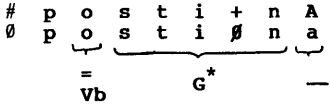
$$\frac{A}{ä}, \frac{O}{ö}, \frac{U}{y} \quad => \quad \left[\# \mid \frac{=}{Vf} \right] \quad G^* \quad \_\_ \qquad (5b)$$

where

Vf = {**ä, ö, y**}
Vb = {**a, o, u**}
Vbf = {**a, o, u, ä, ö, y**}

$$G = \frac{=}{-Vbf} - \#$$

The rules state that archiphonemes correspond to back vowels if there is a preceding back vowel (which may be followed by an arbitrary number of harmonically indifferent segments). Archiphonemes correspond to front vowels if there are no back vowels in the stem. Note that both vowel duplication and vowel harmony refer only to the surface context. If several endings are stacked at the end of the stem, the harmony value propagates from the stem to the first archiphoneme, and from the surface value of this to the next archiphoneme, etc. The following configuration arising from **posti** 'post office, mail', and essive singular **+nA** will serve as a simple example:



Only the correspondence $\begin{matrix}A\\a\end{matrix}$ is permitted. The context of (5a) is satisfied as shown in the figure. The term G^* can cover arbitrarily long stretches of the configuration as long as there are no surface phonemes affecting the harmony. The **ä** correspondence is not permitted because the G^* term cannot be extended to the beginning of the word (#) past the **"o"**.

Note that the automaton corresponding to the vowel harmony rule (Appendix A, automaton 8) is extremely simple, simpler than the rule above. This is a bit surprising, because the rule exhibits serious overlapping: the previous CP is a decisive part of the left context for the next occurrence of CP.

3.2.6 Plural i between vowels

The plural **i** corresponds to **j** between vowels. This is a morphologically conditioned alternation because otherwise **ViV** sequences are permitted, e.g. if they arise from consonant gradation. Thus we get **taian** 'of the spell' from **taika**, and **oion** 'I straighten' from **oiko-** (two exceptions: **aika - ajan** 'time' and **poika - pojan** 'boy'). The plural **i** rule affects only two endings **&+ien** and **7+iA**, because in all other plural endings the **i** is followed by a consonant. Thus we have **talojen** 'of the houses' and **kieliä** 'languages (partitive)'. Note that the condition "between vowels" refers to the surface. In the lexical representation **kielE7+iA** there would be a vowel on both sides, but the surface form is decisive. Thus the rule:

$$\begin{matrix} i \\ j \end{matrix} \quad <=> \quad \overset{=}{V} \quad "+" \quad — \quad \overset{=}{V} \qquad (6)$$

3.2.7 Deletion of n in front of possessive suffixes

All the endings of the genitive and illative terminate in **n**. This consonant is normally present but it is suppressed if a possessive suffix follows, e.g. **taloon** 'to (the) house' but **taloomme** 'to our house'. This rule is morphologically conditioned because in most instances there is no phonotactic reason for the absence of **n**, e.g. **taloonsi** would be phonotactically possible. A deletion rule is a simpler solution than positing the existence of parallel endings, one set without the **n** requiring the possessive suffix and another set with the **n** forbidding it:

$$\begin{matrix} n \\ \emptyset \end{matrix} \quad <=> \quad — \quad "/" \qquad (7)$$

(where the slash denotes the juncture in front of possessive suffixes). E.g the following configuration (arising from **kielE** 'language', **+n** 'genitive singular', and **/mme** 'possessive') conforms with (7):

k	i	e	l	E	+	n	/	m	m	e
k	i	e	l	e	ø	ø	ø	m	m	e

(Note that the features (+, /) correspond to zero by default and **E** to **e** according to rule 4a.)

3.2.8 Consonant gradation

The basic type of consonant gradation in Finnish is that voiceless stops alternate with "weaker" variants in certain contexts. At some earlier stages, the gradation was quite productive and phonologically conditioned but in the present synchronic state of Finnish neither of these claims holds. The quantitative gradation (**kk-k, pp-p, tt-t**) is still quite productive and it applies to neologisms, loan words, and even to most foreign names. The qualitative gradation (single stop alternations) is less productive because it is not consistently applied to neologisms, native names, and hardly ever to foreign names (cf. Yli-Vakkuri 1976). Thus it is necessary to mark at least some of the words in the lexicon for gradation.

I have marked all gradable stops in the lexical representations by using capital letters **K**, **P** and **T**. Leaving the double stops unmarked would have made the gradation rules more complicated and removed the possibility of handling foreign names violating the main rule. This convention also reflects the fact that gradation is active only at the end of the stem. The explicit marking thus relieves us from considering alternatives in places where the gradation is not applicable, e.g. the lexical representation **katTo** indicates that the initial **k** and the first **t** are constant.

I shall first give the rules for the distribution of weak grades of gradating stops and then the rule for selecting the correct grade.

The rules for **K** are the most complicated:

$$\begin{matrix} K \\ \emptyset \end{matrix} \quad => \quad V \left(Ch\right) \left(k\right) \text{ —} \tag{8a}$$

where

$$Ch = \left\{ l,\ r,\ h \right\}$$

This rule covers the normal quantitative gradation where **kk** alternates with a single **k**, and also other common instances such as **k** between vowels, e.g. **rako - raon** 'slit' and after consonants **l**, **r** and **h**. **K** corresponds to **ø** only in these environments. The exception rules below exclude some special cases where **K** corresponds to other surface characters, although these contexts are included in those of (8a):

$$\begin{matrix} K \\ ' \end{matrix} \quad => \quad V \ \langle V \rangle \text{ —} \langle V \rangle \quad => \quad \begin{matrix} K \\ -\emptyset \end{matrix} \tag{8b}$$

$$\begin{matrix} K \\ v \end{matrix} \quad => \quad C \ \langle Vu \rangle \text{ —} \langle Vu \rangle \quad => \quad \begin{matrix} K \\ -\emptyset \end{matrix} \tag{8c}$$

where

$$Vu = \left\{ u,\ y \right\}$$

(8b) reflects the ortographic convention that missing surface **k** must be indicated with an apostrophe if it occurs between identical vowels and is preceded by more than one vowel, e.g. **leuku - leu'un** 'knife', but **haka - haan** 'bolt'. (8c) covers the special case of **K** between two **u**'s where the first follows a consonant. There are only some five common words of this type, e.g. **puku - puvun** 'a dress'. (8d-f) deal with **K** after consonants:

$$\begin{matrix} K \\ g \end{matrix} \quad => \quad V \ n \text{ —} \tag{8d}$$

$$\begin{matrix}\mathbf{K}\\ \mathbf{j}\end{matrix} \;\Rightarrow\; \mathbf{V}\ \mathbf{Ch}\ \text{--}\ \left[\mathbf{e} \mid \begin{matrix}\mathbf{E}\\ \mathbf{\emptyset}\end{matrix} \mid \begin{matrix}\mathbf{E}\\ \mathbf{e}\end{matrix} \mid \mathbf{i}\ \$\ \mathbf{n}\right] \;\Rightarrow\; \begin{matrix}\mathbf{K}\\ \mathbf{-\emptyset}\end{matrix} \qquad (8e)$$

Rule (8d) describes the variation as in **sanko - sangon** 'bucket'. Rule (8e) is an exception to (8a). When **K** is followed by "e" and preceded by **l, r** or **h** the weak grade is **j**. (8e) overrides rule (8a) because no rule can give the definitive permission. Even a single coercion rule which is violated effectively forbids a correspondence. The last alternative (8e) refers to **in - imE** alternation, where there is no triggering **E** in the nominative stem, e.g. **poljin - polkimen** 'pedal'. Note that context restriction rules rather than composite rules were used because the strong alternantive is also possible depending on the appropriate context.

The rules for **P** are less complicated because the left contexts are easily formulated as simple disjoint expressions:

$$\begin{matrix}\mathbf{P}\\ \mathbf{v}\end{matrix} \;\Rightarrow\; \mathbf{V}\ \left(\mathbf{Cl}\right)\ \text{---} \qquad (9a)$$

$$\begin{matrix}\mathbf{P}\\ \mathbf{m}\end{matrix} \;\Rightarrow\; \mathbf{V}\ \mathbf{m}\ \text{---} \qquad (9b)$$

$$\begin{matrix}\mathbf{P}\\ \mathbf{\emptyset}\end{matrix} \;\Rightarrow\; \begin{matrix}=\\ \mathbf{V}\end{matrix}\ \left(\mathbf{Cl}\right)\ \mathbf{p}\ \text{---} \qquad (9c)$$

The rule (9c) refers to a surface vowel, because **P** is used in comparative ending and may be preceded by duplicated vowels.

The rules for **T** are equally straightforward with unique left contexts:

$$\begin{matrix}\mathbf{T}\\ \mathbf{d}\end{matrix} \;\Rightarrow\; \begin{matrix}=\\ \mathbf{V}\end{matrix}\ \left(\mathbf{h}\right)\ \text{---} \qquad (10a)$$

$$\begin{matrix}\mathbf{T}\\ \mathbf{<Cn>}\end{matrix} \;\Rightarrow\; \mathbf{V}\ \mathbf{<Cn>}\ \text{---} \qquad (10b)$$

$$\begin{matrix}\mathbf{T}\\ \mathbf{\emptyset}\end{matrix} \;\Rightarrow\; \begin{matrix}=\\ \mathbf{V}\end{matrix}\ \left(\mathbf{Cnh}\right)\ \mathbf{t}\ \left(\begin{matrix}=\\ \mathbf{\emptyset}\end{matrix}\right)^{*}\ \text{---} \qquad (10c)$$

where

$$\mathbf{Cn} = \{\mathbf{l, r, n}\}$$
$$\mathbf{Cnh} = \{\mathbf{l, r, n, h}\}$$

The selection of the weak or strong grade normally depends on whether the syllable is closed or open. There are, however, exceptions to such a rule: e.g. some plural genitives (**haraḵoiden**), partitives (**haraḵoita**), and possessive suffixes (**kauppamme**), infinitives of contracted verbs (**haḵata**) and some passive forms (**kudotaan**). In abstract

generative phonology this may be solved by reconstructing the open and closed syllables in the underlying forms (e.g. Wiik 1967). A more realistic way of describing this is to assume a triggering morphological feature to be present in endings which cause the gradation (F. Karlsson 1983 and Karttunen 1981). I have followed this line of reasoning and represented the feature with a **$** in front of endings that trigger gradation. I associate this property with the particular allomorphs (or quasimorphemes) of the endings rather than with the morphosyntactic features. This trigger has a limited range, typically not beyond one intervening vowel. The exact right context that triggers the gradation is the following:

$$\text{GRAD} = \left(\begin{matrix}\mathbf{V}\\=\end{matrix}\right)\ \left(\text{"+"}\ \ \mathbf{i}\ \left(\text{")"}\right)\right)\ \ \$$$

The weak alternatives may only occur with the triggering context to the right. The rules (8)-(10) leave two possibilities for each gradating stop: the strong alternative (**k**, **p**, **t**) and one of the weak alternatives. Thus we need a rule to exclude the strong one when gradation is applicable, and an another for excluding the weak one when gradation is blocked. Only truly gradable consonants in stems are coded as gradable by representing them as morphophonemes (**K**, **P**, **T**). These morphophonemes are also used in some endings, and these sometimes have an **s** to their left which blocks gradation. The selection is accomplished by coercion rules:

$$\begin{matrix}\mathbf{K}\\-\mathbf{k}\end{matrix}\ ,\ \begin{matrix}\mathbf{P}\\-\mathbf{p}\end{matrix}\ ,\ \begin{matrix}\mathbf{T}\\-\mathbf{t}\end{matrix}\ \ \text{<=}\ \ [\,-\mathbf{s}\,]\ \underline{\quad}\ \text{GRAD} \qquad \text{(11a)}$$

$$\begin{matrix}\mathbf{K}\\\mathbf{k}\end{matrix}\ ,\ \begin{matrix}\mathbf{P}\\\mathbf{p}\end{matrix}\ ,\ \begin{matrix}\mathbf{T}\\\mathbf{t}\end{matrix}\ \ \text{<=}\ \ \mathbf{s}\ \underline{\quad} \qquad \text{(11b)}$$

$$\underline{\quad}\ \ [\text{-GRAD}] \qquad \text{(11c)}$$

This set of gradation rules combined with the lexical marking of stops and the presence of trigger features in appropriate endings provides a general mechanism for handling consonant gradation. Normally, there is at most one gradation in a word form, but in the nominal forms of verbs as well as in adjectival paradigms there may be several. The following example has three gradating consonants, all of which correspond to the weak alternatives on the surface.

h	a	k	K	a	$	t	*	$	Z	T	U	$	+	i	m	P	A	$	+	i	s	s	A
h	a	k		a		t					u			i	m	m				i	s	s	a

The lexical level consists of five parts: the stem **hakKa** 'beat', an

entry from an alternation pattern **$t**, the second participle ***ZTU** ending, comparative ending **$+imPA**, and the inessive plural ending **$+issA**. The meaning of this word form is 'in those that have been beaten most'. Four entries have the trigger **$**, and all except the participle are within the active range.

It has been stressed that Finnish consonant gradation is morphologically conditioned. One could modify the description presented here to reflect the fact that the weak grade typically occurs in front of a closed syllable while the strong grade occurs in front of a closed one. Because exceptions occur in both directions, we need two features, say **$** and "^". Endings obeying the overall rule could be left unmarked, but all endings, which weaken although they should not, would still have the **$** feature, and the other exceptions would be marked with a "^". The gradation would function as before if we would modify the above definition of GRAD to be roughly:

$$\binom{V}{=} \quad \left("+" \quad i \left(")"\right)\right) \left[C \quad C \mid C \quad \# \mid \$\right]$$

This would match the ordinary closed syllables and the ones marked for weakening, but not those marked with "^".

3.2.9 Selection of alternative endings

There are different ways to describe the distribution of partitive, illative and plural genitive endings in nouns. One can establish declensions, each having typical sets of endings. This approach, which is adequate for many Indo-European languages, has been used e.g. in the Dictionary of Modern Standard Finnish (Nykysuomen sanakirja) and by Penttilä (1963). In Finnish this results in quite a number of declensions: more than eighty are listed in the dictionary. Recently, there has been a tendency to describe Finnish in terms of a smaller number of declensions (see e.g. F. Karlsson 1983b, section 8.2, for references and discussion).

The orthodox generativist approach does not see the problem as one of ending selection. The allomorphs are derived from a common underlying representation by quite abstract and opaque rules. In the two-level model, as in concrete phonology, rules of this type are excluded. The two-level model uses quasimorphemes (concept due to F. Karlsson 1977) which are not derived from each other (or from a common source) but stand as independent units representing identical morpho-syntactic properties. Quasimorphemes may contain archiphonemes and

they are intermediate between concrete allomorphs and invariant morphemic shapes or morphosyntactic features, which represent the meaning function of the endings. The endings in ending lexicons represent quasimorphemes.

The distribution is simplest to describe if we focus on the endings and regard them as responsible for the choice as was done by Setälä (1963). If an ending has a restricted distribution, I assign the ending an appropriate property, which I call a selector feature. The difference between this approach and the use of declensions becomes apparent when we notice that in Finnish each of the selective endings has a certain "tolerance of stem types", but most of the tolerances are distinct from each other. This is incompatible with the concept of declensions, where the domains should coincide.

In addition to selecting their environment the endings may affect the stem formation. One has to choose whether to assign the causing feature to the quasimorphemes or to the underlying morpholexical features. Vowel alternations are clearly dependent on the phonological representation (the presence of plural **i**). I have also chosen to assign the feature triggering the gradation to the quasimorpheme. The other alternative is to associate it with the underlying morphosyntactic features (e.g. F. Karlsson 1983b), and there are not too many points where it matters. There is no difference between the illative singular and plural, and the partitive singular in this respect. In the partitive and genitive plural there is a slight difference with respect to the quasimorphemes **itA**, **iden** and **itten**. The advantage gained by using the trigger in these quasimorphemes is not great, however, if we take the distribution into account. The triggering succeeds only in trisyllabic or longer stems and these are peculiar in the gradation of certain other forms: the plural forms of the illative and essive, and the comitative (cf. 3.1.6 and see F. Karlsson 1983b, section 10.3.1, and Itkonen 1966 for this "special gradation").

The tolerated environment can be expressed in most cases with two factors: syllable structure of the stem and the structure of what comes after the last consonant of the stem. The syllables may be counted using the following set:

$$\mathbf{V{:}} \;=\; \left\{ \mathbf{V} \;,\; : \right\}$$

and the following RPEs:

$$\mathbf{Q} \;=\; \left[\begin{matrix} = \\ \mathbf{C} \end{matrix} \;\middle|\; \begin{matrix} \mathbf{C} \\ = \end{matrix} \;\middle|\; \begin{matrix} \cdot \\ \boldsymbol{\beta} \end{matrix} \;\middle|\; \begin{matrix} = \\ \cdot \end{matrix} \right]$$

$$W = \begin{matrix} V: \\ V \end{matrix}$$

$$QW = Q^{*}\ W$$

$$WQ = W \left(\begin{matrix} V: \\ = \end{matrix}\right) \quad Q \quad Q^{*}$$

We shall define eleven elementary types of stems, one set for monosyllables (prefixed with **1**), one for strictly bisyllabic stems (prefixed with **2**), one set for stems with at least two syllables (prefixed with **2+**), and one type at least trisyllabic stems (prefixed with **3+**).

1CV = QW		**la** 'the note la'
1VV = QW W		**maa** 'country, earth'
1VØ = QW $\begin{matrix} : \\ Ø \end{matrix}$		**maita** '(some) countries'
2C = QW (W) Q Q^{*} $\left(\begin{matrix} V \\ Ø \end{matrix}\right)$		**kieltä** 'of a language'
2CV = 2C W		**talo** 'a house'
3C = 2C WQ^{*}		(an auxiliary definition)
2+VV = 2C WQ^{*} W **V**		**torstai** 'Thursday'
2+V: = 2C W $\begin{matrix} : \\ V \end{matrix}$		**suklaa** 'chocolate'
2+VØ = 2C W $\left[\begin{matrix} : \\ Ø \end{matrix} \mid \begin{matrix} V \\ Ø \end{matrix}\right]$		**suklaita** '(some) chocolates'
2+C = 2C WQ^{*} $\left(\begin{matrix} V \\ Ø \end{matrix}\right)$		**papadopoulos** '(proper name)'
2+V.V = 3C **V** . **V**		**autio** 'desert'
3CV = 3C **V**		**asteikko** 'scale'

These patterns classify stems, not lexemes. The stems are formed from roots and perhaps some alternation patterns, and they are subject to the effects of e.g. plural **i**. Plural stems are therefore never of types 1VV, 2+VV and 2+V:, and types 1VØ and 2+VØ may only be plural stems. Below, the distributions for endings are defined by listing the tolerated stem types for each of the sensitive endings.

The common ending for partitive singular is **-A**, and the lexical representation of it **!+A**. It can be attached only to at least bisyllabic stems which end in a single vowel:

! => [2CV | 3CV] —— (12a)

The **1+tA** quasimorpheme of the partitive singular may be attached only to monosyllabic stems, or to those which end either in a consonant or two vowels. The rule for the corresponding selector is:

1 => [1CV | 1VV | 2+C |
2+VV | 2+V: | 2+V.V] —— (12b)

The selector **2** is used in the **2+ten** variant of the genitive plural and in the quasimorpheme **2+iin** of the illative plural. (12c) permits these endings after all stems terminating in a consonant. (12d) excludes the stems ending in **t**. This slightly overgenerates **ten** genitives by permitting some archaic forms (so do most standard grammars):

2 => 2+C —— (12c)

$\underset{\{\ \}}{\mathbf{2}}$ <= **t** —— (12d)

The illative singular quasimorpeme **3+:n** is permitted only in stems with at least two syllables, if the stem ends in a short vowel:

3 => [2CV | 3CV | 2+V.V] —— (12e)

The second alternative for illative singular is **4+h:n**, which occurs after diphthongs (and the monosyllabic -V: and the marginal -V stems):

4 => [1CV | 1VV | 2+VV] —— (12f)

The third alternative is **5+seen**, which occurs after a double vowel in stems with at least two syllables:

5 => 2+V: —— (12g)

There are three quasimorphemes having an identical distribution: genitive plural **$6+iden** and **$6+itten** and partitive plural **$6+itA**. These occur after stems with simplified double vowels or diphthongs,

or after stems with at least three syllables and ending in a short vowel:

6 => [1CV | 1VØ | 2+VØ | 2+V:V | 3CV] —— (12h)

The partitive plural **7+iA** occurs after consonant stems and stems ending in a short vowel:

7 => [2+C | 2CV | 3CV] —— (12i)

The illative plural **8+ihin** occurs in all plural stems:

8 => [1CV | 1VØ | 2CV | 3CV | 2+VØ | 2+V.V] —— (12j)

The other illative plural **9+isiin** occurs only after a simplified double vowel:

9 => 2+VØ —— (12k)

There are two more endings with selective environments, the genitive plural endings **%+en** and **&+ien**, but these do not refer to the syllable structure of the stem. **%+en** is attached to inflectional singular stems which end in an unchanged single **i**:

$$\% \Rightarrow \overset{=}{C}\ i \text{ ——} \qquad (12l)$$

&+ien is attached to the plural stem (which is usually identical to the singular stem) and it occurs only after single rounded vowels, or after a consonant (where the intervening vowel (**E**, **a**) is deleted):

$$\& \Rightarrow \overset{=}{C}\ \overset{V}{\emptyset} \text{ ——} \qquad (12m)$$

$$\overset{=}{C}\ \left[\overset{=}{o} \mid \overset{=}{ö} \mid \overset{=}{u} \mid \overset{=}{y}\right] \text{ ——} \qquad (12n)$$

3.2.10 The morphophonemes in infinitive and passive endings

The verbal conjugation is complex but very regular in Finnish. There are some endings the shape of which varies according to the structure of the stem, but this variation is far simpler than in nominal inflection. In the verbal paradigm considerable variation is encountered only with respect to the passive and the so-called first

and second infinitive. The choice of the proper allomorph is made exclusively on the basis of the immediately preceding context, and there is only one possible choice. The selector mechanism used for nouns would of course be available for verbs too. I have preferred to establish two morphophonemes and appropriate rules to provide the correct surface forms of the endings because of the simpler nature of the phenomenon and because the surface alternation involves some assimilations along with the selection.

The morphophonemes are denoted by **Z** and **D** and they are present in the following endings:

***$Ziin**	"PAST PSS PE4"
***$ZtAvA**	"PCP1 PSS"
***$ZTU**	"PCP2 PSS"
***DA**	"PRES PSS"
(DA	"INF1 NOM"
De+ssA	"INF2 ACT INE"

The choice of such lexical representations is closely associated with the choice of underlying stems. E.g. the first **t** in **hakattu** or the only **t** in **hakata** could be attributed either to the stem (as in the present description) or to the ending.

The **Z** morphophoneme has a simple distribution. It corresponds to **t** after a short vowel, and as zero otherwise:

$$\begin{matrix} Z \\ t \end{matrix} \quad <=> \quad Q \ W \ \underline{\quad} \qquad (13a)$$

$$\begin{matrix} Z \\ \emptyset \end{matrix} \quad => \quad \underline{\quad} \qquad (13b)$$

These rules produce the desired surface forms, e.g. **kudottu** 'knitted', **juoØtu** '(that has been) drunk'.

The **D** morphophoneme corresponds to a consonant identical to a preceding resonant:

$$\begin{matrix} D \\ <Cr> \end{matrix} \quad <=> \quad <Cr> \ \underline{\quad} \qquad (13c)$$

This rule produces correct first infinitives like **purra** 'to bite'. The **D** corresponds to a **d** after two vowels or an **h** preceded by a vowel:

$$\begin{matrix} D \\ d \end{matrix} \quad <=> \quad W \ \left[W \mid h \right] \ \underline{\quad} \qquad (13d)$$

This rule accounts for forms like **nähdä** 'to see' and **jäädä** 'to stay'. The **D** corresponds to **t** after **s** or a single vowel in passives (marked with feature *):

$$\begin{matrix}\mathbf{D}\\ \mathbf{t}\end{matrix} \quad <=> \quad \left[\mathbf{s} \;\middle|\; \mathrm{Q}\ \mathrm{W}\ \text{"*"}\right] \quad \_\_ \qquad (13e)$$

This alternative covers forms such as **juosta** 'to run' and **kerätään** 'are collected (by someone)'. In other instances (i.e. after a **t** or a short vowel in forms not marked as passive), the **D** corresponds to zero:

$$\begin{matrix}\mathbf{D}\\ \mathbf{\emptyset}\end{matrix} \quad => \quad \_\_ \qquad (13f)$$

This option is for forms like **hakatØaan** 'are beaten (by someone)' and **kutoØa** 'to knit'.

3.2.11 Two assimilations

There are two verb endings with initial **n**, i.e. the active potential **ne** and the active second participle **nut, nyt.** The initial **n** assimilates to preceding **l, r** and **s:**

$$\begin{matrix}\mathbf{n}\\ \mathbf{<Cs>}\end{matrix} \quad <=> \quad \mathrm{W}\ \mathbf{<Cs>}\ \text{"("} \quad \_\_ \qquad (14a)$$

where

$$\mathbf{Cs} = \{\mathbf{l, r, s}\}$$

This rule produces forms like **purrut** 'one that has bitten'. The second kind of assimilation affects the underlying **t** at the end of the stem, and assimilates it to the following **n:**

$$\begin{matrix}\mathbf{t}\\ \mathbf{n}\end{matrix} \quad <=> \quad \mathrm{W}\ \_\_\ \text{"("}\ \mathbf{n} \qquad (14b)$$

This rule accounts for such forms as **hakannut** 'one that has beaten'.

CHAPTER 4
THE TWO-LEVEL PROGRAM

The two-level model has been implemented as a computer program written in the PASCAL programming language. This language was selected partly because of its wide availability and the portability of PASCAL programs, and partly bacause it provides sufficient means for expressing complex structures. PASCAL compilers produce an efficient code to the extent that the two-level program is quite fast and small enough to run even on a microcomputer (with a restricted lexicon). The program consists of eight modules and a main program for testing and demonstrating the model. Each module processes tasks and structures which are relatively independent of the other parts of the program. The overall structure of the program is outlined in the following diagram:

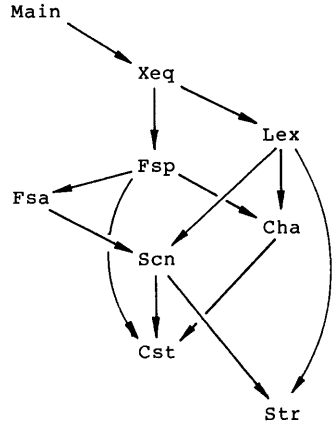


Figure 4-1

The arrows denote dependencies between the modules. Some modules use other modules by calling subroutines or functions that are defined in the used module. The arrows go from the user module to the one being used. The modules are arranged hierarchically so that the arrows only go downwards. Most modules are best characterized by the structures they contain and support:

Xeq This is the driver of the two-level program. It executes the analysis (or production) of the word forms making particular use of the lexicon (**Lex**) and the concrete pair set (in **Fsp**).

Lex Contains the lexicon system and provides incremental letter by letter access to it.

Fsp Constructs the concrete pair set (CPS) from the automata and aligns the automata to make them operational. It executes the array of automata during the analysis and production.

Fsa Contains the finite state automata which implement the two-level rules. It also provides access to the transitions and labels in the automata.

Cha Contains the definitions of the alphabet. Furthermore, it recognizes the surface characters and the names of various subsets.

Scn Scans the input extracting various kinds of items (letters, numbers, strings). **Scn** is used by the other modules to read in their input.

Cst Stores the character sets used in the program. Provides functions for testing membership.

Str Stores strings and provides access to and operations on them.

In the following, I shall describe the modules as far as they are relevant for the use of the two-level program as a linguistic model. For most modules only the specification of their logical structure is given without any details of the actual program. The specifications are written on a higher abstraction level free from the restrictions of PASCAL and the limited resources of present computers. The variables in a specification may use space that is not available in the computer's memory, and the procedure specifications may use logical quantification and mathematical induction instead of iteration. The actual implementation of the program must, of course, find a way to represent the abstract structures in the available computer memory, as well as a way to compute the desired results using a reasonable number of iterative steps. The algorithms of the central parts of the analy-

sis and producing routines are relevant to the interpretation of the two-level program as a model. Thus their implementations are given as well.

4.1 AUXILIARY MODULES

The modules for strings (**Str**), character sets (**Cst**), the input (**Scn**), and the alphabet (**Cha**) are not intended to be proper parts of the linguistic two-level model. They are auxiliary routines and structures for the more important parts of the model, and their presence is a consequence of the nature of the PASCAL language. If PASCAL would contain built-in facilities (like INTERLISP) for such entities, there would probably have been no need to include them as separate modules.

4.1.1 String module

The PASCAL programming language as it was defined by Wirth (e.g. Jensen & Wirth 1974) does not provide a built-in mechanism for storing and accessing variable length strings. The intrinsic type of PASCAL strings (PACKED ARRAY .. OF CHAR) is not suitable as such because all strings passed as parameters or used in comparisons should be of constant length for type matching. Effective storage of strings becomes, however, essential when we use extensive lexicons of more than ten thousand entries. To overcome these difficulties I have reserved a module for strings.

To specify the string module we sketch an abstract data type for strings. This means that the strings are specified indirectly by defining the functions and routines needed for handling them. The strings are abstract in the sense that the specificaton hides their concrete implementation. The specifications are written here in a modified PASCAL-like formalism which is close to the notation of the MODULA-2 programming language (Wirth 1979).

```
MODULE Str;

    EXPORT StrInit, StrNew, StrNull, StrLen,
        StrAppend, StrMove, StrPrint, StrEq;
```

```
TYPE string =  RECORD
                    Max: INTEGER;
                    Ch: SEQUENCE OF CHAR;
               END;
```

A string is a sequence of characters plus a maximum length to be the limit when the string variable is assigned new values. The net string of a string variable **s** is thus:

```
<s.Ch(1), s.Ch(2), ..., s.Ch(s.Ch.LEN)>
```

To be consistent such a string must never be longer than its maximum length. Such a requirement to be always satisfied is called an invariant and is a part of the specification:

```
INVARIANT
     FOR ALL x: string .
          x.Max >= x.Ch.LEN;
```

To store all the strings that the rest of the program will need, the Str module must have a suitable abstract variable. The strings will be inside the Str module and they are referred to by routines of this module. Thus a positive integer, the index of the string, is adequate for identifying strings, if they are stored as a sequence:

```
VAR  ssq: SEQUENCE OF string;
```

The string module contains routines and functions for establishing new strings, for their incremental building, for accessing their lengths and individual characters, and for appending, moving, comparing, and printing them.

4.1.2 Character set module

PASCAL provides a nice mechanism for sets, but the standard does not guarantee it to be sufficient for representing sets of CHAR even though some compilers permit this. Consequently, I have assigned a module for representing character sets in such a manner that it is feasible in all installations. We need a sequence of character sets as an abstract variable:

```
MODULE Cst;

        EXPORT CstInit, CstIncl, CstIn;

        VAR  csq: SEQUENCE OF SET OF CHAR;
```

To create a new set we use a procedure that appends a new (initially empty) set to the sequence csq:

```
PROCEDURE CstNew(VAR cs: INTEGER);
    csq:=csq & {};
    cs:=csq.LEN;
```

Characters may be added one by one to a set by using the following routine (which assumes that the set cs has been allocated with the above routine):

```
PROCEDURE CstIncl(cs: INTEGER; c: CHAR);
    ASSUME (0 < cs <= csq.LEN);
    csq(cs):=csq(cs) + {c};
```

To test whether a character belongs to a set or not, the following function is used:

```
FUNCTION CstIn(cs: INTEGER; c: CHAR);
    ASSUME (0 < cs <= csq.len);
    RETURN(c IN csq(cs));
```

4.1.3 Input scanner module

I have reserved a module for input to enable reading larger entities like quoted strings and identifiers. Also practical reasons, such as the need of sufficient error diagnostics speak for a separate abstract data type for input. PASCAL, like most programming languages, provides facilities for reading in items like integers. But PASCAL assumes that the input always contains an item of the expected type, otherwise the entire program is aborted. This leaves little room for programmatic diagnostics, which would be of real importance for locating errors in large and complicated inputs (e.g. the lexicon system and the automata).

The module Scn provides routines which are equivalent to the PASCAL statements RESET, GET, EOF and EOLN but keep track of the line and column numbers in addition to maintaining the full input line in a buffer. In this way the program can indicate the nature and location of possible errors. Otherwise this module is of little interest, and its specification is skipped here.

4.1.4 Alphabet module

A separate module (**Cha**) has been provided for the alphabet partly to permit future extensions for complicated alphabets. Finnish, like many other languages, has a simple alphabet where letters (even ä and ö) are expressed with single characters. As this is not the general case, **Cha** is intended to interface complicated alphabets to the common core of the two-level program. At present, the alphabet and even the names of its subsets consist of single characters.

The module **Cha** reads in the alphabet, the character to denote "null" in the rules, the "any" character (=), and the subset definitions. Both the lexicon and the automata use an internal character representation defined by **Cha**, and each character has, in addition to its internal code, an external "name". **Cha** contains the routines for translating external representations into the internal code and vice versa.

The **Cha** module provides two functions for the **Fsp** module. These test the relations between concrete characters and sets. Each of the symbols used in the automata labels is classified as belonging to one of the following four categories:

```
TYPE ChaType = (Null, Any, Concr, Set);
```

When the alphabet is initially constructed, the type of each symbol is recorded. I denote the class by **ChaClass(symb)**. The first function tests the inclusion/identity relation between symbols:

```
FUNCTION ChaMatch(c: Cha; symb: Cha): BOOLEAN;
    IF(ChaClass(symb) = Null) OR
       (ChaClass(symb) = Concr) THEN RETURN(c=symb)
    ELSE IF (ChaClass(symb) = Any) THEN
              RETURN(TRUE)
         ELSE RETURN(c IN symb);
```

The second function used by **Fsp** gives the class of a symbol pair (which is a column label in the automata):

```
TYPE ChaPairType = (ChaCC, ChaCO, ChaOC, ChaOO);

FUNCTION ChaPairClass(up, lo: Cha): ChaPairType;
    ...
```

The values of the function correspond to combinations "concrete-concrete", "concrete-set", "set-concrete" and "set-set" ("O" stands for an "open" symbol, i.e. a set).

4.2 RULE MODULES

The finite state module **Fsa** supports automata which simulate the two-level rules. The **Fsa** module is fairly ignorant of the two-level model, and its automata are almost identical to ordinary deterministic finite state automata. The **Fsp** module directs and coordinates the automata according to the two-level model. At present the two-level rules are hand-coded as finite state automata which are input to the program. The principles of the correspondence between automata and two-level rules is discussed in section 4.2.3. A compiler for automating this translation is planned and it is sketched in section 4.2.4.

4.2.1 Finite state module

The formal definition of a finite state automaton consists of a set of states, an input alphabet, a state transition function, an initial state, and the subset of final states. The common formalism for finite state automata is the so called state transition diagram where the states are represented as small circles and the transitions as arrows going between the circles. Before we discuss how to write the automata we discuss how represent them. Figure 4-2 shows an example of a hand coded automaton which corresponds to the rule (6) in chapter 3 (plural **i** between vowels):

$$\begin{matrix}\mathbf{i}\\ \mathbf{j}\end{matrix} \quad <=> \quad \begin{matrix}=\\ \mathbf{V}\end{matrix} \; + \; \underline{\quad} \; \begin{matrix}=\\ \mathbf{V}\end{matrix}$$

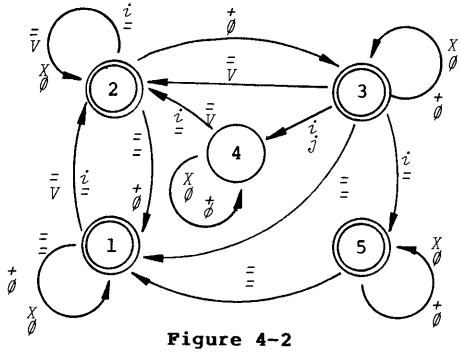


Figure 4-2

In the following, I always identify the states by numbers where the state 1 is, by convention, the initial state. Concentric circles mark some states as final states, i.e. they are permissible states for the execution to terminate. The only manner in which this differs from standard automata is in the input alphabet, which here consists of pairs rather than of single characters. The automata within the **Fsa** module are ignorant of their association to rules and descriptions to the extent that each automaton is stored as if it had an alphabet of its own. It is the task of the **Fsp** module to synchronize the automata into an effective rule component.

State transition diagrams are considered easy to understand and are therefore widely used for representing finite state automata. The diagrams have, however, the disadvantage of getting "spaghetti-like" when the size of the automata grows even to moderate dimensions. Furthermore, because diagrams are not feasible as a formalism for inputting automata to a program (using ordinary terminals), I represent the automata in tabular form as in figure 4-3.

	=	+	i	i	X	=
	V	0	=	j	0	=
1:	2	1	2	0	1	1
2:	2	3	2	0	2	1
3:	2	3	5	4	3	1
4.	2	4	2	0	4	0
5:	0	5	0	0	5	1

Figure 4-3

This table represents the same "plural i" automaton as figure 4-2. It consists of a matrix with a row for each state and a column for

each symbol (of the particular input alphabet of the automaton). The rows are preceded by a state number. Final states have a colon after the number whereas nonfinal states have a period. The columns are labelled with the pairs. The transitions are indicated by the state numbers in the transition matrix. If the automaton is in state **s**, we select row number **s** and look for the column corresponding to the current input symbol (i.e. pair). The number in this cell gives the next state. The missing transitions, i.e. the rejections, are indicated by zeroes in the matrix.

The **Fsa** module supports a sequence of finite state automata in tabular form to be used by **Fsp** and **Xeq**. The specification of **Fsa** is approximately as follows:

```
MODULE Fsa;

     IMPORT Scn, Cha, Str;
     EXPORT FsaLast, FsaSymbol, FsaNextState, FsaFinality
          FsaWidth;

     TYPE side = (up, lo);
          fs = RECORD
                  height, width: INTEGER;
                  lab: ARRAY  1..width  OF
                           ARRAY side OF Cha;
                  trans: ARRAY  1..height  OF
                           ARRAY  1..width  OF 1..height;
                  final: ARRAY  1..height  OF BOOLEAN;
               END;

     VAR  fsq: SEQUENCE OF fs;

     FUNCTION FsaLast: INTEGER;
          RETURN(fsq.LEN);

     FUNCTION FsaSymbol (m: INTEGER;
               sd: side; col: INTEGER): Cha;
          ASSUME((0 < m <= fsq.LEN) AND
               (0 < col <= fsq(m).width));
               RETURN(fsq(m).lab(col,sd));
```

```
FUNCTION FsaNextState(m: INTEGER;
          OldState: INTEGER; col: INTEGER): INTEGER;
     ASSUME((0 < m <= fsq.LEN) AND
          (0 < OldState <= fsq(m).height) AND
          (0 < col <= fsq(m).height));
          RETURN(fsq(m).trans(OldState,col));

FUNCTION FsaFinality(m, state: INTEGER): BOOLEAN;
     ASSUME((0 < m <= fsq.LEN) AND
          (0 < state <= fsq(m).height));
          RETURN(fsq(m).final(state));

FUNCTION FsaWidth(m: INTEGER): INTEGER;
     ASSUME(0 < m <= fsq.LEN);
     RETURN(fsq(m).width);
```

Because the size and the shape of the automata vary, an economical way to store them is to use pools for the transitions, labels, and finalities, and to allocate only the exact space needed for each automaton. This requires separate pointers indicating the beginning of the area of each automaton. It is evident that such details should be hidden from the rest of the program. The Finnish description consists of some twenty automata. Their total storage requirement is reduced to a tenth by pooling, i.e. to only a few thousand transitions and a few hundred states.

4.2.2 Alignment of the automata

The **Fsp** module interprets the automata that simulate the rules. It searches through the labels of the automata and builds the concrete pair set (CPS). Then **Fsp** aligns the columns of the automata by setting up a mapping from the CPS into the set of input symbols of each automaton. The routine **FspMove** uses this mapping to drive the automata properly when simulating the two-level rules.

The alignment is a complex process. Its purpose is to interprete the column labels as abstract pairs of the type discussed in section 2.3.2. Each pair <u,l> in the CPS is tested against the column labels of each automaton. It can match a label <U,L> in one of the following four ways (if it matches it at all):

1) both parts of the column label are concrete characters and u = U and l = L
2) U is a concrete character and u = U, and L is a set and l IN L
3) U is a set and u IN U, and L is a concrete character and l = L
4) both U and L are sets and u IN U and l IN L.

This order gives the precedence of the matches. If there are several matches of equal rank, the label of the leftmost column wins. Each pair is thus mapped to the leftmost column with the highest priority match. To ensure that there is at least one match for each concrete pair, each automaton must, by convention, have the label <"=","="> in their last column. (As this is the rightmost column, it is a default with the lowest priority.)

This explicit principle is useful when hand-compiling the rules as automata. It lets us use more specific column labels as exceptions to wider classes, e.g. <i,V> as an exception to <V,V>, and <i,j> as an exception to <i,"=">. The hierarchy of the label pairs is easy to understand as long as the labels correspond to subsets of CPS which are either disjoint or nested. It is recommended to avoid labels with partially overlapping subsets of the CPS, unless their transition columns are identical.

The formal specification of the **Fsp** module is as follows:

```
MODULE Fsp;

     IMPORT Cha, ChaType, ChaMatch, ChaLast,
          Fsa, FsaSymbol, FsaSide, FsaWidth;

     EXPORT FspStates, FspMarkPairs, FspAlignColumns,
          FspUpSelect, FspLoPairs, FspNextLo, FspLoSymbol,
          FspFeasible, FspMove, FspFinal;

     TYPE FspStates = SEQUENCE OF FsaState;
          pair =    RECORD
                         up, lo: Cha;
                    END;
```

```
VAR  CPS: SET OF pair;
     FspCol: ARRAY Cha, Cha OF
                    SEQUENCE OF FsaColumn;
     upper, lower: Cha;
     pi: INTEGER;
     psq: SEQUENCE OF Cha;

PROCEDURE FspMarkPairs;
     CPS := {p: pair .
               "p is a concrete pair in some fsa"};

EXPL "p is a concrete pair in some fsa":
          FOR SOME m: 1..FsaLast .
               FOR SOME k: 1..FsaWidth(m) .
                    p = <FsaSymbol(m,up,k),
                         FsaSymbol(m,lo,k)> &
                    ChaPairClass(p) = ChaCC;

PROCEDURE FspAlignColumns;
     FOR ALL <u,l>: pair .
          FspCol(u,l) := colsq .
               "colsq gives the columns for <u,l>";

EXPL up(m,k) = FsaSymbol(m,FsaUpper,k);
     lo(m,k) = FsaSymbol(m,FsaLower,k);

     "colsq gives the columns for <u,l>":
          colsq.LEN = FsaLast &
          FOR ALL m: 1..FsaLast .
               "colsq(m) is the column for <u,l>";

     "k is the column for <u,l>":
          "<u,l> matches column label k" &
          "<u,l> matching k is at highest priority" &
          "k is the leftmost match for <u,l>";

     "<u,l> matches column label k":
          ChaMatch(u,up(m,k)) &
          ChaMatch(l,lo(m,k));
```

```
"<u,l> matching k is at highest priority":
     FOR ALL k': 1..FsaWidth(m) .
          "<u,l> matches column label k'" ==>
               ChaPairClass(up(m,k'),lo(m,k')) >=
               ChaPairClass(up(m,k),lo(m,k));

"k is the leftmost match":
     FOR ALL k': 1..FsaWidth(m) .
          "<u,l> matching k' is at highest pr."
          ==> k <= k';
```

Now we have specified the crucial concepts "concrete pair set" and "alignment of the automata". Access to the CPS is provided by the following function:

```
FUNCTION FspFeasible(u,l:Cha): BOOLEAN;
     upper := u;
     lower := l;
     RETURN(<u,l> IN CPS);
```

The **Fsa** module does not contain any built in state variables, because the automata are operated from outside. The intermediate states of the automata are stored in **Xeq** as sequences of type **FspStates** defined in this module. **Xeq** uses two procedures:

```
FUNCTION FspMove(VAR: prev, new: FspStates): BOOLEAN;
     new := states .
               "states are next states from prev";
     RETURN("no error states in new");

EXPL "states are next states from prev":
          FOR ALL m: 1..FsaLast .
             states(m) =
               FsaNextState(m,FspCol(upper,lower,m));

     "no error states in new":
          FOR ALL m: 1..FsaLast .
               new(m) > 0;

FUNCTION FspFinal(VAR ss: FspStates): BOOLEAN;
     RETURN("all final states in ss");
```

```
EXPL "all final states in ss":
          FOR ALL m: 1..FsaLast .
               ss(m) > 0;
```

The **Xeq** module could have been almost symmetric with respect to the production and analysis of word forms. A few routines have been added to avoid an unnecessary and inefficient loop of the type:

```
FOR lo := 0 TO ChaLast DO
     IF FspFeasible(up,lo) THEN ...
```

With the additional routines only the pairs in CPS will be processed:

```
FspUpSelect(up);
WHILE FspLoPairs DO
BEGIN
   IF FspMove(...) THEN ...(...,FspLoSymbol,..);
   FspNextLo;
END;
```

These access routines are specified as follows:

```
PROCEDURE FspUpSelect(u: Cha);
     psq :=  "psq is a sequence of the elements of
               {l: Cha . <u,l> IN CPS & p.up = u} ";
     pi := 1;

FUNCTION FpsLoPairs: BOOLEAN;
     RETURN(pi <= psq.LEN);

PROCEDURE FspNextLo;
     pi := pi+1;

FUNCTION FspLoSymbol: Cha;
     RETURN(psq(pi).lo);

EXPL "x is a sequence of the elements of set":
               "elements of sequence x in set" &
               "set elements in sequence x" &
               "no identical elements in sequence x";
```

```
"elements of sequence x in set":
     FOR ALL j: 1..m .
          x(j) IN set;

"set elements in sequence x":
     FOR ALL y IN set .
          FOR SOME j: 1..m . y = x(j);

"no identical elements in sequence x":
     FOR ALL j,k: 1..m .
          (x(j) = x(k)) ==> (j = k);
```

In the actual implementation, the sequence for each upper character is computed only once, and the calls to these routines only move pointers within the lists.

4.2.3 Coding rules as automata

Let us study the composite rule (6) in chapter 3 which describes the correspondence of the plural i to a j on the surface:

$$\begin{matrix} i \\ j \end{matrix} \quad <=> \quad \begin{matrix} = \\ V \end{matrix} \quad "+" \quad \underline{\quad} \quad \begin{matrix} = \\ V \end{matrix}$$

It was claimed above that the state transition diagram in figure 4-2 and the tabular automaton in figure 4-3 (both in section 4.2.1) represent the automaton corresponding to this rule. In the following we discuss how one can transform rules into such automata by hand-compilation. First, we have to identify the alphabet for the automaton. As seen in the **Fsa** module, the input alphabet consists of five possible symbols (which are character pairs):

$$\begin{matrix} = & + & i & i & X & = \\ V & \emptyset & j & = & \emptyset & = \end{matrix}$$

The last one is a default pair representing all other pairs which are not mentioned. The pair $\begin{matrix} X \\ \emptyset \end{matrix}$ represents the morphological features.

For **Fsp**, however, the input alphabet of the automaton will be CPS. This will be achieved by expanding each label which represents an abstract pair to list all its members. In this expansion the priority scheme of the previous section is followed, and thus a concrete pair overrides a more abstract one, thus e.g. $\begin{matrix} i \\ j \end{matrix}$ is not included in $\begin{matrix} i \\ = \end{matrix}$. The

original smaller alphabet will serve as column labels as we write the automata, the expansion is needed for the proper interpretation of the automata.

The definition of states and transitions proceeds in stages. First one establishes states and transitions which recognize the left context of the rule:

	=	+	i	i	X	=
	V	0	=	j	0	=
1:	2				1	
2:		3			2	
3:					3	

Note the two last columns which usually are part of every automaton. The former implements the convention that features are skipped between any two symbols, and the latter guarantees that every pair refers to some column in the automaton. When we arrive at state 3, we have recognized the LC of the rule. Not until then are we ready to encounter $\begin{matrix} i \\ j \end{matrix}$ or other pairs with a lexical i:

	=	+	i	i	X	=
	V	0	=	j	0	=
1:	2				1	
2:		3			2	
3:			5	4	3	
4.					4	
5:					5	

The recognition of LC CP leads to a nonfinal state 4, because CP is not permitted unless RC is also satisfied. Complementary pairs will lead to state 5, which is final, but must exclude the possibility of RC. Thus, in state 4 we must have the vowel, and in 5 we must not (transition 0 marks the rejection and "*" acceptance):

	=	+	i	i	X	=
	V	0	=	j	0	=
1:	2				1	
2:		3			2	
3:			5	4	3	
4.	*	0	*	0	4	0
5:	0	*	0		5	*

Now, the configuration does not have to start with the LC, the sequence LC CP RC may start at a later stage. The automaton must therefore be able to restart appropriately (cf. Aho & Corasick 1975). If we encounter something else in state 1, we remain there. If we encounter e.g. another vowel in state 3, we fall off the track, but still know that we have seen a vowel, and thus go to state 2. (The CP may not, however occur prematurely.) In the same way the vowel completing the RC will be a valid part of the left context:

	=	+	i	i	X	=
	V	0	=	j	0	=
1:	2	1	2	0	1	1
2:	2	3	2	0	2	1
3:	2	3	5	4	3	1
4.	2	4	2	0	4	0
5:	0	5	0	0	5	1

Now we have completed the automaton and can see how it works. Its input is a configuration, e.g. a correct one:

Input:	t	a	l	o	&	+	i	e	n	
	t	a	l	o	ø	ø	j	e	n	
State:	1	1	2	1	2	2	3	4	2	1

The current state number is shown after each pair recognized. Note the restarting both before the actual LC is reached as well as after the final **e**.

The part of the automaton used above corresponds to the context restriction "=>" part of the rule. The coercion part "<=" is activated if we feed the automaton with an incorrect configuration:

Input:	t	a	l	o	&	+	i	e	n
	t	a	l	o	ø	ø	i	e	n
State:	1	1	2	1	2	2	3	5	0

4.2.4 Sketch of a rule compiler

Hand-coding the rules as automata has been proven possible, and several descriptions have been made with this technique. The need for a rule compiler is, however, obvious. The compiler would allow one to write descriptions using the rule formalism, and would result in more readable (and therefore more accurate) descriptions. The model would

also be easier to apply to new languages.

The automatic compilation is not an entirely new idea. To name but one example, an algorithm for compiling finite state automata to recognize regular expressions were presented in 1972 by Aho and Ullman 1972 (Aho & Ullman 1972, p. 255-258). Kaplan and Kay (1981) provided a rule compiler in their systen called "morphographemic grammar", which has geatly influenced the present two-level model.

If we could express the accepted configurations for each elementary rule (and especially each bunch of "=>" rules referring to a single CP) as a regular pair expression, the compilation could proceed along the lines of Kaplan and Kay. Coercion rules "<=" turn out to be easier, because they may be compiled into automata at the level of elementary rules. A rule:

$$\begin{matrix} x \\ y \end{matrix} \quad <= \quad LC \quad \_\_ \quad RC$$

can always be expressed through its complement as:

$$- \left[.. \quad LC \quad \begin{matrix} x \\ -y \end{matrix} \quad RC \quad .. \right]$$

even if the CP occurs within its context expressions. Automata, which are in a conjunctive relation, can be merged with simple techniques if desired (cf. Karttunen 1983).

Restriction rules "=>" are equally simple to express as RPEs as long as there is only one rule referring to a character pair:

$$- \left[[.. \quad -LC \quad CP \quad ..] \mid [.. \quad CP \quad -RC \quad ..] \right]$$

Combining these in the general case, where the context may contain arbitrary many occurrences of CP, seems to be difficult. One cannot exclude the possibility of at least one CP being in the LC, or else there would be no way to express e.g. the propagation of vowel harmony in a single rule. If some general restriction on the occurrences can be accepted, there is a way to express the disjunction as a RPE, and therefore a way to compile these rules into automata. Finnish morphology would only need the possibility of having a single CP in the left context (as the leftmost symbol). With this restriction the RPE would be manageable. The automaton could perhaps also be reached through algorithmic modification of the elementary restriction automata.

The morphographemic grammar of Kaplan and Kay requires combining all automata into a single one before the system is operational. One important difference between the two-level model and their model is

that the two-level model is operational as soon as we have the separate automata. The joining is thus not crucial but the two-level program can be optimized if the number of independent automata is reduced (the speed of an automaton does not depend on its size).

It is not guaranteed that all automata can be combined into a single one of reasonable size. Certain types of rules have a multiplicative effect upon the size of the final automaton. All long distance dependencies are costly in this sense, e.g. the vowel harmony rule inevitably doubles the states. Similar penalties must be paid if left or right contexts consist of sequences or iterations rather than single characters. If one cannot reach reduction to a single automaton, one should partition the collection of the automata into a few subsets having similar contexts and join each such subset. Merging two similar automata may, in fact, result in an automaton with less states than the total number of states in the components. According to an experiment carried out by Lauri Karttunen (personal communication), the two-level automata of my Finnish description can be unified into a single automaton consisting of a few thousand states. (This corresponds to about half a million state transitions if a completely filled matrix would be used.)

4.3 LEXICON MODULE

The **Lex** module corresponds to the lexicon system of the two-level model. It contains the sublexicons which consist of entries for words and endings. The entries contain linkages defining the morphotactic structure. Because the two-level program is a model of the dynamic processes of recognition and production, the lexicon must have adequate concepts for retrieving the entries.

The lexicons were introduced and used in the previous chapters as if they were lists of entries. I call this version the external lexicon system because lexicons are written and given to the program in this format. Such a structure is not adequate for retrieval. Therefore I build another structure based on the same information. The internal lexicon system organizes the lexicons as tree structures giving the proper framework for incremental accessing. The external version of the lexicon system is specified as follows:

```
MODULE Lex;

    IMPORT Str,Cha, ...;
    ...
    CONST EndAlt = ...;

    TYPE Ent =      RECORD
                        repr: SEQUENCE OF Cha;
                        cont: CARDINAL;
                        info: Str;
                    END;
         Lex = SEQUENCE OF Ent;
         Alts = SEQUENCE OF CARDINAL;

    VAR  Lexs = SEQUENCE OF Lex;
         Conts = SEQUENCE OF Alts;

    INVARIANT
         "Conts in all entries are valid" &
         "Alts in Conts are valid" &

    EXPL
         "Conts in all entries are valid":
              FOR ALL i: 1..Lexs.LEN .
                   FOR ALL j: 1..Lexs(i).LEN .
                        1 <= Lexs(i,j).cont <= Conts.LEN;

         "Alts in Conts are valid":
              FOR k = 1..Conts.LEN .
                   FOR l = 1..Conts(k).LEN .
                        (1 <= Conts(k,l) <= Lexs.LEN)
                        OR (Conts(k,l) = EndAlt);
```

This defines the logical content of the external lexicon system as a list of lexicons each consisting of a list of entries. The continuation component in each entry is given as an index to the list of possible patterns (Conts). Each of these patterns is a list of sublexicon numbers or a special value EndAlt indicating that the word form may end in this entry. The structure of the external lexicon system is very similar to the format in which the lexicons are fed to the program.

The purpose of the internal version is to provide incremental

access functions to the lexicons. This system is seen as a sequence of trees consisting of nodes. Some nodes are connected to each other via branches. Nodes are defined according to the distinct prefixes in the lexical representations. The nodes and branches form letter trees (e.g. Knuth 1973, pp. 481-505).

First, we introduce a relational operator ..= denoting "is a prefix of".

```
EXPL "u ..=  s":
          FOR SOME v .   (u v = s);
```

With this operator we define the set of all distinct prefixes for entry representations in the lexicon(l) (the actual number of lexicons is denoted by Nlx):

```
EXPL PrefSet(l) =
        {p . FOR SOME l: 1..Nlx .
                FOR SOME j: 1..Lex(l).LEN .
                   p ..= Lex(l,j).repr}
```

These sets of distinct prefixes correspond to the nodes of the internal form of the lexicons. The following abstract variable stores the prefixes as a sequence (one for each lexicon) of sequences (representing the sets):

```
TYPE prefix: SEQUENCE OF Cha;

VAR  pf:  SEQUENCE OF
               SEQUENCE OF prefix;

INVARIANT
     FOR ALL l: 1..Nlx .
          "pf(l) is a sequence of the elements
               of PrefSet(l)";
```

The distinct prefixes serve as nodes in the abstract letter trees. The branches in the trees are defined with the following predicate:

```
EXPL branch(pf(l,i),pf(l',j),c):
               l = l' &
               pf(l,i) c = pf(l',j);
```

Thus there are branches only within a tree corresponding to one lexicon. There is a branch from a source node to a target node if the target prefix is equal to the source prefix followed by the label character. The branch predicate is false for nonexisting nodes.

The trees must start from some node. The initial node in each tree corresponds to the null string prefix, and is denoted by:

```
EXPL FirstNode(l) = (l,i) . pref(l,i) = NULL;
```

Partial and complete phonological representations of entries can be expressed using nodes, branches, and initial nodes. Next, the entries should be related with the tree stuctures. The entries (if any) associated with each node are:

```
EXPL EntSet(pf(l,i)):
        {e: Ent .  e.repr = pref(l,i)};
```

The branch linkage is not the only way of connecting nodes. The morphotactic structure was defined using continuations from complete representations to other lexicons. The list of references to lexicons, where the next entry might be, is given by a continuation class. The following general connection predicate defines the incremental access routines to the whole lexicon system:

```
EXPL conn(pf(l,i),pf(l',j),c,e):
        "a branch from pf(l,i) to pf(l',j)" OR
        "pf(l',j) is a continuation via e" OR
        "e is a final entry at pf(l,i)";

    "a branch from pf(l,i) to pf(l',j)":
        l = l' & e = NULL &
        branch(pf(l,i),pf(l',j),c);

    "pf(l',j) is a continuation via e":
        e IN EntSet(pf(l,i)) &
        pf(l',j) = FirstNode(l') &
        l' IN Conts(e.cont);

    "e is a final entry at pf(l,i)":
        e IN EntSet(pf(l,i)) &
        EndAlt IN Conts(e.cont);
```

The fact that the lexicon uses three different types of linkages is reflected already in the definition. The types can be recognized by characteristic combinations of null arguments of the predicate conn(n,n',c,e):

conn(n,n',c,0)
: Normal letter tree linkages, where e is null, the target node n' is nonnull, c is the label character.

conn(n,FirstNode(l'),0,e)
: Continuations to other lexicons according to a continuation class of entry e, where n' is one of the nodes FirstNode(i), and the label c is null, (e is of course nonnull).

conn(n,0,0,e)
: Completions of final entries, where a representation has been completed and the entry may be the last in a word form. Then the target node pointer n' is null, label c is null, and the entry e is nonnull.

The incremental access to the lexicon is accomplished through a few procedures and functions accessible from the **Xeq** module. The routines are used in cooperation to process all continuations from a given node **n**:

```
LexSelectSource(n);
WHILE LexMoreTargets DO
BEGIN
     IF LexViaEntry = 0 THEN
               (* process a branch *)
          ELSE IF LexViaNode = 0 THEN
               (* process a completion *)
          ELSE (* process a continuation *);
     LexNextTarget;
END;
```

Note that the node numbers have to be stored outside the **Lex** module (the actual program uses single numbers **n** instead of pairs (l,i)). This node number is used in calling **LexSelectSource** at a later time. The idea of these routines can be illustrated by setting up a few abstract variables:

```
VAR  targets: SEQUENCE OF RECORD
                              en: LexEntry;
                              lb: Cha;
                              tg: LexNode;
                          END;
     ti: INTEGER;
```

The heart of the definition is the routine LexSelectSource which forms the sequence of all connections from a given node:

```
PROCEDURE LexSelectSource(n: LexNode);
     targets := <t(1),t(2), ..., t(m)> .
                "t is a list of all connections from n";
     ti:=1;

EXPL "t is a list of all connections from n":
          "all t:s are connections" &
          "all connections are in t" &
          "all elements in t are distinct";

     "all t:s are connections":
          FOR ALL i: 1..m .
               "there is a t(i) conncetion from n";

     "there is a t(i) connection from n":
          conn(n,t(i).tg,t(i).lb,t(i).en);

     "all connections are in t":
          FOR ALL n': LexNode, c: Cha, e: LexEntry .
               conn(n,n',c,e) ==>
                    FOR SOME i: 1..m .
                         n' = t(i).tg &
                         c = t(i).lb &
                         e = t(i).en;

     "all elements in t are distinct":
          FOR ALL i,j: 1..t.LEN .
               (t(i) = t(j)) ==> i = j;
```

The routine to get the next connection just moves the target pointer one position forward:

```
PROCEDURE LexNextTarget;
     ti := ti+1;
```

The test for more targets is equally simple:

```
FUNCTION LexMoreTargets: BOOLEAN;
     RETURN(ti <= targets.LEN);
```

The components of the current element of targets can be accessed with the following functions:

```
FUNCTION LexTarget: LexNode;
     RETURN(targets(ti).tg);

FUNCTION LexViaLabel: Cha;
     RETURN(targets(ti).lb);

FUNCTION LexViaEntry: LexEntry;
     RETURN(targets(ti).en);
```

4.4 EXECUTION MODULE

The execution module **Xeq** uses the rules and the lexicon in order to analyze or produce word forms. The logical aspect of **Xeq** is the accomplishment analyses according to the definitions in chapter 2. In addition to satisfying the formalism the actual implementation of this module is also relevant when the program is evaluated as a model of human morphological recognition and production processes. The algorithm and the structure in the program are quite explicit, but any claims for isomorphism between it and the true phenomena must be modest and restricted to the overall strategies (cf. chapter 5).

4.4.1 Specification of the analysis routine

When analyzing a word form, the **XeqAnalyze** routine produces all feasible partial analyzes of the form and gives as output the complete ones. We shall specify the module as having an abstract variable holding the collection of these analyzes.

```
TYPE step =     RECORD
                     up, lo: Cha;
                     nd: LexNode;
                     en: LexEntry;
                     st: FspStates;
                END;
     trace = SEQUENCE OF step;

VAR  traces: SET OF trace;
     results: SET OF trace;

INVARIANT
     FOR ALL st: step .
          (st.en ≠ 0  ==>  up = 0) &
          (st.en = 0  ==>  (nd ≠ 0) & (up ≠ 0));
```

The task of the analyzing procedure is to assign proper values to traces and to results:

```
PROCEDURE XeqAnalyze(word: SEQUENCE OF Cha);
     traces := {tr: trace .
                    "tr is a valid partial analysis"};
     results := {tr: trace .
                    "tr is a valid complete analysis"};

EXPL "tr is a valid partial analysis":
          "lo components form a prefix of word" &
          "up components agree with the lexicon" &
          "configuration satisfies the rules";

     "tr is a valid complete analysis":
          "tr is a valid partial analysis" &
          "tr is a complete trace";

     "lo components form a prefix of word":
          tr(1).lo  tr(2).lo ... tr(tr.LEN).lo  ..=  word;

     "up components agree with the lexicon":
          "first step is the initial step" &
          "steps follow the connections";
```

```
"first step is the initial step":
     tr(1).nd = LexRootNode;

"steps follow the connections":
     FOR ALL i: 2..tr.LEN .
        conn(tr(i-1).nd,tr(i).nd,tr.(i).up,tr(i).en);

"configuration satisfies the rules":
     "all states feasible and successive";

"all states feasible and successive":
     "initial states all ones" &
     FOR ALL i: 2..tr.LEN .
          (FspMove(tr(i-1).st,stat) &
          stat = tr(i).st);

"initial state all ones":
     FOR ALL j: 1..Nfsa .
          tr(1).st(j) = 1;

"tr is a complete trace":
     "all surface characters used" &
     "automata of tr(tr.LEN) in final states" &
     "tr ends in a terminal conection";

"all surface characters used":
     InputString =
          <tr(1).lo, tr(2).lo, ..., tr(tr.LEN).lo>;

"automata of stp in final states":
     FOR ALL j: 1..Nfsa .
          fsq(j).final(stp.st(j));

"tr ends in a terminal connection":
     tr(tr.LEN).nd = 0  &
     tr(tr.LEN).up = 0;
```

4.4.2 Implementation of the execution module

This section discusses the implementation of the analysis and production routines. The implementation consists of concrete data structures and algorithms which are committed to certain strategies (such as "breadth first" parsing).

The **Xeq** module views the lexicon system as a set of letter-trees with branches and continuation connections. The **XeqAnalyze** routine can be visualized as pointing a searchlight to the lexicons. As letters of the input word form proceed, the searchlight is directed along the branches and connections. The spot follows only branches with feasible labels. The light spot is not just a point, it has some size. The analysis process is thus nondeteministic to some degree. This implies either parallel processing (in networks) or sequential checking of the possibilities (in computers).

Word entries are retrieved in this model without overhead arising from other entries with identical beginning but different continuation (their presence is not even noticed). Each input character causes a move in the direction of the searchlight and the set of possibly matching entries is restricted implicitly as the branches are taken. The processing between successive input symbols consists mostly of a transition in the rule automata.

4.4.3 Data representation

An implementation must implement its specification by declaring concrete variables that represent the abstract variables of the specification, and by giving the explicit algorithms needed for computing the required results. The concrete variables represent the abstract ones if there is a way to compute the abstract values from the concrete ones. The implementation has the following type and variable declarations:

```
TYPE
     XeqTaskInd = 0 .. XeqTaskMax;
     XeqTask = RECORD
                    XeqPos: INTEGER;
                    XeqCha: Cha;
                    XeqNod: LexNode;
                    XeqPrev: XeqTaskInd;
                    XeqEnt: LexEntry;
                    XeqSta: FspStates;
               END;

VAR
     XeqString: ARRAY [1..60] OF Cha;
     XeqLen: INTEGER;

     XeqQue: ARRAY [XeqTaskInd] OF XeqTask;
     XeqHead,
     XeqTail: XeqTaskInd;
```

The declared variables correspond to two entities: the input string to be analyzed (or used as a lexical string in the production), and a queue. The input string is stored as characters:

```
XeqString[1], ..., XeqString[XeqLen]
```

The characters are in the internal code defined by module **Cha.** The queue contains two parts delimited by pointers **XeqHead** and **XeqTail.** The active part of the queue consists of elements:

```
XeqQue[XeqHead], ..., XeqQue[XeqTail]
```

The history part of the queue records the elements removed from the active part:

```
XeqQue[1], ..., XeqQue[XeqHead-1]
```

Tasks are removed from the head of the queue and new tasks are entered to its tail. The routine **XeqEnque** enters tasks to the tail and **XeqDeque** removes them from head of the queue. **XeqLeft** tests whether there are tasks left in the queue. These routines are trivial.

Each task in either part of the queue represents a step of the specification. The components of a step can be computed from the

components of a task in a following way:

up	⟷	XeqCha	
lo	⟷	XeqString[XeqPos]	(if XeqPos > 0)
		null	(otherwise)
nd	⟷	XeqNod	
en	⟷	XeqEnt	
st	⟷	XeqSta	

The roles of **XeqCha** and **XeqPos** are exchanged in the production (thus **XeqCha** is always the deduced character, and **XeqPos** always points to a input character in **XeqString**).

Each **trace** in the specification was defined to be a sequence of **steps**. Such a **trace** can be computed for each task in the queue by following the **XeqPrev** links backwards until the task number 1 is reached. To be precise, the trace corresponding to a task number **k** is the sequence of steps corresponding to the following sequence of tasks:

$$\langle XeqQue[i_1], \ldots, XeqQue[i_n] \rangle$$

where

$i_1 = 1$ &
$i_n = k$ &
FOR ALL j: 1..n-1 .
    XeqQue[j+1].Prev = j

The set of all **traces** is the set of sequences according to the above definition for all tasks in the queue when the processing is completed.

4.4.4 The analyzing algorithm

The analysis routine **XeqAnalyze** (see figure 4-4 on the next page) starts by resetting the queue (**XeqReset**) to a single dummy task pointing to the initial node of the root lexicon. In this task all rule automata are set to the initial state (i.e. state 1). Thereafter the algorithm proceeds by focusing on the current task at the head of the queue. The task is first tested for being a complete analysis. The subset which was called "**results**" in the specification is constructed only implicitly by outputting the trace sequence for complete tasks.

For nodes representing partial analyses the routine tries all feasible connections to the next nodes in the lexicon system. If the branch label and the next surface character fit each other and conform to the rules, a new task is added to the queue. Finally, when all connections have been tested and processed, the head task is removed from the queue.

The test for completeness of the task implements the corresponding definition in the specification. All possible connections from the source node are ennumerated by the mechanism presented in the **Lex** module (**LexSelectSource**, **LexMoreTargets**, and **LexNextTarget**, see 4.7).

At each point of the analysis there are two possibilities: either the next pair in the configuration contains a real surface character, or the lexical character corresponds to a null character. This is reflected by two separate calls to **XeqAnalTask** which evaluates the feasibility of the connection by advancing the rule automata.

The **XeqAnalTask** routine (see figure 4-5) uses the function **LexViaEntry** to find out what kind of a connection is being proposed (c.f 4.7). This loop is not entered for terminal tasks (those where **XeqNod** = 0). Thus there are two types of connections: ordinary branches and continuation patterns. Branches are checked for the rule conformance (by **FspMove**) before a new task corresponding to the target node is entered into the queue. Linkages for continuation patterns are processed by the simple entering of a dummy task which refers to the initial node of the corresponding lexicon.

The analysis thus proceeds using a "breadth-first" strategy which produces all alternatives in parallel. A "depth-first" strategy would have been equally simple to implement. It would have pursued one alternative as far as possible before considering the other ones. For syntax there has been some discussion on these strategies and their mixtures (see e.g. Kaplan 1972, Marcus 1980). The depth-first strategy sometimes reaches the correct solution very quickly, sometimes very slowly. I prefer the breadth-first strategy because there are no drastic variations in the speed of human morphological processing. The breadth-first strategy is also easier to test against the requirement of processing in real time (bounded processing between input characters). Furthermore, the breadth-first strategy is easier to equate with nonsequential models like electronic circuitries or neural networks.

```
PROCEDURE XeqAnalyze;
     VAR  p: INTEGER;
          c: Cha;
     BEGIN
        XeqReset;
        WHILE XeqQueLeft DO
        BEGIN
           WITH XeqQue[XeqHead] DO
           BEGIN
              p:=XeqPos;
              c:=XeqString[p+1];
              IF p = XeqLen THEN
                        (* all surface characters used *)
                IF FspFinal(XeqQue[XeqHead].XeqSta) THEN
                        (*automata in final states *)
                  IF XeqNod = 0 THEN
                        (* may end here *)
                  BEGIN
                     XeqOutPrep(XeqHead);
                     OutProc; (* a complete analysis *)
                  END;
              IF XeqNod <> 0 THEN
              BEGIN       (* analyze all connections *)
                 LexSelectSource(XeqNod);
                 WHILE LexMoreTargets DO
                 BEGIN
                    XeqAnalTask(p,0);(* null onsurf. *)
                    IF LexViaEntry = 0 THEN
                        XeqAnalTask(p+1,c); (* nonnull *)
                    LexNextTarget;
                 END;
              END (* XeqNod <> 0 *);
           END (* with *);
           XeqDeque;
        END (* XeqLeft *);
     END;
```

Figure 4-4

```
PROCEDURE XeqAnalTask(p: INTEGER; c: Cha);
     BEGIN
        IF p <= XeqLen THEN
        BEGIN
           IF LexViaEntry = 0 THEN
           BEGIN
              IF FspFeasible(LexViaLabel,c) THEN
                 IF FspMove(XeqQue[XeqHead].XeqSta,
                            XeqQue[XeqTail].XeqSta) THEN
                    XeqEnque(p,LexViaLabel,
                             LexTarget,LexViaEntry);
           END ELSE
           BEGIN
              XeqQue[XeqTail].XeqSta:=
                   XeqQue[XeqHead].XeqSta;
              XeqEnque(p,LexViaLabel,
                       LexTarget,LexViaEntry);
           END;
        END;
     END;
```

Figure 4-5

4.9.3 The producing algorithm

The procedure **XeqProduce** (figure 4-6) implements production without using the lexicon. The routine **XeqProduce** assumes that its input (**XeqString**) contains a concatenation of lexical representations. The production starts in the same way as the analysis, by resetting the queue to a dummy initial task. The production proceeds character by character by accessing the concrete pair set in the **Fsp** module (routines **FspUpSelect**, **FspNextLo**, and functions **FspLoLeft**, and **FspLoSymbol**). Before any task at the head of the queue is processed, it is tested for completeness. The two-level rule formalism allows several alternative realizations and **XeqProduce** computes them all. The rules of the Finnish description are, however, unambiguous.

```
PROCEDURE XeqProduce;
     VAR  pos: INTEGER;
          c: Cha;
     BEGIN
        XeqReset;
        WHILE XeqQueLeft DO
        BEGIN
           pos:=XeqQue[XeqHead].XeqPos+1;
           IF pos>XeqLen THEN
           BEGIN
              IF FspFinal(XeqQue[XeqHead].
                          XeqSta) THEN
                 BEGIN
                    XeqOutPrep(XeqHead);
                          OutProc;
                 END;
           END ELSE
           BEGIN
              c:=XeqString pos ;
              FspUpSelect(c);
              WHILE FspLoPairs DO
              BEGIN
                 IF FspMove(
                       XeqQue[XeqHead].XeqSta,
                       XeqQue[XeqTail].XeqSta)
                       THEN XeqEnque(pos,FspLoSymbol,0,0);
                 FspNextLo;
              END;
           END;
           XeqDeque;
        END;
     END;
```

Figure 4-6

CHAPTER 5
INTERPRETATION OF THE TWO-LEVEL MODEL

This chapter discusses the interpretation of the two-level model from the standpoint of various frameworks: (1) from that of its use as a traditional autonomous linguistic model, (2) from that provided by the use of external evidence (such as data from child language acquisition), and (3) from that of its status as a dynamic model of morphological perception and production.

The two-level model consists of mathematical definitions and a computer program. There are, of course, many possibilities of designing explicit models of such complicated phenomena: by building mechanical devices, by setting up mathematical equations, by constructing electric circuits, etc. (for a discussion of various types of models see e.g. Black 1962, chapter XII).

Programs are flexible enough to be modified during the course of the study, and the explicitness of computer models makes thorough testing of models and descriptions possible. The possibility of extensive testing deserves emphasis because the human mind has great difficulties in managing formal interactions in systems consisting of hundreds or thousands of items (such as linguistic rules). A computerized model will readily reveal unanticipated interactions and lead to iterations to correct the model.

Not all programs are attractive as models. Programs without an underlying theory and explicit concepts are useless in this respect. Therefore, considerable effort was devoted in chapter 4 to relate the two-level program to the formalism and theory presented in chapter 2. In this way the program, and especially its specification, is an integral part of the model.

5.1 Autonomous interpretation

The formalism of the two-level model (chapter 2) and the specification of the two-level program (chapter 4) form an explicit morphological theory. As long as we take this theory and associated two-level descriptions as such, we may interpret them as autonomous models of morphology. They provide a framework for describing lexical elements, morphotactic structure, and rules. Together, these components

form morphological descriptions. If we ignore the dynamic aspect of the program, i.e. the implementation of the central procedures, we have an autonomous model of the morphology. If we would go a step further and omit all arguments based on the productivity of certain word types and variations, we would arrive at a "pure" autonomous model which describes the morphology as an independent system of relations between forms.

Obviously, such a subset of the two-level model is incommensurable with other autonomous morphological theories, e.g. with abstract generative phonology. It competes with them, but there are no explicit criteria for evaluating one as superior to the other. The two-level model does not propose a counterpart to the evaluation metric of early generative phonology. Descriptions should first of all "fit the facts". Selection among competing (equally fitting) descriptions should perhaps not be done according to explicit formal criteria alone. Informal criteria like brevity and transparency may be more appropriate, but brevity should not be exaggerated to an extent where transparency is compromised.

This view is a close parallel to the now widely accepted position in evaluating programs. There used to be two goals for programming: (1) to make programs as brief as possible or (2) to make them capable of being run as rapidly as possible. Later on, it was realized that it is even more important to make programs correct, and this could be done only if they are readable. There is no conclusive formal criterion for readability of programs, just as there is none for books. Therefore, it has become apparent that the goal in programming should be analogous to good style in writing, which is not (even in principle) expressable as a plain formal criterion (Kernigham & Plauger 1974).

Before any informal comparisons between the two-level model and other morphological models are made, the differeces in their domains and goals must be stressed. The two-level model does not attempt to remove all redundancies from the lexicon and rules. It regards redundancy as an integral part of the language system which is present especially in the dynamic processes.

The two-level formalism is no more complicated than the generative one. Both rely on the concept of regular expressions. There is a trade-off between the rule ordering of the generative formalism and alternation patterns in my model. The descriptive power of rule ordering is great and it adds to the complexity of the generative model. Parallel rules in the two-level model are inherently simpler. They and the alternation patterns have an intimate connection with regular

grammars and finite state automata, which are regarded as the simplest formalisms available.

Even if generative rules can be expressed as finite state automata (Kaplan & Kay 1981) there remains a quantitative difference. The ordered generative rules result in one very large automaton, whereas the parallel rules of the two-level model correspond to a set of small separate automata. The difference in their total size might be several magnitudes.

Two-level descriptions are brief: the Finnish description consists of less than 10 pages in the input format of the program (where the rules are as automata and the lexicon system is in the external format). It is hard to make fair comparisons to other models, but it seems that two-level descriptions are reasonably brief, if we remember that it is an (almost) full description (of all forms of all inflectional types).

The readability of descriptions is difficult to estimate. Apparently simple formal statements may have combined effects which are not readily understood. Especially in the generative formalism, there are often unanticipated interactions between rules, and even if such are detected, they may be difficult to fix. An attempted correction may result in a leak somewhere else. The concreteness of the two-level model restricts the possibility of interactions. There are only two representations without any intermediate stages. The rules are almost independent of each other because there is only one common environment for all rules.

5.2 The role of external evidence

"External evidence" consists of several kinds of language related more or less behavioral phenomena which can be directly observed. Basically autonomous models may use such evidence as an evaluation criterion for finding preferred or optimal descriptions. Of the numerous types of external evidence, only errors in the production of word forms (especially by children or aphasic patients) are discussed in the present work (section 5.4).

The use of external evidence as an evaluation criterion is characteristic of "concrete" and/or "natural" morphology. It has arisen from a failure to agree upon a common internal evaluation metric (such as feature counting schemes), and it is used as a criterion supplementing the information obtained utilizing autonomous criteria.

Linell (1979) characterizes the notion "psychological reality" basically in terms of external evidence. According to him, a psychologically real description tries to model real aspects of language knowledge and use. Models of this knowledge are bound to propose some internal representations of linguistic units, and to define morphology in reference to these representations. The models may commit themselves to the actual dynamic processes of word recognition and production, or they may define the relations only logically, without any direct reference to such processes. The discussion of the former aspect is delayed until section 5.5. The latter approach is more common, and it leads to a model which is still essentially equivalent to a specification of a program (like the autonomous model). The only difference is that the internal representations and some of the formal relations are here open to external evaluation.

Linell (1980) defines such models to be performance grammars or "behavioral competence grammars". They do not deal with real performance processes, but they can make restricted commitments to the processual aspect of recognition and production. A model has to account for data from e.g. performance errors in child language acquisition and in aphasic speech. Performance errors may be interpreted as being some intermediate phases or revealing near associations. Adequate models of production and recognition processes should account for both the presence and and the systematic absence of various errors.

Restricted statements of the overall processual aspects of a model can be called the strategy of the model. It represents the gross order of operations as well as the principles used to select from the alternatives. If we return to the two-level program, and particularly to the implementations of the procedures for analysis and production, we see that the strategies of the two-level model are present in these implementations. The procedures contain a multitude of irrelevant details, but they also express the overall strategies: left-to-right breadth first procession and incremental look-up from tree-shaped lexicons.

5.3 The relation between various interpretations

At this point we are ready to sketch the overall situation in the interpretation of the two-level model. The two-level program, together with the associated formalism, contains three (partially overlapping) parts which have the following characteristic interpretations or roles:

(1) The specification and the formalism (without any reference to the implementation) is interpreted as an autonomous morphological theory. The specification together with a description of a particular language form an autonomous competence grammar.

(2) The specification, the formalism, and the implementation of the central procedures (XeqAnalyze and XeqGenerate) form a performance model. The processual characteristics of this are subject to restricted feasibility evaluation according to criteria of processing complexity and testing against empirical observations. A two-level description in the framework of this performance model is a performance grammar subject to evaluation according to performance errors and similar external evidence.

(3) The full implementation of the program, where all procedures are included, is evaluated only computationally, and is not interpreted as a linguistic model as such. The methods used in it are, however, relevant for interpreting the performance model because of the intimate relation prevailing between specifications and implementations. The complexity of the implementation can thus be used in interpreting the processual claims of the performance model.

These three interpretations of the two-level program together with the morphological component of a speaker-listener form a configuration:

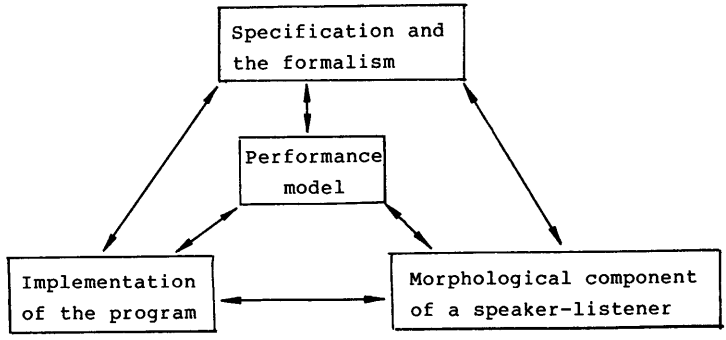


Figure 5-1

There are many important relations between the components in this figure:

(1) The performance model presents an analogy model of the real morphological component to be described, and it is claimed to be isomorphic to certain important aspects of the real component.

(2) The implementation simulates the real component without claiming isomorphism to it.

(3) The specification provides the concepts for discussing the other components.

(4) The implementation implements the specification and the performance model.

5.4 Interpretation of internal representations

The lexical representations, alternation patterns, and rules of the two-level description of Finnish have to be related to performance data. The evaluation of internal representations and rules consists here of explaining and predicting the presence and absence of various kinds of performance errors. The primary design goal of the two-level model has been to model the process of producing and recognizing word forms. The obvious extensions needed in order for it to cover the phenomenon of productivity are sketched below.

Lexical redundancy can here be understood as the regularities in word inflection which restrict the free exploitation of possibilities. Lexical redundancy establishes types of word entries which have a similar gestalt and an identical morphological behaviour. F. Karlsson (1983b) describes the lexical redundancy of Finnish starting from the base form of each word entry. The nominative singular is used as the base form for nouns, and the strong vowel stem for verbs. The morphology of verbs and most nouns can be stated according to the patterns of their base form. For some types of nouns the base form is not sufficient, an additional inflected singular and/or plural form is needed. F. Karlsson denotes this kind of additional information by marking the less productive base forms with an apostrophe.

In the two-level lexicon, regularities are explicitly marked in the word entries. Some markings are expressed as morphophonemes in the

representation, some as continuation classes. This corresponds to the view that the two-level lexicon is a dynamic lexicon of internalized entries. The one-time operations of creating or modifying entries are performed by an external component. Productivity could be described in the framework of the two-level model by establishing a productivity mapping which maps the base form of a new word to the possible word entries weighting the alternatives according to the productivity of the corresponding inflectional type. The entry formation seems to be prone to errors, especially in child language acquisition.

Far from all new words are initially encountered in their base form, although the nominative is the most common case for most nouns and the strong vowel stem is present in the most frequent verbal forms. Thus, in practice, the base form must sometimes first be deduced before productivity mapping can be applied. Bybee and Slobin (1982) have proposed the use of so called schemas for the analysis of such inflected forms. By combining schemas with a productivity mapping, it is possible in principle, to account for the ability of the speaker to inflect foreign names and new words.

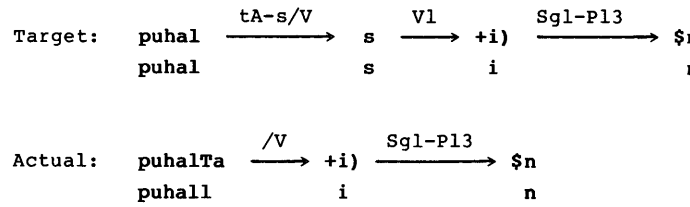
The study of performance errors in the framework of the two-level model will here concentrate on two points:

(1) Using two-level configurations as the basic objects.

(2) The role of the two-level rules.

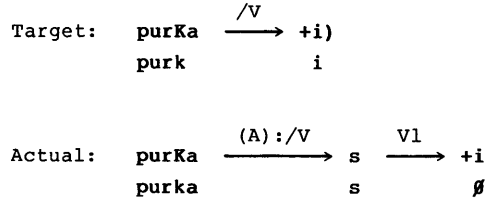
We are thus interpreting the two-level lexicon system only partially, we discuss the (phonological) lexical strings, but not the linkage mechanism. The errors are studied by deducing the configuration corresponding to the intended or "target" form, and comparing it with the configuration assumed to underly the actually produced form.

The vowel stem of a verb and the nominative of a noun are often represented in two parts: the root as the lexical representation of a word entry, and the rest as the lexical representation of the first alternative in the corresponding pattern. Most of the attested child language errors analyzed by Kauppinen (1977) and the aphasic errors studied by Kukkonen (1982) can be interpreted as mistakes in deducing the lexical representation. The overriding tendency is towards stem formation patterns of more productive inflectional classes. An example (from Kauppinen) is the production of **puhallin** instead of **puhalsin** 'I blew'. The configurations corresponding to the intended and actually used forms are:



The vowel stem **puhalta** has been associated with a more productive inflectional type by using the **/V** continuation pattern (without alternations) instead of a more marked **tA-s/V**. Further examples of such errors (from Kauppinen) are **vähennit, uskallin, väännöin** and **käännöin.**

A similar confusion of inflectional types occurs when verbs with no alternation pattern are used as contracted verbs:



The presence of several parallel instances (**sulas, tappasin, mahtas, palas** and **laulas**) shows that the class of contracted verbs is more productive than the nonalternating (cf. F. Karlsson 1983b). (In Kauppinen's data, there were only few instances of the opposite: **vastohan, kuivi.**) Shifts towards productive types are frequent also in forming the first infinitive (**jaata, jauhata** and **tapata**) with few instances of the reverse (**hyppää**). The errors in forming the second participle further corroborate the hypothesis that typical errors can be described as mixing of stem formation patterns, and that the basic trend is from **tA-s/V** and similar less productive patterns to the more productive **/V**, and for stems in **/V** ending in **-a** or **-ä** towards contracted **(A):/V.**

Nominal inflection provides further parallel evidence of errors in deducing the correct base form (see Kauppinen 1977 for data and F. Karlsson 1983b for a summary and analysis of the trends).

As the above diagrams show, the error in deducing the base form can, in many instances, be interpreted as a mistake in forming the lexical string. The lexical stem is usually formed according to the rules of a more productive type. In regard to language acquisition, it

is quite natural to interprete the errors as unsuccesful deductions of base forms.

Certain types of slips of adult speakers could, perhaps, be interpreted as mislinkages. The aphasic studied by Kukkonen (1981) produced errors which could be characterized as selecting the wrong stem alternative, e.g. **kertomuksta** pro **kertomusta** 'story, partitive singular', and **käti** pro **käsi** 'hand, nominative singular'. In the first instance, the inflectional stem has been used instead of the nominative stem, and in the latter vice versa. The normal rules are obeyed: the **E** is realized as zero according to rule 4c (cf. 3.2.4), and the distributional restrictions of the **tA** allomorph are satisfied (cf. rule 12a in section 3.2.9). (The only violation is against the restriction rule 4d which has a marginal status as it describes lexical redundancy.) Similarly in **käti** the inflectional stem **käTE** without endings could have been realized according to normal rules (**E** as **i** at the end of a word).

What is more striking than the presence of the above errors is the extreme rarity of violations against two-level rules. Rules for vowel harmony and vowel doubling seem not to be violated at all. It is hard even to imagine anyone producing ***maahun** instead of **maahan**. According to Kauppinen's and Kukkonen's data, the allomorph selection rules are strictly obeyed. (The only violation is **kahitten** pro **kaksien** which deals with the idiosyncratic word **kaksi** 'two', mentioned in 3.1.8. The child actually experienced trouble with the irregular plural stem of this word. The above instance was accompanied by other errors: **kahille** pro **kaksille**, and **kahtia** pro **kaksia**.)

The vowel alternations at the end of the stem show just one type of error. The child seems to know that certain vowels at the end of the stem are subject to morphophonemic alternations. He knows e.g. that an **a** at the end of the base form is subject either to an **a-o** or to an **a-Ø** alternation. This corresponds in the two-level model to the choice between **Å** and **a**, respectively, in the lexical representation. Confusing these is natural because their distribution is rather complicated and belongs to the redundancy of the lexicon. But the child has acquired all rules for dealing with these morphophonemic alternations, and thus he always produces either zero or **o** before plural **i**, never a surface **a**. Errors in Kauppinen's data may be interpreted so that the child had deduced the base form to be e.g. **porukka** instead of **porukkÅ** 'team', which leads to incorrect plurals like **porukkia**. The same applies to nouns ending in **i**, where the **i** either corresponds to a surface **e** or to zero (as if it were **E**), but never to an **i**. (It should

be stressed here that the morphophonemes in the two-level model need not be interpreted to have any specific phonetic substance. They can also just represent alternations and thus, in some respects, be similar to alternation patterns.)

These observations lead to a simple and explicit model for describing and predicting morphological performance errors in terms of the two-level framework:

> The performance errors in producing word forms are mostly faulty choices in the construction of the lexical level - there are hardly any errors in the application of the two-level rules.

This characterization corresponds to the notion "automatic alternation" of Bloomfield (1933) and Hockett (1958). Thus if two-level rules are used for all automatic alternations and only for them, we arrive at a model where the status of being a "rule" correlates with this observable feature of the speakers of the language. If one strives for realistic descriptions, this criterion can be used for determining which alternations should be described with two-level rules, and which with alternation patterns.

Clear evidence in favour of this two-level model and against SPE-type generative phonology comes from the absence of slips containing intermediate stages of abstract derivations (Kukkonen 1982, Kauppinen 1977). Abstract phonology uses rules even for those alternations where the two-level model has a pattern with unrelated alternatives. E.g. Wiik (1967) describes Finnish morphophonology starting from abstract underlying forms. To produce a forms like **kuollut** 'dead' and **kuolleen** 'of a dead' he posits an underlying representations **kool+neth** and **kool+neth+n**, respectively. There is no evidence of slips with intermediate stages of such derivations (with **-oo-** or anything resembling **-neth**). The two-level description recognizes the fact that **-ut** and **-ee-** are no more actively related. The alternation is therefore described as an alternation pattern Ut/A (cf. 3.1.8).

The absence of intermediate forms can also be accounted for by so called concrete morphology (e.g. F. Karlsson 1983b). The two-level model differs in its predictions from concrete morphology (at least) in the following respect. In concrete morphology it is difficult to explain why there are hardly any errors of the type **kaupaissa** 'in the shops' or **lasiissa** 'in the glasses' where the stem final vowel on the surface would be the same as in the base form. Such forms should be

likely to occur because they contain the unchanged base form stem and they do not violate any phonotactic principles of Finnish.

The two-level model is an approach based on the same objectives as concrete phonology but it is more concentrated on the feasibility of the dynamic processing. Current theories of concrete phonology have not elaborated detailed performance models. The use of alternation patterns provides a starting point for solving the problems of whether the lexicon contains one single representation for each word or whether there should be a separate representation for each inflected form (or perhaps one for each distinct stem). The first alternative leads to a complicated recognition process and the latter to difficulties in learning new words. The concept of alternation patterns combines the benefits of both of these approaches.

5.5 Computational complexity

In the study of syntax there has been a recent trend towards models that are simpler to process than the transformational one. Gazdar (1979) has worked on context-free grammars, Church (1980), and Ejerhed and Church (1983) have gone even further to finite-state techniques. These trends are quite natural if the models are to include processual phenomena of production and recognition.

In computer science there is a branch analyzing the computational complexity of algorithms. Algorithms are evaluated according to their space and time requirements, i.e. how much memory they need for storing intermediate results and how much time they need for accomplishing the computation. The time requirements are expressed as functions of the size of the input, and they are upper limits which cover even the worst cases. Processing in linear time means that the computation is performed in a time that is proportional to the length of the input. Processing in real time means that there is a constant upper limit for the processing effort between each input symbol. The requirement that processing take place in real time is stricter than the other that it take place in linear time (which may postpone some processing of the earlier symbols until all symbols have been input). Many algorithms, however, consume time which sharply increases with the length of the input. Some need time proportional to the exponential function 2^n, some to a constant power of of the input length, e.g. n^3.

Some care is needed in interpreting such time requirements (Berwick & Weinberg 1982). Models of competence (i.e. specifica-

tions) cannot be falsified with arguments of computational complexity of a specific algorithm, because algorithms are not even a part of such models. Proving the nonexistence of an efficient alternative algorithm is extremely difficult (if not impossible).

But everything is different if we are proposing models of the processes of language production and recognition. Then the models contain algorithms, and therefore exponential or cubic time requirements are fatal, if they turn out to be the actual requirement in tasks which humans can process and solve in real time. We must reject such algorithms as unrealistic. However, it would not rule out the possibility that there do perhaps exist better algorithms for solving the same task. Thus, other parts of the model might be valid even if an algorithm is unfeasible.

These anticipated time requirements refer to sequential execution, and they might be quite different if special purpose electronic circuits or other parallel devices are used instead. Therefore, we might accept an algorithm in our computer model although it is poor in computational behaviour, if we know that there exists an alternative implementation or a device that behaves acceptably. We must also consider the nature of the individual steps in an algorithm, because these affect the complexity. Some operations are elementary for the computer, others are for neural networks. Multiplication and division of large numbers are carried out by single computer instructions in a microsecond, but they may be extremely difficult for the human mind. In associative tasks the ratio may be the reverse.

5.6 Processability

There are hardly any detailed models of the actual human morphological analysis and production processes. The commitment to the processual aspects has been restricted to relatively high levels of the processing. Linell (1976) argues for "morphological operations" claiming them to be indivisible in the processual sense, but he gives no model for the execution of such operations. The essence of his arguments concern the representations and the relations between them, rather than the operations in the dynamic sense. In syntax there has recently been more interest in the processual aspect. We parse sentences in real time. It is therefore natural to search for processual models capable of the same (Church 1980, Ejerhed & Church 1983).

The ultimate linguistic reality would be based on neurophysio-

logical facts describing the actual processes of language use. The present knowledge of the neural functions in the human brain is, however, quite limited, but we have indeed some access to the actual phenomena, especially to the basic mechanisms at the elementary level of neurons (see e.g. Kohonen 1982 on a model of neural self-organization). Unfortunately, we have presently no uncontroversial means to reveal how higher structures are processed in the human brain.

Therefore an argumentation based on neurophysiological reality cannot be decisive at this stage. But we can use indirect evidence from elementary neural operations and try to evaluate what kinds of structures and processes would be easy or difficult for networks of neurons to process (see e.g. Minsky 1981). Present knowledge indicates that brains have a vast memory capacity but that sequential processing is more difficult (Singh 1966, Kohonen 1980). Such a profile would clearly be quite the opposite of current computers' capabilities. It is, indeed, incompatible e.g. with having rewriting systems as models of dynamic neural processes.

Psychological reality was attributed to the specification of a modular program and to the overall strategies in the implementation of the few central procedures. If we move to finer details in such a program, the sequential execution of procedures will start to have intermediate results which could by no means have any correspondence to linguistic reality. Low-level constructs like program loops and numerical variables exist in programs mostly because we have to use the only tools that are available in programming languages. Additions, subtractions, tests for equal values, branch instructions and the like are clearly just consequences of the machine architecture of computers and the design of programming languages, they do not represent real cerebral processes.

Computational complexity as discussed above applies at the higher levels. The complexity must be approached differently in those lower level procedures the internal structures of which are not taken to represent anything real. Such lower level modules and functions thus have to be evaluated as whole units. Ultimately, we should answer the question whether there exists an equivalent neural capability in the brain. Because we cannot answer this, we try to answer the question whether the operation is simple enough to be processed by neural networks (as far as we can tell). A positive answer could be reached if there exist alternative devices that perform the same tasks in real time, and that neural networks could perform similar tasks.

We would not have severe problems if we could decide what tasks

neural networks could and what they could not perform. We cannot do this but, fortunately, there are some classes of mechanisms that are inherently simple and that can easily be accomplished on several kinds of devices. Finite state automata have such good qualities (provided that their size remains reasonable). On the one hand, they execute in real time, i.e. the length of the input does not affect the speed (and the computation between input symbols is bounded). On the other hand, finite automata are equivalent to many other models, e.g. so called regular grammars, which are the simplest of phrase structure grammars. Therefore, we could propose as a special case:

Reducibility to finite state automata

> A model of the recognition and/or production processes should be simple enough to be simulated with finite state automata.

A model satisfying this criterion has the advantage that we can speculate that neural networks could perform the same task as the model. The neurons would certainly not simulate the automata, but they could, in principle, perform an equivalent function. Models that are not reducible to finite state mechanism are, of course, not overruled, but the reducible ones do have an advantage over them. The two-level model rules are implemented as finite state automata, and thus it would be reasonable to claim that they could correspond to simple neural networks processing the rules in real time, just like a hard-wired circuitry transforms a signal implicitly without complex operations.

5.7 Evaluation of the implementation

The full implementation of the two-level model simulates the processes of morphological recognition and production. It is linguistically interesting only in the sense that it can be used for evaluating the complexity of the components of the model. In contrast to the minor linguistic relevance, the implementation has an obvious practical relevance.

In natural language processing systems, the two-level program (as such or rewritten in another programming language) could serve as a morphological module. Being adequate for a wide range of languages it would improve the language independence of such systems.

The size and complexity of the program are reasonable. The program consists of some two thousand lines of PASCAL code, most of them dealing with common concepts (strings, sets, trees, finite state automata and queues). If the program is small enough to run on a microcomputer, it certainly can be a part of natural language systems which conventionally are quite large. The execution speed is reasonable: on a Burroughs B7800 computer some 0.1 seconds of processor time is needed for analyzing an inlected word form. The PASCAL program now runs on DEC-20 as well, and it has been reprogrammed in INTERLISP by Karttunen and his students (Karttunen 1983, Gajek & al. 1983). They have also written two-level descriptions for further languages: Japanese (Alam 1983), Rumanian (Khan 1983), French (Lun 1983), and English (Karttunen & Wittenburg 1983).

Information retrieval systems based on texts in inflected languages could benefit from the two-level program. The searches could be made accurate by letting through only hits that are true inflected forms of the search key. Technical advantages could be gained by reducing the number of distinct word types by replacing inflected word forms by their base form (i.e. by lemmatizing the word forms). At the same time compound words (written without a separator between the parts) could be retrieved according to their components.

REFERENCES

Ager, A. E., F. E. Knowles, and Joan Smith (eds.), 1979. Advances in computer-aided literary and linguistic research. (Proceedings of the Fifth International Symposium on Computers in Literary and Linguistic Research, 3-7 April 1978.) Birmingham.

Aho, Alfred V, and Margaret J. Corasick, 1975. Efficient string matching: An aid to bibliographic search. Communications of the ACM, Vol. 18, No. 6, pp. 333-343.

Aho, Alfred V., and Jeffrey D. Ullman, 1972. The theory of parsing, translation and compiling. Volume I: Parsing. Englewood Cliffs, N. J. (Prentice-Hall.)

Alam, Yokiko Sasaki, 1983. A two-level morphological analysis of Japanese. Texas Linguistic Forum, No. 22, pp. 229-252.

Allén, Sture, 1970. Frequency dictionary of present-day Swedish based on newspaper material. Volume 1, Graphic words, homograph components. Stockholm.

---, 1971. ---. Volume 2. Lemmas. Stockohlm.

Anttila, Raimo, 1980. Field theory and morphology. Lingua Posnaniensis, XXII, pp. 15-19.

Back, Ralph-Johan and Kimmo Koskenniemi, 1980. Constructing verifiable programs: A design method and a case study. Research Reports No. 11, Computing Centre of the University of Helsinki. Helsinki.

Berwick, Robert C., 1983. Computational complexity and lexical-functional grammar. American Journal of Computetional Linguistics, Vol. 8, No. 3-4, pp. 97-109.

Berwick, Robert C. and Amy S. Weinberg, 1982. Parsing efficiency, computational complexity, and the evaluation of grammatical theories. Linguistic Inquiry, Vol. 13, No. 2, pp. 165-191.

Black, Max, 1962. Models and metaphors: Studies in language and philosophy. Ithaca, N.Y. (Cornell University Press.)

Bloomfield, Leonard, 1933. Language. New York.

Booth, Taylor L., 1967. Sequential machines and automata theory. New York, London, Sydney. (John Wiley and Sons, Inc.)

Brodda, Benny, and Fred Karlsson, 1980. An experiment with Automatic morphological analysis of Finnish. Papers from the Institute of Linguistics, University of Stockholm, Publication 40. (Reprinted 1981: Department of General Linguistics, University of Helsinki, Publications, No. 7.)

Bruck, A., R. Fox, and M. Lagaly (eds.), 1974. Papers from the parasession on natural phonology. Chicago Linguistic Society.

Bybee, Joan L. and Dan I. Slobin, 1982. Rules and schemas in the development and use of the English past tense. Language, Vol. 58, No. 2, pp. 256-289.

Chang, May See, 1980. The Morphological analysis of Bahasa Malaysia. Proceedings of COLING 80, pp. 578-585. Tokyo.

Chapin, Paul G., and Lewis M. Norton, 1968. A Procedure for morphological analysis. Information system language studies number eighteen, MTP-101. Massachusetts. (The Mitre Corporation.)

Chomsky, Noam, and Morris Halle, 1968. The sound pattern of English. New York. (Harper and Row.)

Dietrich, Rainer, 1973. Automatische Textwörterbücher: Studien zur maschinellen Lemmatisierung verbaler Wortformen des Deutschen. Tübingen. (Max Niemeyer Verlag.)

Ejerhed, Eva, and Kenneth Church, 1983. Finite state parsing. F. Karlsson (ed.) 1983c, pp. 410-432.

Eliasson, Stig, 1975. On the issue of directionality. In: Dahlstedt, K.-H. (ed.), The Nordic Languages and Modern Linguistics, 2. Stockholm. (Almqvist & Wiksell.)

Farkas, D., W. Jacobsen, and K. Todrys (eds.), 1978. Papers from the Parasession on the lexicon. Chicago Linguistic society. Chicago, Ill.

Fleck, Margaret M., 1982. Design options for a morphological analysis system. Unpublished senior essay, Linguistics, Yale.

Fodor, J. A., T. G. Bever, and M. F. Garrett, 1974. The psychology of language. (McGraw-Hill.)

Fujimura, O., (ed.), 1973. Three dimensions of linguistics Theory. Tokyo. (TEC Company.)

Gajek, O., H. Beck, D. Elder, and G. Whittemore, 1983. A two-level morphological analyzer: LISP implementation. Texas Linguistic Forum, No. 22, pp. 187-202.

Gazdar, G., 1979. English as a context-free language. Univerisity of Sussex.

---, 1981. Unbounded dependencies and coordinate structure. Linguistic Inquiry, Vol. 12, pp. 155-184.

Hann, M. L., 1974. Principles of automatic lemmatization. ITL: Revue of applied linguistics, 23, pp. 1-22.

---, 1975. Towards an algorithmic methodology of lemmatization. ALLC Bulletin, (Summer 1975), pp. 140-150.

Hellberg, Staffan, 1971. Automatisk lemmatisering: En modell för uppträdande av böjningsserier i ett frekvenslexicon. Språkdata, Göteborg. (Stencil.)

---, 1972. Computerized lemmatization without the use of a dictionary: A case study from Swedish lexicology. Computers and the humanities, Vol. 6, No. 4 (March 1972), pp. 209-212.

---, 1978. The morphology of present-day Swedish. (Data Linguistica, No. 13.) Stockholm. (Almqvist & Wiksell.)

Hockett, Charles F., 1954. Two models of grammatical description. Word, Volume 10, pp. 210-237.

---, 1958. A course in modern linguistics. New York.

Hopcroft, John E., and Jeffrey D. Ullman, 1969. Formal languages and their relation to automata. Reading, Massachusetts. (Addison-Wesley.)

House, Roger, 1980. Comments on program specification and testing. Communications of the ACM, Vol. 23, No. 6, pp. 324-331.

Itkonen, Terho, 1966. Mellakoihin vai mellakkoihin? Tietolipas. (Suomalaisen Kirjallisuuden Seura.)

Jackendoff, Ray, 1975. Morphological and semantic regularities in the lexicon. Language, Vol. 51, No. 3, pp. 639-671.

Jensen, Kathleen and Niklaus Wirth, 1974. PASCAL User manual and report. (Springer Verlag)

Jones, Alan, and R. F. Churchhouse (eds.), 1976. The computer in literary and linguistic studies. Cardiff.

Joshi, S. D., and Paul Kiparsky, 1979. Siddha and asiddha in Pāṇinian phonology. In: Daniel A. Dinnsen (ed.), Current approaches to phonological theory. (Indiana Univ. Press.)

Kain, Richard Y., 1972. Automata theory: Machines and languages. New York. (McGraw-Hill.)

Kaplan, Ronald, 1972. Augmented transition networks as psychological models of sentence comprehension. Artifical Intelligence, Vol. 3, pp. 77-100.

Kaplan, Ronald M., and Martin Kay, 1981. Phonological rules and finite-state transducers. Paper read at the annual meeting of the Linguistic Society of America in New York City.

Karlsson, Fred, 1974. Centrala problem i finskans böjningsmorfologi, morfofonematik och fonologi. Suomi 117:2. Helsinki. (Suomalaisen Kirjallisuuden Seura.)

---, 1976. Finskans struktur. Lund. (Liberläromedel.)

---, 1977. Eräistä morfologian teorian ajankohtaisista ongelmista. Sananjalka, 19, pp. 26-56.

---, 1979. Finsk grammatik. Helsinki. (Suomalaisen Kirjallisuuden Seura.)

---, 1982. Suomen peruskielioppi. Helsinki. (Suomalaisen Kirjallisuuden Seura.)

---, 1983a. Finnish grammar. Porvoo. (WSOY.)
---, 1983b. Suomen yleiskielen äänne ja muotorakenne. Porvoo.
---, (ed.) 1983c. Papers from the seventh scandinavian congress of linguistics. Vol. I & II. Department of general linguistics, University of Helsinki, Pulications, No. 9 & 10.
Karlsson, Göran, 1978. Kolmi- ja useampitavuisten nominivartaloiden loppu-A:n edustuminen monikon i:n edellä. Rakenteita. Publications of the Department of Finnish and General linguistics of the University of Turku, No. 6, pp. 86-99. Turku.
Karttunen, Lauri, Rebecca Root, and Hans Uszkoreit, 1981. TEXFIN: Morphological analysis of Finnish by computer. A paper read at 71st Annual Meeting of the SASS, Albuquerque, New Mexico.
Karttunen, Lauri, 1983. KIMMO: A general morphological processor. Texas Linguistic Forum, Vol. 22, pp. 165-186.
Karttunen, L., and K. Wittenburg 1983. A two-level analysis of English. Texas Linguistic Forum, No. 22, pp. 217-228.
Kauppinen, Anneli, 1977. Mikon kielioppia: Havaintoja 3 vuoden 4 kuukauden ikäisen pojan kielestä ja sen kehityksestä vuoden aikana. Licentiate's thesis. Department of Finnish, University of Helsinki.
Kay, Martin, 1975. Syntactic processing and functional sentence perspective. In: Nash-Webber & Schank 1975, pp. 6-9.
---, 1977a. Morphological and syntactic analysis. In Zampolli, 1977, pp. 131-234.
---, 1982. When meta-rules are not meta-rules. In Spark-Jones & Wilks (eds.) 1982.
Kay, Martin, and Gary R. Martins, 1970. The MIND system: The morphological-analysis program. Memorandum RM-6265/2-PR (April 1970). Santa Monica, California. (The RAND Corporation.)
Kernigham, and Plauger, 1974, The elements of programming style. (McGraw-Hill.)
Khan, Robert, 1983. A two-level morphological analysis of Rumanian. Texas Linguistic Forum, No. 22, pp. 253-270.
Khan, R., J. Liu, T. Ito, and K. Shuldberg, 1983. KIMMO user's manual. Texas Linguistic Forum, No. 22, pp. 203-215.
Kiparsky, Paul, 1973. Phonological representations. In Fujimura, 1973, pp. 1-136.
Klein, Wolfgang, und Rainer Rath, 1971. Automatische Lemmatisierung. Linguistischen Arbeiten Germanistischen Instituts und des Instituts für Angewandte Mathematik der Universität des Saarlandes, Arbeitsheft Nr. 10. Saarbrücken.

Klopprogge, Manfred, and Joachim Tschampel, 1976. Automatische Reduktion von Wortformen in deutschen Texten. Interner Bericht Nr. 11/76. Universität Karlsruhe.

Knuth, Donald, 1973. The art of computer programming. Volume 1: Fundamental algorithms. (Second edition 1979.) (Addison-Wesley.)

Kohonen, Teuvo, 1980. Voivatko tietokoneet kehittyä aivojen kaltaisiksi? TIEDE 2000, No. 2, pp. 35-40.

---, 1982, Self-organized formation of topologically correct feature maps. Biological Cybernetics, Vol. 43, pp. 59-69. (Springer.)

Koskenniemi, Kimmo, 1983. Two-level model for morphological analysis. Proceedings of the Eighth International Joint Conference on Artificial Intelligence, (IJCAI-83), pp. 683-685.

Kukkonen, Pirkko, 1982. Tapaustutkimus motorisesta afasiasta: Nuoren aivoinfarktipotilaan kieliopillisia vaikeuksia. Master's thesis in general liguistics at the University of Helsinki.

Källgren, Gunnel, 1981. FINVX - A system for the backwards application of Finnish consonant gradation. Papers from the Institute of Linguistics, University of Stockholm. Publication 42.

---, 1983. Computerized analysis and synthesis of Finnish nominals. F. Karlsson (ed.), pp. 433-444.

Linell, Per, 1976. On the structure of morphological operations. Linguistische Berichte, 44/76, pp. 1-29.

---, 1979. Psychological reality in phonology: A theoretical study. Cambridge. (Cambridge University Press.)

---, 1980. Linguistics and psycholinguistcs: same or different. ALVAR: Studies presented to Alvar Ellegård on the occasion of his 60th birthday. Jens Allwood, and Magnus Ljung (ed.). SPELL, No 1, pp. 100-115. University of Stockholm, Department of English.

Lovins, Julie Beth, 1969, Development of a stemming algorithm. Mechanical Translation and Computational Linguistics, Vol. X, pp 22-31.

Lun, S., 1983. A two-level morphological analysis of French. Texas Linguistic Forum, No. 22, pp. 271-278.

Maegaard, Bente, og Hanne Ruus, 1977. En orientering om DANwORD. SAML 4.

---, 1978. DANwORD: Frequency studies in modern Danish. COLING 78, Information Abstracts. Bergen.

Marcus, Mitchell, 1980. A theory of syntactic recognition for natural language. (The MIT Press.)

Matthews, P. H., 1965a. Some concepts in word-and-paradigm morphology. Foundations of Language, Volume 1, pp. 268-289.

---, 1965b. The inflectional component of a word-and-paradigm grammar. Journal of Linguistics, Volume 1, pp. 139-171.

Meunier, Jean, Jean Boisvert and Francois Denis, 1976. The lemmatization of contemporary French. In Jones et al. 1976, pp.208-214.

Minsky, Marvin, 1981. K-lines: A theory of memory. Norman (ed.) 1981, pp. 87-104.

Nash-Webber, Bonnie, and Roger Schank, (eds), 1975. Theoretical issues in natural language processing (TINLAP-1). Cambridge, Mass.

Norman, Donald E. (ed.), 1981. Perspectives on cognitive science. New Jersey. (Ablex/Lawrence Erlbaum.)

Pacak, M., 1966. Computational morphology. T. A. Informations, Volume 1, pp. 25-36.

Paunonen, Heikki, 1976. Allomorfien dynamiikkaa. Virittäjä, 1976, pp. 82-107.

Penttilä, Aarni, 1963. Suomen kielioppi. 2nd edition. (WSOY.)

Price, James D., 1969. An algorithm for analyzing Hebrew words. Computer Studies in the Humanities and Verbal Behavior, Volume 2, pp. 137-165.

Rath, Rainer, 1971. Probleme der automatischen Lemmatisierung (AL). Zeitschrift fur Phonetik, Sprachwissenschaft und Kommunikationsforschung, Band 24, Heft 5, pp. 409-425. Berlin. (Akademie-Verlag.)

Raun, Alo, 1959. Suomen kielen deklinaatioista ja konjugaatioista. Virittäjä, 4, pp. 348-351.

Robins, R. H., 1959. In defence of WP. Transactions of the Philological Society, 1959, pp. 116-144. (Reprinted in R. H. Robins, Diversions of Bloomsbury, North-Holland 1970.)

Ruus, Hanne, 1977. Ordmekanik. SAML III.

Salomaa, Arto, 1969. Theory of automata. Oxford. (Pergamon Press.)

---, 1973. Formal languages. New York. (Academic Press.)

Schott, Gerda, 1972. Automatic analysis of inflectional morphemes in German nouns: An algorithm for automatic indexing. Acta Informatica 1, pp. 360-375.

Setälä, E. N., 1963. Suomen kielioppi: Äänne- ja sanaoppi. (16th edition.) Helsinki. (Otava.)

Schlesinger, I. M., 1977. Production and comprehension of utterances. New York.

Singh, Jagit, 1966. Great ideas in information theory, language and cybernetics. New York. (Dover Publications, Inc.)

Skousen, Royal, 1975. Substantive evidence in phonology: The evidence from Finnish and French. Janua Linguarum, Series Minor, 217. The Hague. (Mouton.)

Spark-Jones, K., and Y. Wilks, 1982. Automatic natural language pars-

ing. University of Essex, Cognitive Studies Centre. (CSCM-10.)
Sågvall, Anna-Lena, 1973. A system for automatic inflectional analysis (implemented for Russian). Stockholm. (Almqvist & Wiksell.)
Sågvall Hein, Anna-Lena, 1978. Finnish morphological analysis in the reversible grammar system. COLING 78, Information Abstracts. Bergen.
---, 1980. An outline of a computer model of Finnish word recognition. Fenno-Ugrica Suecana, 3, pp. 7-26.
Trubetzkoy, N. S., 1939. Grundzuge der Phonologie. Prague: Cercle Linguistique de Prague. (4. Auflage, 1958. Göttingen)
Tuomi, Tuomo, 1972. Reverse dictionary of modern standard Finnish. (Suomalaisen Kirjallisuuden Seura.)
Turner, Joshua, 1980. The structure of modular programs. Communications of the ACM. Vol. 23, No. 5, pp. 272-277.
Vennemann, Theo, 1974. Words and syllables in natural generative grammar. In Bruck & Fox & Lagaly (eds.) 1974, pp. 346-374.
Weber, Heinz Josef, 1976. Automatische Lemmatisierung - Zielsetzung und Arbeitsweise eines linguistischen Identifikationsverfahrens. Linguistische Berichte 44/ 76, pp. 30-47.
Wiik, Kalevi, 1967. Suomen kielen morfofonemiikkaa. Publications of the Phonetics Department of the University of Turku, No. 3.
---, 1975. Vokaalisoinnun ongelmia. Publications of the Phonetics Department of the University of Turku, No. 14.
Winograd, Terry, 1983. Language as a cognitive process. Volume 1: Syntax. (Addison-Wesley.)
Wirth, Niklaus, 1971. The programming language Pascal. Acta Informatica, 1, pp. 35-63.
---, 1979. Modula-2. Institut fur Informatik, ETH, Zurich.
Wolfart, H. Christoph, 1979. An experiment in automatic inflectional analysis: Latin. Ager et al. (eds.), 1979, pp. 73-94.
Wolfart, H. C., and F. Pardo, 1973. Computer-assisted linguistic analysis. University of Manitoba Antropology Papers, 6. Winnipeg.
Yli-Vakkuri, Valma, 1976. Onko suomen kvalitatiivinen astevaihtelu epäproduktiivinen jäänne? Sananjalka, 18, pp. 53-69.
Zampolli, Antonio (ed.), 1977. Linguistic structures processing. Fundamental Studies in Computer Science, Volume 5. Amsterdam. (North-Holland.)

APPENDIX A
THE AUTOMATA FOR FINNISH

```
ALPHABET
     a b c d e f g h i j k l m n o p q r s t u v w x y z ä ö '
     A O U E K P T D Z : Å Ä
     . ( ) + _ / §   $ * ´
     1 2 3 4 5̄ 6 7 8 9 % & !
     NULL 0
     ANY  =
     SUBSET V  a e i o u y ä ö A O U E Å Ä
     SUBSET C  b c d f g h j k l m n p q r s t v w x z K P T
     SUBSET B  a o u Å
     SUBSET F  ä ö y Ä
     SUBSET L  l r
     SUBSET X  . $ ´ + ( ) * / _ § 1 2 3 4 5 6 7 8 9 % & ! Z D
     END

"(1) Surface Characters: to introduce default correspondencies"  1  29
      a b c d e f g h i j k l m n o p q r s t u v w x y z ä ö =
      a b c d e f g h i j k l m n o p q r s t u v w x y z ä ö =
 1:   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

"(2) Vowel Duplication: Rule 1a,e"          12   21

      = : = : = : = : = : = :  = : = : = :  / :  X =
      a a e e i i o o u u y y  ä ä ö ö h  0 0  0 =

  1:  2 0 3 0 4 0 5 0 6 0 7 0  8 0 9 0 1  1 1  1 1
  2: 10 2 3 0 4 0 5 0 6 0 7 0  8 0 9 0 2  2 2  2 1
  3:  2 0 3 3 4 0 5 0 6 0 7 0  8 0 9 0 3  3 3  3 1
  4:  2 0 3 0 4 4 5 0 6 0 7 0  8 0 9 0 4  4 4  4 1
  5:  2 0 3 0 4 0 5 5 6 0 7 0  8 0 9 0 5  5 5  5 1
  6:  2 0 3 0 4 0 5 0 6 6 7 0  8 0 9 0 6  6 6  6 1
  7:  2 0 3 0 4 0 5 0 6 0 7 7  8 0 9 0 7  7 7  7 1
  8:  2 0 3 0 4 0 5 0 6 0 7 0 11 8 9 0 8  8 8  8 1
  9:  2 0 3 0 4 0 5 0 6 0 7 0  8 0 9 9 9  9 9  9 1
 10: 10 2 3 0 4 0 5 0 6 0 7 0  8 0 9 0 2 12 0 10 1
 11:  2 0 3 0 4 0 5 0 6 0 7 0 11 8 9 0 8 12 0 11 1
 12:  0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0  0 0  0 1

"(3) Ignored Lengthenings: Rules 1b, 1c, 1d"   7 11

      V  :  :  )  +  (  i  X  .  /  =
      V  V  0  0  0  0  =  0  0  0  =

  1:  2  1  0  1  1  1  2  1  1  1  1
  2:  3  3  4  7  2  2  2  2  1  2  1
  3:  2  2  1  7  6  6  2  3  1  2  1
  4.  0  0  0  4  5  5  0  4  0  0  0
  5.  0  0  0  5  0  0  3  5  0  0  0
  6:  2  0  1  6  6  6  0  6  1  1  1
  7:  2  0  1  7  7  7  2  7  1  1  1
```

"(4) Stem final a, ä, Å, Ä: Rule 2a" 5 17

	a	A	Å	ä	A	Ä	a	A	Å	ä	Ä	+	i	*	´	X	=
	a	a	a	ä	ä	ä	0	0	o	0	ö	0	=	0	0	0	=
1:	4	4	4	4	4	4	2	2	2	2	2	1	1	1	1	1	1
2.	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	2	0
3.	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	3	0
4:	4	4	4	4	4	4	2	2	2	2	2	5	1	0	1	4	1
5:	4	4	4	4	4	4	0	0	0	0	0	0	0	5	5	5	1

"(5) Stem final a, ä <--> e: Rules 2b, 2c" 7 18

	C	.	=	:	a	ä	A	Å	Ä	a	ä	A	Å	Ä	´	*	X	=
	=	0	V	=	e	e	e	e	e	=	=	=	=	=	0	0	0	=
1:	1	1	2	2	0	0	0	0	0	2	2	2	2	2	1	1	1	1
2:	3	3	2	2	0	0	0	0	0	2	2	2	2	2	4	4	2	2
3:	3	3	4	4	5	5	5	5	5	6	6	6	6	6	0	4	3	4
4:	4	4	4	4	7	7	7	7	7	4	4	4	4	4	4	4	4	4
5.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	5	0
6:	4	4	4	4	0	0	0	0	0	4	4	4	4	4	0	0	6	4
7.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	7	0

"(6) Stem final i, and E: Rules 3, 4a,b,c,e" 8 18

	i	i	E	E	E	1	2	+	(	*	´	_	§	/	i	e	X	=
	i	e	i	e	0	0	0	0	0	0	0	0	0	0	=	e	0	=
1:	5	2	8	6	3	1	1	1	1	1	1	1	1	1	1	1	1	1
2.	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	2	0
3.	0	0	0	0	0	1	1	4	4	4	0	0	0	0	0	0	3	0
4.	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	4	0
5:	5	2	8	6	3	1	1	7	1	1	1	1	1	1	1	1	5	1
6.	0	0	0	0	0	0	0	7	7	7	7	0	0	0	0	0	6	0
7:	0	0	0	0	0	0	0	7	7	7	7	1	1	1	0	1	7	1
8:	0	0	0	0	0	0	0	8	0	0	0	1	1	0	0	1	8	0

"(7) Consonant stem from E: Rules 4d" 6 13

	=	=	=	=	=	=	=	E	1	2	X	=	=
	V	l	r	n	t	h	s	0	0	0	0	C	=
1:	2	1	1	1	1	1	1	6	1	1	1	1	1
2:	2	3	3	3	4	4	4	2	2	2	2	1	1
3:	2	1	1	1	4	1	1	5	1	1	3	1	1
4:	2	1	1	1	1	1	1	5	1	1	4	1	1
5:	2	1	1	1	1	1	1	0	1	1	5	1	1
6:	2	1	1	1	1	1	1	0	0	0	6	1	1

"(8) Vowel Harmony: Rules 5a, 5b" 2 10

	=	=	§	A	O	U	A	O	U	=
	F	B	0	ä	ö	y	a	o	u	=
1:	1	2	1	1	1	1	0	0	0	1
2:	1	2	1	0	0	0	2	2	2	2

"(9) Plural i Between Vowels: Rule 6" 5 6

	=	+	i	i	X	=
	V	0	=	j	0	=
1:	2	1	2	0	1	1
2:	2	3	2	0	2	1
3:	2	3	5	4	3	1
4.	2	4	2	0	4	0
5:	0	5	0	0	5	1

"(10) Deletion of n in front of poss. suff.: Rule 7" 4 5

	+	n	n	/	=
	0	n	0	0	=
1:	2	1	0	0	1
2:	2	4	3	1	2
3.	0	0	0	1	0
4:	2	4	0	0	2

"(11) Weakening of p: Rules 9a,b,c" 5 10

	=	=	=	=	P	P	P	P	X	=
	V	L	m	p	p	v	m	0	0	=
1:	2	1	1	1	0	0	0	0	1	1
2:	2	3	4	5	1	1	0	0	2	1
3:	2	1	1	5	1	1	0	0	3	1
4:	2	1	1	5	1	0	1	0	4	1
5:	2	1	1	1	1	0	0	1	5	1

"(12) Weakening of T: Rules 10a,b,c" 8 15

	=	s	=	=	=	=	=	T	T	T	T	T	T	=	=
	V	s	h	l	r	n	t	t	d	l	r	n	0	0	=
1:	2	8	1	1	1	1	1	0	0	0	0	0	0	1	1
2:	2	8	3	4	5	6	7	1	1	0	0	0	0	2	1
3:	2	8	1	1	1	1	7	1	1	0	0	0	0	3	1
4:	2	8	1	1	1	1	7	1	0	1	0	0	0	4	1
5:	2	8	1	1	1	1	7	1	0	0	1	0	0	5	1
6:	2	8	1	1	1	1	7	1	0	0	0	1	0	6	1
7:	2	8	1	1	1	1	1	1	0	0	0	0	1	7	1
8:	2	8	1	1	1	1	1	1	0	0	0	0	0	8	1

"(13) K <-> 0 or v after CU: Rule 8a" 7 7

	=	=	=	K	K	X	=
	C	u	y	0	v	0	=
1:	2	1	1	1	0	1	1
2:	2	3	4	1	0	2	1
3:	2	1	1	7	5	3	1
4:	2	1	1	7	6	4	1
5.	0	1	0	0	0	5	0
6.	0	0	1	0	0	6	0
7.	0	0	0	0	0	7	1

"(14) K <-> ' between identical vowels: Rule 8b" 26 13

	= C	= a	= e	= i	= o	= u	= y	= ä	= ö	K 0	K '	= 0	= =
1:	1	2	2	2	2	2	2	2	2	1	0	1	1
2:	1	3	4	5	6	7	8	9	10	1	0	2	1
3:	1	3	4	5	6	7	8	9	10	11	19	3	1
4:	1	3	4	5	6	7	8	9	10	12	20	4	1
5:	1	3	4	5	6	7	8	9	10	13	21	5	1
6:	1	3	4	5	6	7	8	9	10	14	22	6	1
7:	1	3	4	5	6	7	8	9	10	15	23	7	1
8:	1	3	4	5	6	7	8	9	10	16	24	8	1
9:	1	3	4	5	6	7	8	9	10	17	25	9	1
10:	1	3	4	5	6	7	8	9	10	18	26	10	1
11.	0	0	1	1	1	1	1	1	1	0	0	11	0
12.	0	1	0	1	1	1	1	1	1	0	0	12	0
13.	0	1	1	0	1	1	1	1	1	0	0	13	0
14.	0	1	1	1	0	1	1	1	1	0	0	14	0
15.	0	1	1	1	1	0	1	1	1	0	0	15	0
16.	0	1	1	1	1	1	0	1	1	0	0	16	0
17.	0	1	1	1	1	1	1	0	1	0	0	17	0
18.	0	1	1	1	1	1	1	1	0	0	0	18	0
19.	0	1	0	0	0	0	0	0	0	0	0	19	0
20.	0	0	1	0	0	0	0	0	0	0	0	20	0
21.	0	0	0	1	0	0	0	0	0	0	0	21	0
22.	0	0	0	0	1	0	0	0	0	0	0	22	0
23.	0	0	0	0	0	1	0	0	0	0	0	23	0
24.	0	0	0	0	0	0	1	0	0	0	0	24	0
25.	0	0	0	0	0	0	0	1	0	0	0	25	0
26.	0	0	0	0	0	0	0	0	1	0	0	26	0

"(15) K after a consonant: Rules 8a,d,e" 12 16

	= V	= L	= n	= h	= k	K k	K j	K 0	K g	e e	E e	E 0	i i	$ 0	= 0	= =
1:	2	1	1	1	1	0	0	0	0	2	2	2	2	1	1	1
2:	2	3	4	5	6	1	0	1	0	2	2	2	2	2	2	1
3:	2	1	1	1	6	1	10	7	0	2	2	2	2	3	3	1
4:	2	1	1	1	6	1	0	0	1	2	2	2	2	4	4	1
5:	2	1	1	1	1	1	10	7	0	2	2	2	2	5	5	1
6:	2	1	1	1	1	1	0	1	0	2	2	2	2	6	6	1
7.	2	0	0	0	0	0	0	0	0	0	0	0	8	1	1	1
8.	0	0	0	0	0	0	0	0	0	0	0	0	0	9	1	1
9.	1	1	0	1	1	0	0	0	0	1	1	1	1	1	1	1
10.	0	0	0	0	0	0	0	0	0	1	1	1	11	0	0	0
11.	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0
12.	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

"(16) Selection of strong/weak grade: Rule 11a,b,c" 12 15

	s	K	P	T	K	P	T	V	:	+	i	)	$	X	=
	s	k	p	t	=	=	=	=	0	0	=	0	0	0	=
1:	12	2	2	2	7	7	7	1	1	1	1	1	1	1	1
2:	1	0	0	0	0	0	0	3	3	4	3	1	0	2	1
3:	1	0	0	0	0	0	0	1	1	4	1	1	0	3	1
4:	1	0	0	0	0	0	0	1	1	0	5	1	0	4	1
5:	1	0	0	0	0	0	0	1	1	0	1	6	0	5	1
6:	1	0	0	0	0	0	0	1	1	0	1	0	0	6	1
7.	0	0	0	0	0	0	0	8	8	9	8	7	1	7	0
8.	0	0	0	0	0	0	0	0	0	9	0	8	1	8	0
9.	0	0	0	0	0	0	0	0	0	9	10	9	1	9	0
10.	0	0	0	0	0	0	0	0	0	10	0	11	1	10	0
11.	0	0	0	0	0	0	0	0	0	11	0	11	1	11	0
12:	12	1	1	1	0	0	0	1	1	12	1	12	12	12	1

"(17) Selection of alternative endings: Rules 12b,...,k" 13 19

	C	=	.	V	:	:	V	1	2	3	4	5	6	7	8	9	§	X	=
	=	C	0	V	V	0	0	0	0	0	0	0	0	0	0	0	0	0	=
1:	1	1	0	2	0	0	2	0	0	0	0	0	0	0	0	0	1	1	1
2:	5	5	5	3	3	4	4	2	0	0	2	0	0	0	2	0	1	2	2
3:	5	5	5	3	3	4	4	3	0	0	3	0	0	0	0	0	1	3	3
4:	5	5	5	3	3	4	4	0	0	0	0	0	4	0	4	0	1	4	4
5:	5	5	0	6	6	0	5	5	5	0	0	0	0	5	0	0	1	5	5
6:	10	10	11	7	8	9	9	0	0	6	0	0	0	6	6	0	1	6	6
7:	10	10	11	11	8	9	9	7	0	0	7	0	0	0	0	0	1	7	7
8:	10	10	11	11	8	9	9	8	0	0	0	8	0	0	0	0	1	8	8
9:	10	10	11	11	8	9	9	0	0	0	0	0	9	0	9	9	1	9	9
10:	10	10	10	13	13	10	10	10	10	0	0	0	0	10	0	0	1	10	10
11:	10	10	11	12	13	11	9	0	0	0	0	0	0	0	0	0	1	11	11
12:	10	10	11	7	8	9	9	12	0	12	0	0	12	0	12	0	1	12	12
13:	10	10	11	7	8	9	9	0	0	13	0	0	13	13	13	0	1	13	13

"(18) Alternative ending !+A: Rule 12a" 3 6

	=	:	!	.	X	=
	V	V	0	0	0	=
1:	2	2	1	1	1	1
2:	3	3	2	1	2	1
3:	3	3	0	1	3	1

"(19) Alternative endings en,ien: Rule 12l,m,n" 4 10

	=	i	=	=	=	=	%	&	=	=
	C	i	o	ö	u	y	0	0	0	=
1:	2	1	1	1	1	1	0	0	1	1
2:	2	3	4	4	4	4	0	1	2	1
3:	2	1	1	1	1	1	1	0	3	1
4:	2	1	1	1	1	1	0	1	4	1

"(20) Exclusion of 2+ten after t: Rule 12d" 2 4

	=	2	=	=
	t	0	0	=
1:	2	1	1	1
2:	2	0	2	1

"(21) D and Z in Infinitives & passives: Rules 13a,...,f" 10 21

	C =	. 0	h h	= V	: V	= t	= r	= l	= n	= s	D 0	D d	D r	D l	D n	D t	Z 0	Z t	* 0	X 0	= =
1:	1	1	1	2	2	4	5	6	7	8	0	0	0	0	0	0	1	0	1	1	1
2:	1	1	10	3	3	4	5	6	7	8	2	0	0	0	0	0	0	4	9	2	1
3:	1	1	1	3	3	4	5	6	7	8	0	1	0	0	0	0	3	0	3	3	1
4:	1	1	1	2	2	4	5	6	7	8	4	0	0	0	0	0	4	0	4	4	1
5:	1	1	1	2	2	4	5	6	7	8	0	0	5	0	0	0	5	0	5	5	1
6:	1	1	1	2	2	4	5	6	7	8	0	0	0	6	0	0	6	0	6	6	1
7:	1	1	1	2	2	4	5	6	7	8	0	0	0	0	7	0	7	0	7	7	1
8:	1	1	1	2	2	4	5	6	7	8	0	0	0	0	0	4	8	0	8	8	1
9:	1	1	1	3	3	4	5	6	7	8	0	0	0	0	0	1	0	4	9	9	1
10:	1	1	1	2	2	4	5	6	7	8	0	1	0	0	0	0	10	0	10	10	1

"(22) Two assimilations: Rules 14a,b" 12 13

	= V	= r	= l	= s	t n	= t	(0	n n	n r	n l	n s	= 0	= =
1:	2	1	1	1	0	1	1	1	0	0	0	1	1
2:	2	3	4	5	9	11	1	1	0	0	0	2	1
3:	2	1	1	1	0	1	6	1	0	0	0	3	1
4:	2	1	1	1	0	1	7	1	0	0	0	4	1
5:	2	1	1	1	0	1	8	1	0	0	0	5	1
6:	2	1	1	1	0	1	0	0	1	0	0	6	1
7:	2	1	1	1	0	1	0	0	0	1	0	7	1
8.	2	1	1	1	0	1	0	0	0	0	1	8	1
9.	0	0	0	0	0	0	10	0	0	0	0	9	0
10:	0	0	0	0	0	0	0	1	0	0	0	10	0
11:	2	1	1	1	0	1	12	1	0	0	0	11	1
12:	2	1	1	1	0	1	0	0	0	0	0	12	1

END

APPENDIX B
A TEST LEXICON FOR FINNISH

```
CONTINUATIONS
     ( /S = S0 S1 S2 S3 )
     ( /S* = S0 S1 S2 S5 )
     ( S0 = S0 ) ( S01 = S0 S1 )
     ( S03 = S0 S3 ) ( S04 = S0 S4 ) ( S1 = S1 )
     ( S12 = S1 S2 ) ( S123 = S1 S2 S3 ) ( S13 = S1 S3 )
     ( S2 = S2 )
     ( S3 = S3 ) ( S4 = S4 )

     ( /A = C1 C2 S0 S1 S2 S3 )
     ( /A* = C1 C2 S0 S1 S2 S5 )
     ( A1 = C1 S1 )
     ( A12 = C1 C2 S1 S2 )
     ( A123 = C1 C2 S1 S2 S3 )
     ( A13 = C1 S1 S3 )
     ( A12C1 = S1 S2 C1 )
     ( A2 = C2 S2 )
     ( C2 = C2 )

     ( P = P K0 § ) ( P1 = P )
     ( K = K0 § ) ( K01 = K0 K1 § )
     ( R = Root )

     ( /V = V0 V1 V2 V3 )
     ( V0 = V0 ) ( V012 = V0 V1 V2 ) ( V02 = V0 V2 )
     ( V023 = V0 V2 V3 )
     ( V03 = V0 V3 )
     ( V1 = V1 ) ( V12 = V1 V2 ) ( V123 = V1 V2 V3 )
     ( V2 = V2 ) ( V23 = V2 V3 )
     ( V3 = V3 )

     ( 1-3-N = Sg1-Pl3 Neg )
     ( 1-3 = Sg1-Pl3 )
     ( 4-N = Pe4 Neg )

     ( 0-i/S = 0-i/S ) ( 0-i/A = 0-i/A )
     ( mi/S = mi/S ) ( si/S = si/S ) ( si/A = si/A )
     ( psi/S = psi/S ) ( tsi/S = tsi/S ) ( ksi = ksi )
     ( 0-e/S = 0-e/S )
     ( r/S = r/S )
     ( n-me/S = n-me/S ) ( n-me/A = n-me/A )
     ( n-mA/S = n-mA/S ) ( n-mA/A = n-mA/A )
     ( n-mPA = n-mPA )
     ( nen/S = nen/S ) ( nen/A = nen/A )
     ( s-kse/S = s-kse/S ) ( s-Te/S = s-Te/S )
     ( s-:/S = s-:/S ) ( s-:/A = s-:/A )
     ( is/A = is/A )
     ( s-he/S = s-he/S )
     ( t-e/S = t-e/S ) ( t-e/A = t-e/A ) ( t-:/S = t-:/S )
     ( s-nTe = s-nTe ) ( Ut = Ut )
     ( 0-:/S = 0-:/S ) ( 0-:/A = 0-:/A )

     ( tA-s/V = tA-s/V )
     ( tA-t,s/V = tA-t,s/V )
     ( tÅ-s/V = tÅ-s/V )
     ( te-s/V = te-s/V )
     ( i-0/V = i-0/V )
     ( E-0/V = E-0/V )
     ( (ä)y/V = (ä)y/V )
```

```
      ( (e)le/V = (e)le/V )
      ( (a)ise/V = (a)ise/V )
      ( (o)i/V = (o)i/V )
      ( sE-0/V = sE-0/V )
      ( ksE-s/V = ksE-s/V )
      ( (e)ne/V = (e)ne/V )
      ( (A):/V = (A):/V )
      ( A/V = A/V )
      ( (i)A/V = (i)A/V )
      ( ise/V = ise/V )

      ( § = §§ § )
      END

LEXICON 0-i/S        0 S0 "";  i S12 ""
LEXICON 0-i/A        0 S0 "";  i A12 ""
LEXICON mi/S         mi S0 "";  mE S12 "";  n S3 ""
LEXICON si/S         si S0 "";  sE S2 "";  TE S13 ""
LEXICON si/A         si S0 "";  sE A2 "";  TE A13 ""
LEXICON psi/S        psi S0 "";  psE S12 "";  s S3 ""
LEXICON tsi/S        tsi S0 "";  tsE S12 "";  s S3 ""
LEXICON ksi          ksi S0 "";  ksE S2 "";  hTe S1 "";  htA P "PTV SG"
LEXICON 0-e/S        0 S03 "";  E S12 ""
LEXICON r/S          $r S03 "";  rE S12 ""
LEXICON n-me/S       $n S03 "";  mE S12 ""
LEXICON n-me/A       $n S03 "";  mE A12 ""
LEXICON n-mA/S       $n S03 "";  mA A12 ""
LEXICON n-mA/A       $n S03 "";  mA A12 ""
LEXICON n-mPA        n S03 "";  mPA S12 ""
LEXICON nen/S        nen K "";  sE S123 "";  s§ R "NOM SG"
LEXICON nen/A        nen K "";  sE A123 "";  s§ R "NOM SG"
LEXICON s-kse/S      s S03 "";  ksE S12 ""
LEXICON s-Te/S       s S0 "";  TE S13 "";  ks S2 ""
LEXICON s-:/S        $s S03 "";  : S12 ""
LEXICON s-:/A        $s S03 "";  : A12 ""
LEXICON is/A         $is S03 "";  i: A12C1 "";
                     ein S03 "SUP"; eimPA S12 "SUP"; eimmin K "SUP MAN"
LEXICON s-he/S       s S03 "";  hE S12 ""
LEXICON t-e/S        t S03 "";  .e S1 "";  E S2 ""
LEXICON t-e/A        t S03 "";  .e A1 "";  E A2 ""
LEXICON t-:/S        t S03 "";  : S12 ""
LEXICON s-nTe        s S0 "";  nTe S1 "";  ns S2 "";  0 S4 ""
LEXICON Ut           Ut S03 "";  e: A12 ""
LEXICON 0-:/S        $ S04 "";  : S12 ""
LEXICON 0-:/A        $ S04 "";  : A12 ""

LEXICON tA-s/V       TA V023 "";  s V1 ""
LEXICON tA-t,s/V     TA V023 "";  T V1 "";  s V1 ""
LEXICON tÅ-s/V       TÅ /V "";  s V1 ""
LEXICON te-s/V       TE V023 "";  s V1 ""
LEXICON i-0/V        i V03 "";  0 V12 ""
LEXICON E-0/V        E V012 "";  0 V3 ""
LEXICON (ä)y/V       y V03 "";  v V12 ""
LEXICON (e)le/V      lE V012 "";  $l V3 ""
LEXICON (a)ise/V     isE V012 "";  $is V3 "";  is V3 ""
LEXICON (o)i/V       i V03 "";  0 V12 "";  itsE V012 ""
LEXICON sE-0/V       sE V012 "";  0 V3 ""
LEXICON ksE-s/V      ksE V012 "";  s V3 ""
LEXICON (e)ne/V      nE V012 "";  $t V3 ""
LEXICON (A):/V       : V0 "";  s V1 "";  0 V2 "";  $t V3 ""
LEXICON A/V          .A V02 "";  s V1 "";  0 V2 "";  $t V3 ""
LEXICON (i)A/V       .A V02 "";  s V1 "";  $t V3 ""
LEXICON ise/V        isE V012 "";  is V3 ""; AjA V02 ""
```

```
LEXICON S0
     0          K     "NOM SG";
     §          R     "NOM SG"

LEXICON S1
     $+n        K     "GEN SG";
     $+n§       R     "GEN SG";
     +n         P1    "NOM SG/PL / GEN SG";
     +nA        P     "ESS SG";
     !+A        P     "PTV SG";
     $+ksi      K     "TRA SG";
     $+kse      P1    "TRA SG";
     $+ssA      P     "INE SG";
     $+stA      P     "ELA SG";
     3+:n       P     "ILL SG";
     4+h:n      P     "ILL SG";
     5+seen     P     "ILL SG";
     $+llA      P     "ADE SG";
     $+ltA      P     "ABL SG";
     $+lle      P     "ALL SG";
     $+ttA      P     "ABE SG";
     $+t        K     "NOM PL";
     %+en       P     "GEN PL"

LEXICON S2
     &+ien      P     "GEN PL";
     $6+iden    P     "GEN PL";
     $6+itten   P     "GEN PL";
     +inA       P     "ESS PL";
     7+iA       P     "PTV PL";
     $6+itA     P     "PTV PL";
     $+iksi     K     "TRA PL";
     $+ikse     P1    "TRA PL";
     $+issA     P     "INE PL";
     $+istA     P     "ELA PL";
     2+iin      P     "ILL PL";
     8+ihin     P     "ILL PL";
     9+isiin    P     "ILL PL";
     $+illA     P     "ADE PL";
     $+iltA     P     "ABL PL";
     $+ille     P     "ALL PL";
     $+ittA     P     "ABE PL";
     +ine       P     "CMT";
     $+in       K     "INS PL"

LEXICON S3
     1+tA       P     "PTV SG";
     2+ten      P     "GEN PL"

LEXICON S4
     +ttA       P     "PTV SG"

LEXICON S5
     $+inA      P     "ESS PL";
     $8+ihin    P     "ILL PL";
     $+ine      P     "CMT"

LEXICON C1
     $´mpi      S0    "CMP";
     $´mPA      S12   "CMP";
     $´mmin     K     "CMP MAN";
     $+sti      K     "POS MAN"
```

```
LEXICON C2
     $+in       S03   "SUP";
     $+imPA     S12   "SUP";
     $+immin    K     "SUP MAN"

LEXICON P
     /ni        K     "SG1";
     /si        K     "SG2";
     /nsA       K     "SG3/PL3";
     /:n        K     "SG3/PL3";
     /mme       K     "PL1";
     /nne       K     "PL2"

LEXICON K0
     _hAn       §     "han";
     _kA:n      §     "kaan";
     _kin       §     "kin";
     _kO        §     "ko";
     _pA        §     "pa";
     _kinkO     §     "kin ko";
     _kA:nkO    §     "kaaan ko";
     _kOhAn     §     "ko han";
     _kOs       §     "ko s";
     _pAhAn     §     "pa han";
     _pAs       §     "pa s"

LEXICON K1
     _s         §     "s"

LEXICON §§
     X          §     "Dummy"


LEXICON V0
     (        1-3-N   "PRES ACT";
     ($         K01   "IMPV ACT SG2";
     (mi nen/S        "DVMI";
     (mA        /S    "DVMA ACT";
     (vA        /A    "PCP1 ACT";
     (mAtTO n-mA/A    "PCP3"

LEXICON V1
     +i)        1-3   "PAST ACT"

LEXICON V2
     (isi)     1-3-N  "COND ACT"

LEXICON V3
     (ne      1-3-N   "POTN ACT";
     (kOOn      K     "IMPV ACT SG3";
     (kAAmme    K     "IMPV ACT PL1";
     (kAA       K01   "IMPV ACT PL2";
     (kAAtte    §     "IMPV ACT PL2";
     (kOOt      §     "IMPV ACT PL3";
     (kO        §     "IMPV ACT NEG";
     *$DA       4-N   "PRES PSS";
     *$Ztiin    K     "PAST PSS PE4";
     *$ZtAisi   4-N   "COND PSS";
     *$ZtAne    4-N   "POTN PSS";
     *$ZtAkO    4-N   "IMPV PSS";
     (DA        K     "INF1 NOM";
     (DA+kse    P1    "INF1 TRA";
     +De+ssA    P     "INF2 ACT INE";
```

```
      +Den       K     "INF2 ACT MAN";
      *$ZtAessA  K     "INF2 PSS INE";
      *$ZtAmAn   K     "DVMA PSS INS";
      *$ZtAvA    /A    "PCP1 PSS";
      (n Ut            "PCP2 ACT";
      *$ZTU      /A*   "PCP2 PSS"

LEXICON Sg1-Pl3
      $n         K     "SG1";
      $t         K     "SG2";
      :          K     "SG3";
      $mme       K     "PL1";
      $tte       K     "PL2";
      vAt        K     "PL3"

LEXICON Neg      $ K   "NEG"

LEXICON Pe4      :n K  "PE4"

LEXICON Root
      kaTo             /S    "Failure of Crops S";
      katTo            /S    "Roof S";
      liuKu            /S    "Glide S";
      puKu             /S    "Dress S";
      aiTo             /A    "Genuine A";
      tunnetTu         /A    "Well-known A";
      auti.o           /A    "Desert A";
      risti            /S    "Cross S";
      kivE             /S    "Stone S";
      läPE             /S    "Hole S";
      kalsium 0-i/S          "Calcium S";
      koira            /S    "Dog S";
      märKä            /A    "Wet A";
      kalÅ             /S    "Fish S";
      pahÅ             /A    "Bad A";
      raaKÅ            /A    "Raw A";
      harakKÅ          /S*   "Magpie S";
      yksikKö          /S*   "Unit S";
      rohke.a          /A    "Brave A";
      ma:              /S    "Earth S";
      sukla:           /S    "Chocolate S";
      kielE            /S    "Language S";
      lu mi/S                "Snow S";
      kä si/S                "Hand S";
      la psi/S               "Child S";
      vei tsi/S              "Knife S";
      ka ksi                 "Two";
      sisa r/S               "Sister S";
      runotTa r/S            "Muse S";
      polKi n-me/S           "Pedal S";
      onnetTo n-mA/A         "Unhappy A";
      pahi n-mPA             "Worst";
      hevo nen/S             "Horse S";
      rauhalli nen/A         "Calm A";
      kato s-kse/S           "Shed S";
      vastau s-kse/S         "Answer S";
      rakkau s-Te/S          "Love S";
      viera s-:/A            "Guest/Strange A";
      raitT is/A             "Sober A";
      mie s-he/S             "Man S";
      kevy t-e/A             "Light-weight A";
      kevä t-:/S             "Spring (time) S";
      kahdeksa s-nTe         "8th";
```

```
        kuoll Ut            "Dead A";
        hankKe 0-:/S        "Project S";

        kuTo  /V            "Knit V";
        muista /V           "Remember V";
        venytTA /V          "Stretch V";
        huu tA-s/V          "Shout V";
        kiel tA-s/V         "Deny V";
        sou tA-t,s/V        "Row V";
        kanTÅ /V            "Carry V";
        saar tÅ-s/V         "Surround V";
        haKE /V             "Fetch V";
        tun te-s/V          "Feel V";
        lähTE /V            "Depart V";
        vaaT i-0/V          "Demand V";
        luenno i-0/V        "Lecture V";
        sa: /V              "Get V";
        jaKe (e)le/V        "Distribute V";
        kä (ä)y/V           "Go/Happen V";
        valit sE-0/V        "Choose V";
        juo ksE-s/V         "Run V";
        ranKa (a)ise/V      "Punish V";
        paKe (e)ne/V        "Flee V";
        rohKe (e)ne/V       "Dare V";
        hakKa  (A):/V       "Hit V";
        raaha (A):/V        "Drag V";
        kiiPe A/V           "Climb V";
        helTi (i)A/V        "Loosen V";
        koKo A/V            "Assemble V";
        silpPu A/V          "Chop V";
        vär ise/V           "Vibrate V"

END
```

APPENDIX C
SAMPLE ANALYSES

A test run with the PASCAL version of the two-level program is given below. Input lines from the terminal are given in **bold** characters. Comments (in parentheses) have been added to the right of some input and output lines. Some points are further discussed at the end of this appendix.

```
run $twol/demo                          (the command to start the program)
§RUNNING 0513
§?                                            (the program asks for input)
0                                      (digits select options from a menu)
0: This Menu (of Main)
1: Quit
2: Generate                                (2 and 3 are the relevant ones)
3: Analyse
4: Dump Alphabet
5: Dump Automata
6: Dump Dictionary
7: Dump Alternations
8: Dump Alignment
3                                                (start the analysis mode)
70 Characters in the Alphabet (85)          (reading in the automata)
22 Automata Read In: 2469 States (3000)
                     171 Rows (200)
                     297 Columns (300)
111 Aligned Pairs (120)                                 (size of the CPS)
90 CONT Lists (150)  233 CONT Items (250)       (reading the lexicons)
67 Lexicons Read in (90)  788 Nodes (900)
721 Branches (900)  301 Entries (500)
627 Strings (1000) 3653 Characters (5000)
                                              (now the program is ready)

katto                                              (the word to be analysed)
katTo                                   (the lexical representation found)
Roof S NOM SG                                        (the lexical info)

katolla                                (configuration: k a t T o $ + l l A)
katTo$+llA                             (               k a t   o     l l a)
Roof S ADE SG                         (katTo "Roof S", $+llA "ADE SG"in Sl)

kattoamme                                       (katTo "Roof S" in Root)
katTo!+A/mme                                        (!+A "PTV SG" in Sl)
Roof S PTV SG PLl                                     (/mme "PLl" in P)

kato                                                  (there is also such
kaTo                                                a word in the lexicon)
Failure of Crops S NOM SG

kadon                                                    (but katto and kato
kaTo$+n                                                are kept apart, because
Failure of Crops S GEN SG                        the gradation is obligatory)

katon
katTo$+n
Roof S GEN SG

katos                                          (this is another noun entry
katos                                            which will be homographic
Shed S NOM SG                                    with katto in some forms)
```

```
katosta                                          (this has two analyses)
katosl+tA                                      (the first interpretation)
Shed S PTV SG
katTo$+stA                                          (and the second one)
Roof S ELA SG

hevonen                                          (hevo "Horse S in Root,
hevonen                                                nen "" in nen/S)
Horse S

hevosen                                                 (sE "" in nen/S)
hevosE$+n
Horse S GEN SG

hevoskatos                                              (a compound word)
hevos$katos
Horse S NOM SG Shed S NOM SG

kudomme                                          (examples of verb forms)
kuTo($mme                                         (kuTo "Knit V" in Root,
Knit V PRES ACT PL1                                 '(' "PRES ACT" in V0,
                                                  $mme "PL1" in Sgi-Pl3)
kutoisimme
kuTo(isi)$mme                          (.., (isi) "COND ACT" in V2, ..)
Knit V COND ACT PL1

hakkaavat                                (homography within a paradigm)
hakKa:(vAt
Hit V PRES ACT PL3
hakKa:(vA$+t
Hit V PCP1 ACT NOM PL

hakatuimmassa                                (three consonant gradations)
hakKa$t*$ZTU$+imPA$+ssA
Hit V PCP2 PSS SUP INE SG

0                                          (there is another menu in the
0: This Menu                                 analysis/generation module)
1: Quit (to Main)                      (to terminate or to switch mode)
2: Trace Dump for Previous Wd
3: Trace ON
4: Trace OFF
1                                             (return to the main program)
Quitting Xeq
2                                              (enter the generating mode)

katTo$+llA                                    (give a lexical representation)
katolla                                        (and you get the surface form)

2                                                  (now, take a trace dump)

 2  1   1 1 1 1 1 1 1 1 1 1 1 2 1 1 1   1 1 2 1 1  1  (1)  k
 3  2   1 1 1 2 2 1 1 1 1 1 1 1 1 1 1   2 1 2 1 1  1  (2)  k
 4  2   1 2 2 4 2 1 2 2 2 1 2 2 1 2 2 1   2 2 1 1 2  2  (2)  ka
 5  4   1 1 1 1 3 1 3 2 1 1 1 6 2 1 4 1   5 1 2 1 7  9  (3)  kan
 6  4   1 1 1 1 3 1 4 2 1 1 1 7 2 1 1 1   5 1 2 2 4 11  (3)  kat
 7  6   1 1 1 1 3 1 1 2 1 1 1 1 1 1 1 7   5 1 2 2 1 11  (4)  kat
 8  6   1 1 1 1 3 1 1 2 1 1 1 1 2 1 1 2   5 1 2 2 4  1  (4)  katt
 9  7   1 5 2 1 4 1 2 2 2 1 2 2 1 2 2 8   6 2 4 1 2  2  (5)  kato
10  8   1 5 2 1 4 1 2 2 2 1 2 2 1 2 2 3   6 2 4 1 2  2  (5)  katto
11  9   1 5 2 1 4 1 2 2 2 1 2 2 1 2 2 1   6 2 4 1 2  2  (6)  kato
12 11   1 5 2 1 4 1 2 2 3 2 2 2 1 2 2 1   6 2 4 1 2  2  (7)  kato
13 12   1 1 1 1 4 1 3 2 1 2 3 4 2 1 3 1 10 1 2 1 6  4  (8)  katol
```

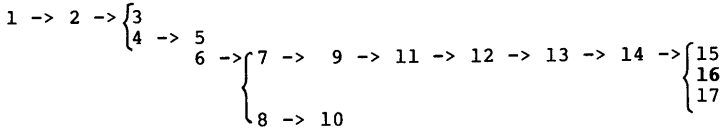
```
14 13   1 1 1 1 4 1 1 2 1 2 1 1 2 1 1 1 10 1 2 1 6  1  (9)  katoll
15 14   1 1 1 2 4 1 1 2 1 2 1 1 1 1 1 1 10 1 2 1 1  1  (10)  katoll
16 14   1 2 2 4 4 1 2 2 2 2 2 2 1 2 2 1 13 2 1 1 2  2  (10)  katolla
17 14   1 3 2 1 7 1 2 2 2 2 2 2 1 2 2 1 13 2 1 1 2  2  (10)  katolle

ristI$+issA
risteissä

2

 2  1   1 1 1 1 1 1 1 1 1 1 1 2 1 1  1  1 1 2 1 5 1  (1)  r
 3  2   1 3 2 1 2 2 2 1 2 1 2 2 1 2 2  1  2 2 1 1 2 2  (2)  re
 4  2   1 4 2 1 2 5 2 1 2 1 2 2 1 2 2  1  2 2 3 1 2 2  (2)  ri
 5  4   1 1 1 1 3 1 4 1 1 1 1 8 2 1 1 12  5 1 2 1 8 5  (3)  ris
 6  5   1 1 1 1 3 1 1 1 1 1 1 1 2 1 1  1  5 1 2 2 4 1  (4)  rist
 7  6   1 3 2 1 4 2 2 1 2 1 2 2 1 2 2  1  6 2 1 1 2 2  (5)  riste
 8  6   1 4 2 1 4 5 2 1 2 1 2 2 1 2 2  1  6 2 3 1 2 2  (5)  risti
 9  7   1 3 2 1 4 2 2 1 2 1 2 2 1 2 2  1  6 2 1 1 2 2  (6)  riste
10  8   1 4 2 1 4 5 2 1 2 1 2 2 1 2 2  1  6 2 3 1 2 2  (6)  risti
11  9   1 3 2 1 4 4 2 1 3 2 2 2 1 2 2  1  6 2 1 1 2 2  (7)  riste
12 10   1 4 2 1 4 7 2 1 3 2 2 2 1 2 2  1  6 2 3 1 2 2  (7)  risti
13 11   1 4 2 1 4 1 2 1 5 2 2 2 1 5 2  1  7 3 1 1 3 2  (8)  ristei
14 11   1 1 2 1 4 1 1 1 4 2 1 1 2 1 1  1 10 1 2 1 1 1  (8)  ristej
15 13   1 1 1 1 4 1 4 1 1 2 1 8 2 1 1 12 10 1 2 1 8 5  (9)  risteis
16 15   1 1 1 1 4 1 1 1 1 2 1 8 2 1 1 12 10 1 2 1 8 1  (10)  risteiss
17 16   1 1 1 2 4 1 1 1 1 2 1 8 1 1 1  1 10 1 2 1 1 1  (11)  risteiss
18 16   1 3 2 1 7 1 2 1 2 2 2 2 1 2 2  1 13 2 1 1 2 2  (11)  risteisse
19 16   1 8 2 4 4 1 2 1 2 2 2 2 1 2 2  1 13 2 1 1 2 2  (11)  risteissä

1                                   (from the analysis/generation menu)
Quitting Xeq                           (returning to the main program)
1                                         (from the main program menu)
Quitting (Main)                              (terminating the program)
```

* * * * *

Notes on the traces

The above session contains two trace dumps. These indicate the detailed procession of the two-level program while it produces word-forms. Similar traces are available in the analysis mode, but they are not shown here because they contain more "noise" steps and are thus less transparent. The tracing can be controlled by menu options 3 and 4 or they can be taken afterwards by option 2 as above.

The trace dump has a line corresponding to each step (or task) of the production process (c.f. section 4.4). The steps are given in the chronological order according to the breadth-first strategy. The two first columns show the nondeterministic procession: the first is just a sequence number, and the second indicates the "parent" step of the task, i.e. the previous step in this branch of the process. In the first of the above examples the succession is as follows:



In the middle of the trace, there is a zone of 22 numbers in tight columns, one for each automaton listed in appendix A. The relative position of the column indicates the numbet of the automaton, e.g. the eighth column corresponds to the automaton (8), which handles

the vowel harmony. The numbers in the columns indicate the states in which the automata are at that point of the process. E.g. the vowel harmony automaton starts in the front harmony state 1, but switches over to the back harmony state 2 at the second letter (a) and stays there. The column 12 in this area shows the procession of the automaton for the weakening of t:

Step:	2	4	6	7	9	11	12	13	14	15
State:	1	2	7	1	2	2	2	4	1	2
Pair:	k	a	t	T	o	$	+	l	l	A
	k	a	t		o			l	l	a

One can deduce the character pairs which cause the transitions by using the remaining left part of the trace. The number in parentheses tells the number of characters consumed from the input. In the production it gives the lexical character. The partial result, in this case the partial surface form tells the other side.

This kind of tracing is the principal means of checking and debugging rule automata. Often the problems are detected while analyzing word forms, but it is practical to switch over to the producing mode to get a cleaner trace. It will be this way, because the model and the program only allow deletion of lexical characters, but not epenthesis. The number of false branches in the analysis is fairly sensitive to the choice of lexical representations. The branching can be reduced by the use of morphophonemes or other markings instead of plain ordinary characters in positions where deletion is effective, because this will reduce the degree of nondeterministic branching. Otherwise, an attempt will be made to insert a lexical character which could possibly be deleted between **every** two characters, and in many cases there will be some matching start, especially at the beginnings of the words. The description of Finnish presented above is a compromise which uses some marking in order to reduce extraneous branching, but is far from optimal in that respect. Its main objectives have been linguistic for which reason it has mostly ignored the concern for efficiency.

PUBLICATIONS OF THE DEPARTMENT OF GENERAL LINGUISTICS, UNIVERSITY OF HELSINKI

1. Raimo ANTTILA, Analogy. 1974. 203 pp. (Out of print.)
2. Finnish Structuralism: Present and Past. 1976. 53 pp. (Out of print.)
3. Eugene HOLMAN, Grade Alternation in Baltic-Finnic. 1975. 216 pp.
4. Esa ITKONEN, Linguistics and Empiricalness: Answers to Criticisms. 1976. 76 pp. (Out of print.)
5. Four Linguistic Studies in Classical Languages. 1978. 121 pp. (Out of print.)
6. Auli HAKULINEN - Fred KALRSSON - Maria VILKUNA, Suomen tekstilauseiden piirteitä: kvantitatiivinen tutkimus. 1980. 189 pp. (Out of print.)
7. Benny BRODDA - Fred KARLSSON, An Experiment With Automatic Morphological Analysis of Finnish. 1981. 97 pp. (Out of print.)
8. Martti NYMAN, Relational and Reconstructive Aspects of Grammatical Systematization. 1982. 64 pp.

9-10. Papers from the Seventh Scandinavian Conference of Linguistics, Vol. I, II. Edited by Fred Karlsson. 1983. 720 pp.

11. Kimmo Koskenniemi, Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production. 1983. 160 pp.

ISBN 951-45-3201-5
ISSN 0355-7170
Offset Oy 1983