MORPHOLOGY
in the
EUROTRA BASE LEVEL CONCEPT

by Peter Lau and Sergei Perschke
Commission of the EC,
Bât. JMO
L - 2920 Luxembourg

ABSTRACT

In recent years the nature and the role of a morphological component in NLP systems has attracted a lot of attention.

The two-level model of Koskenniemi which relates graphemic to morphological structure has been succesfully implemented in the form of finite state automata.

In EUROTRA a solution which combines morphological and surface syntactic processing in one CFG implemented in a unification grammar framework has been tried out. This article contrasts these two approaches considering especially the feasibility of building morphological modules for a big multilingual MT system in a decentralised R & D project.

0. INTRODUCTION

The development of sophisticated NLP applications has created a need for specific processing in order to be able to cope with large vocabularies without creating monstruous dictionaries. Earlier approaches often avoided morphology more or less by listing full wordforms in the dictionary or by simply segmenting some inflectional endings with a few general rules.

Much recent work is based on the Two-level Model (Koskenniemi, 1983) and relates directly or indirectly to the original implementation of this model in the form of finite state transducers (FST). The original notation and implementation have been further developed and refined (cf. e.g. Black, 1986 and Bear, 1986) in order to improve compilation and runtime, debugging and rule-writing facilities.

Still some problems persist and others have not been touched yet. This paper presents an alternative, but not contradictory, solution which has to some extent been tried out in the EUROTRA Machine Translation Project and argues that the two-level approach may not be entirely viable in a decentralised R&D project which aims at the creation of a big multilingual MT system.

I. THE TWO-LEVEL MODEL

The original presentation of the model (Koskenniemi, 1983) shows that it is possible to treat the inflectional morphology (including spelling rules) of a highly inflected language like Finnish by establishing correspondences between a surface alphabet and a lexical alphabet (the two levels) and using a lexicon to determine which combinations of characters and morphemes are legal. Moreover, this is done by means of declarative rules, thereby avoiding the procedural problems of generative phonology, and the algorithm used is language independent. Together with the fact that the model may be used for synthesis as well as for analysis this is a strong argument in favour of employing a two-level approach to morphology.

Later work points to some important shortcomings of the original implementation of the model in the form of FST's (Black, 1986). Especially compilation and runtime requirements and debugging are seen to pose severe problems. In Black's words:"Debugging automata is reminiscent of debugging assembly language programming in hex". Considering that the (linguistic) user is interested in the rules rather than in the low-level implementation of them, Black et al. have proceeded to develop high-level notations in the form of rules which are interpreted directly, instead of being compiled into FST's.

Nonetheless, they entirely respect the two-level approach in their notation. Their rules still establish correspondences between, on one side, elements of a lexical alphabet (the characters of the natural language alphabet plus the empty character (Ø), the morpheme boundary (+), and archiphonemes (noted as capital letters)) and, on the other side, the elements of a surface alphabet (the characters of the natural language plus the empty character), and they use a lexicon to determine which combinations of characters make up legal morphemes. Their work shows the relative independence of the rule formalism from its implementation - accepting the two-level model by no means forces one to accept FST's as an implementation vehicle - and it shows that the rules for combination of characters (spelling rules or morpho-graphemics) are best treated in isolation from the rules for combination of morphemes (morpho-syntax).

This latter approach has been further developed by Bear (Bear, 1986). He combines a two-level approach to morpho-graphemics with a unification grammar approach (a modified PATR rule interpreter) to morpho-syntax. The resulting

implementation preserves the generality and flexibility of the treatment of morpho-graphemic phenomena like allomorphy while, at the same time, avoiding the problems of treating morpho-syntax in the lexicon, which in reality is what happens in Koskenniemi's original model where the lexical entries for root morphemes are marked for "continuation classes" (references to sub-lexicons which determine the legal combinations of morphemes).

Furthermore, by treating morpho-syntax in a unification grammar framework, Bear obtains an effect which is very important provided that morphological analysis and synthesis are normally regarded as elements or modules of systems which also do other kinds of language processing, e.g. syntactic parsing: He reaches a stage where the output of the morphological analyser is something which can easily be used by a parser or some other program (Bear, 1986, p. 275).

Still, one must admit that only subsets of morphology have been treated within the two-level framework and its successors. Most of the work seems to have centred on inflectional morphology with a few excursions into derivation and a total exclusion of compounding which is a very important phenomenon in languages like German, Dutch and Danish. It is also noteworthy that none of the implementations mentioned above could be used for the analysis (or synthesis) of running text because they know no capital letters, no numbers, no punctuation marks or special characters, nor formatting information. This does not mean that such things could not be taken care of in combination with a two-level framework (for instance by a pre-processor of some kind), it just means that in order to cater for them one needs new kinds of notations and implementations (as numbers could hardly be analysed as lexicon entries) with the corresponding interfacing problems (cf. Bear's motivation for using a unification grammar for morpho-syntax).

II. THE EUROTRA BASE LEVEL

1. Background

EUROTRA is a decentralised R & D project aiming at the development of a multilingual machine translation system. Thus, on top of the classical coder consistency problems known from the development of big MT systems like SYSTRAN, EUROTRA has to ensure consistency of work done in some 20 geographically dispersed sites. This calls for a strong, coherent, understandable, problem oriented and comprehensive framework. Considering also that the software development in the project is supposed to be based on rapid prototyping, it becomes clear that the project has to build on some general idea about how things will fit together in the end. We cannot afford to build independent modules (e.g. an FST implementation of a morphological component, a PATR-II grammar for our syntactic component implemented in PROLOG, some SNOBOL programming for the treatment of text formatting, special characters etc, and a relational database for our dictionaries) and then start caring about the compatibility of these modules afterwards.

Consequently, the EUROTRA base level which treats all kinds of characters (alpha-numeric, special, control etc.) and morphemes and words has been conceived as a part of the general EUROTRA framework and described in the same notation as the syntactic and semantic components.

In the absence of a dedicated user language (which is being developed now) the EUROTRA notation is the language of the virtual EUROTRA machine. This virtual machine stipulates a series of so-called generators (G's) linked by sets of translation rules (t-rules). Each generator builds a representation of the source text (in analysis) or the target text (in synthesis) and it is the job of the linguists who are building the translation system to use these generators in such a way that they construct linguistically relevant levels of representation (e.g. morphological, syntactic constituent, syntatic relation and semantic representations). The individual generators are unification grammars consisting of constructors which are basically functions with a fixed number of arguments and atoms which are constructors with no arguments.

An atom has the form

$$(\text{name}, \{\text{feature description}\})$$

The feature description is a set of attribute-value pairs (features) with one distinguished feature, called the name, which is caracteristic for each generator (e.g., for the surface syntactic generator it would be syntactic category). The name is placed outside the curly brackets, and only the value is given.

A constructor has the form

$$(n, \{fd\})\ [(n_1, \{fd_1\}) \ldots .(n_n, \{fd_n\})]$$

H E A D A R G U M E N T S

where the n=name and fd=feature description. In functional terms this represents a function (described by the head) over n arguments.

The t-rules relate the representation built by a generator to the atoms and constructors of the subsequent G, thereby making it possible for this G to build a new representation of the translation of the elements of the preceding one in a compositional way (cf. EUROTRA literature (2,3 and 4) in the reference list).

The virtual machine has been implemented in PROLOG and an Early-type parser has been used to build the first representation in analysis (viewed as a tree-structure over the input string). This implementation, of course,

represents a choice. Other programming languages and parsers might have been used. The system implemented by Bear, e.g. indicates that a two-level approach to morpho-graphemics may be combined with a unification grammar approach to morpho-syntax. For various reasons, though we have not chosen this solution.

2. Text structure and lexicographic consistency

The first serious problems encountered in choosing a two-level approach to morphology in an MT system is the question of what to do with all those characters which are not letters. If we find a piece of text like

A. This question will be discussed with the Director General on April 25th.

we do not want an analysis which tells us that the system has found 4 nouns (one being a 'proper' noun), 3 verbs (one finite, two infinites), two determiners, two prepositions and some unintelligible elements which another machine will have to take care of. We want to know that "Director General" is a compound which, syntactically, behaves like a single noun, that "April 25th" is a date (because it may be a time-modifier of a sentence), that "A" is an index which indicates some enumerative structure of the text, that "." is a punctuation mark which may indicate that a sentence ends here, and probably more information which we need if we want to build a representation of the whole text and not just of some selected words or simple sentences.

It seems difficult to see how the two-level approach could cope with compounds, apart from entering them all into the lexicon, and this would really be a heavy burden on the lexicon of compounding languages. Single letters like "A." and even punctuation marks might be included in the lexicon, but numbers could not for obvious reasons.

Furthermore, control and escape sequences which determine most of the text structure (font, division into chapters, sections, paragraphs etc.) in any editor or word processor might be entered into the lexicon, but the two-level approach does not provide any solution to the problem of giving these sequences an interpretation which is useful in building a representation of the text structure.

In order to cope with these problems, we have chosen, in EUROTRA, to define the input and the output of the system as extended ASCII files. The ASCII characters, including numbers, special and control characters, are defined as the atoms of the first level of representation and thereby provided with an interpretation which makes it possible for them to serve as arguments of constructors which build a tree-structure representing the text and all its elements, also those elements which are not words.

The second problem in the two-level approach is that, apart from the fact that some textual elements seem to be totally outside the scope of the lexicon, even those elements which go into the lexicon pose a series of problems in our context.

For MT to be of any use and efficiency we need large dictionaries which cover a substantial part of the vocabularies of those languages treated by the MT system. It is known from a lot of MT systems that the coding of large dictionaries (or lexica) cannot be left to a small group of people working together in close contact for a limited period of time. Many coders working over long periods are needed, and they will constantly be maintaining, revising and up-dating the work of one another. For such an enterprise to succeed one needs extremely strong and detailed guidelines for coding, and the coding language should be as simple and transparent as possible and contain no contentious elements from a theoretical point of view. Morpheme boundaries, archiphonemes and null-characters are hardly uncontentious in the sense that, e.g. everybody agrees on the root form to employ in 'reduction' ('reduce' or 'reduc' ?), and even the slightest disagreement will invariably jeopardize the intercoder consistency which is absolutely necessary for an MT project to succeed.

3. Character normalization and morpheme identification

The atoms of the base level identify and interpret the characters of the input file in that the name of the atom unifies with the input character (for non-printable characters hexadecimal notation in quotes is used):

(A, { type = letter, subtype = vowel, char=a, case = upper })

(à, { type = letter, subtype = vowel, char=a, case = lower, accent = grave })

('1B', { type = control_char, subtype = escape })

In a unification grammar which allows the use of named and anonymous variables, it is easy to join all variants of the letter 'a' under one heading (a constructor in EUROTRA terms) and percolate all relevant features to this heading by means of feature-passing. This is called normalisation in our terms, and it simply means that all typographical variants of a character are collapsed so that the dictionary will only have to contain one character type. A normalizing constructor for 'a' could be:

(a, {type = letter, subtype = vowel, case = X, accent = Y})

[('?, {char = a, case = X, accent = Y})]

where '?' is the anonymous variable. The argument of this constructor will unify with any atom containing the feature 'char = a' and accept the values for 'case' and 'accent' found in these atoms. By feature-passing these values will then be percolated to the head.

At this stage the representation of the input file is a sequence of normalised characters. This sequence is now matched against the dictionary or lexicon which is just another set of constructors of the form

(for, { class = basic_word, type = lexical, cat = prep, paradigm = invariant })

[f, o, r]

(for, { class = basic_word, type = prefix, paradigm = derivation })

[f, o , r]

Matching here means the kind of matching which occurs in unification. This means, of course, that the overgeneration may be severe in some case, e.g. each of the 's' appearing in Mississippi will i.a. be interpreted as a plural morpheme. This overgeneration must be constrained. We are working with this problem and some results are ready, which confirm that our approach to character normalisation and dictionary look-up, i.e. the one described above, provides for a straight-forward, strict and yet perfectly understandable and uncontroversial coding of dictionary entries. The set of possible features and the co-occurrrence constraints holding between those features are defined in advance. What the dictionary coder has to do is to choose the relevant features for each lexical item (basic word in our terminology) and write them into the relevant constructor which will operate in total independence of any other constructor. There will be no problems with linking sub-lexicons or discussing morpheme boundaries, because each constructor operates directly on the sequence of surface characters, i.e. the problem of whether the surface form of 'ability' is a b i l Ø i t y or a b i l Ø Ø i t y does not exist (cf. Black 1986, p. 16). The ensuing problems in relation to the treatment of allomorphy are exposed below.

4. Implementation

The EUROTRA Base Level has been implemented by means of a prototype version of the virtual machine implemented in PROLOG with an Early-type parser. This prototype was constructed in such a way that the parser would only work in one of the generators, i.e. the first generator employed in analysis, while the other generators would produce transforms of the tree-structure built by the first generator.

Due to this constraint, we had to collapse morpho-syntax and surface syntax into one generator which built a tree over the sequence of characters of the input file via normalized characters, basic words, complex words (inflected, derived and compound wordforms), phrasal nodes (NP, VP, PP etc.) and ending at an S top node. The resulting grammars became very big, and testing in most cases had to be done with sub-grammars in order to prevent loading and parsing times from becoming prohibitive.

Actual implementation work was done in 5 languages (English, German, Dutch, Danish and Greek), and several sub-grammars were successfully implemented and tested. The most important experience was that the different groups participating in the project were able to understand the base level specifications and to use them or deviate from them in a principled way producing comparable results.

The prototype used for this first implementation, however, was a fairly unelegant and user-unfriendly machine which was rather intended to be running specifications than a vehicle of constructing and testing grammars. With a more streamlined prototype two constraints on implementation and testing of grammars would be relieved: loading and runtime requirements would diminish radically and it should be possible to use parsing or parsing-like procedures in more than one generator.

This would allow us to construct a full MT system with a standardised and simple dictionary format and capable of treating all kinds of characters which may appear in an input file.

5. The base levels

The linguistic specifications of this system, which is to be implemented in the present phase of the project, have been elaborated in some detail. The input to the system will be files containing characters in a 7 or, preferably, 8 bit code (in order to cover the multilingual EUROTRA environment). The characters unify with atoms of the type described above. The atoms then unify with abstract wordform, sentence, paragraph etc. constructors of the following kind:

(wordform) [+(?, {type = letter})]

(sentence) [+ wordform, (?, {type = punctuation_mark})]

(paragraph) [+ sentence, (fin_paragraph, {char = double_CR})]

where ? is still the anonymous variable, '+' is the Kleene plus signifying one or more of the following argument and 'double carriage return' is assumed to be the character (or sequence) indicating termination of a paragraph in the text.

These abstract constructors will build a tree-structure representing the full input text from the characters via the words, the sentences, the paragraphs, the sections etc. to a top T(ext) node, of course with some overgeneration, e.g. some punctuation marks do not terminate a sentence, but the overgeneration will be filtered out by subsequent generators using morphological, syntactic and semantic information.

The generator following the first (text structure) level will normalise the characters by a many-to-one mapping of, e.g. variants of 'a', and all the basic words of the system component (e.g. the English analysis component), i.e. the major part of the monolingual dictionary, will be present in this generator in the form of constructors (cf. the 'for' constructor mentioned above). This will cause some overgeneration as illustrated above with the example 'Mississippi' but an abstract wordform constructor which is connected by a t-rule to the representations built by the abstract wordform constructor of the previous (text structure) level will filter out spurious results:

(wordform) [+(?, {class = basic_word})]

Given that 'mi', 'i' and 'ippi' are not all basic words of English, no interpretation of the 's' as plural or third person singular markers will be allowed, because each wordform has to cover exactly one sequence of basic words exhaustively without overlapping.

Assuming that 'Mississippi' is a basic word of English present in the dictionary (as a constructor of this level), the sequence of normalised characters 'mississippi' will receive at least one legal interpretation which is then translated into the subsequent (morpho-syntactic) level by a t-rule.

The treatment of allomorphic variation in this approach will rely on alternating arguments in the basic word constructors. In order to cover the alternation y - ie found in, e.g., city --> cities' we shall have to use a basic word constructor of the following form:

(city, { }) [c, i, t, (i;y)]

where ';' is the alternation operator. This constructor will unify with any of the two sequences 'citi' and 'city', and if we create two basic word constructors over the plural ending of nouns (covering at the same time the third person singular of the present tense of verbs), i.e. (s) and (es), e.g.

(es, { }) [e, s]

we may cover the wordform 'cities' by (citi) and (es). A definite advantage of using this approach is that it covers allomorphic variation inside the root form like in German plural of nouns:

Mann --> Männer

by (mann, { }) [m, (a, ä), n, n]

The only way of covering this phenomenon in the two-level approach seems to be by entering both 'Mann' and 'Männ' into the dictionary as possible roots.

The generator following the level where basic word identification takes place contains, as its atoms, the basic words translated by t-rules from the representations built by the basic word constructors. The characters, which are the atoms of the previous level, are cut off by receiving a 0 translation.

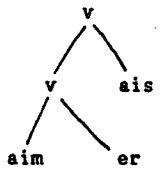
The constructors of this generator are wordform (or complex word) constructors covering the various inflectional paradigms, the different classes of derivation and compounding. The following constructor would build representations of all French verbs of the regular er-paradigm in the infinitive, including the information that these representations may be used as arguments of constructors over future and conditional forms (which include the infinitive):

(v, {class = wordform, cat = v, lexical_unit = X,
verbform = infinitive,
inflectional_class = regular_verb_er,
inflectional_paradigm = inf_cond_fut})

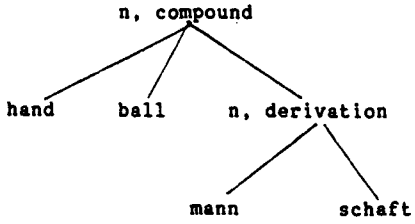
[(X, {class = basic_word, type = lex,
inflectional_class = reg_verb_er})

(er, {class = basic_word, type = inflection,
inflectional_class = reg_verb_er,
inflectional_paradigm = inf_cond_fut})]

The constructor over conditional forms will take this representation plus a basic word representing a conditional ending as its arguments, and the final representation of, e.g. 'aimerais' will be equivalent to a tree with all relevant information percolated to the top node:

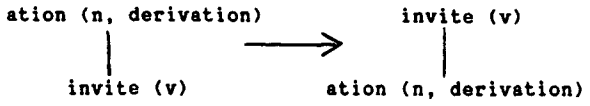


The morpho-syntactic generator builds the same kind of representations of derivations and compounds. The leaves of the trees always correspond to basic words, and consequently, this generator will build representations of, e.g. all compounds the elements of which are present in the basic word identification generator:



The morpho-syntactic representations are translated into the following (surface syntactic) level in such a way that wordforms which are exhaustively described by their top node (invariant words, inflections and some derivations like the agentive (e.g. 'swimmer')) appear as atoms, while all others (all other derivations and compounds) appear as structure (constructors) with the relevant categorial information in the top node:

At subsequent deep syntactic or semantic levels information from other nodes of the word tree may be needed. This can be provided by letting t-rules transform the tree in such a way that the relevant information goes to the top node (e.g. if the frame of the root of a derivation is needed for semantic purposes, the root features are moved to the top of the tree). In this way relevant morphological information will always be available when it is needed:



The resulting tree is used in a deep syntactic or semantic generator where the information that this element was originally a derived noun is irrelevant, because the element has already been placed in the overall structure on the basis of this information. Nonetheless, the 'ation'-node is not cut off, because it is relevant for transfer to know that a verb-noun derivation and not just a verb is being translated.

III. CONCLUSION

The EUROTRA base levels build a full representation of the text structure by treating all characters of the input file including special and control characters. They normalise the characters in such a way that the system dictionary may function independently of lay-out, font and other typographic variations. They provide separate treatments of morpho-graphemics and morpho-syntax, and the representations of the words are of such a kind that they may be used not only for syntactic, but also for semantic processing.

At the same time, the dictionary entries are simple basic word constructors over sequences of characters. No specific phonological knowledge is required for the coding of these entries, and so a possible source of inconsistency among coders is avoided.

The fact that EUROTRA constructors closely resemble traditional rewrite rules together with the coocurrence restrictions imposed by the EUROTRA feature theory alleviates the debugging of grammars and dictionaries. No real programming experience in the classical sense is needed. The constructors, however, do not imply undirectionality like the rules of generative phonology. They work equally well both ways, and consequently, they serve for analysis as well as for synthesis. The constructors of a generator all apply in parallel, thereby avoiding the kind of interaction which is typical of ordered sets of rules.

This design, in our opinion, provides a good set of tools for ensuring consistent implementation of grammars and dictionaries across a decentralised and multilingual MT project.

REFERENCES

1. Ananiadou, Effie & John McNaught. A Review of Two-Level Morphology. Manchester 1986. Unpublished EUROTRA paper

2. Arnold, Douglas. EUROTRA: A European perspective on MT. IEEE Proceedings on Natural Language Processing, 1986

3. Arnold, D.J. & S. Krauwer, M. Rosner, L. des Tombe, G.B. Varile. The <C,A>, T Framework in EUROTRA: A theoretically committed notation for MT. Proceedings of COLING '86. Bonn, 1986

4. D.J. Arnold, L. Jaspaert, R. Johnson, S. Krauwer, M. Rosner, L. des Tombe, G.B. Varile & S. Warwick. A Mu-1 View of the <C,A> T Framework in EUROTRA. Proceedings of the Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages. Colgate University, Hamilton, New York, 1985.

5. Bear, John. A Morphological Recognizer with Syntactic and Phonological Rules. Proceedings of COLING '86. Bonn, 1986

6. Black, Alan W. Morphographemic Rule Systems and their Implementation. Unpublished paper, Department of AI, University of Edinburgh, 1986

7. Koskenniemi, Kimmo. Two-Level Morphology: A general computational model for word-form recognition and production. University of Helsinki, Department of General Linguistics, 1983.