LANGUAGE TECHNOLOGY ALGORITHMS INSPIRED BY AN AGGLUTINATIVE, FREE-PHRASE-ORDER LANGUAGE

Gábor Prószéky & Csaba Merényi
MTA-PPKE Hungarian Language Technology Research Group
and
MorphoLogic

Abstract

A crucial requirement for doing machine translation, machine-aided translation, information retrieval or computational information processing of texts of agglutinative languages, in general, is an efficient computational morphology. Having done the morphological processing of the word-forms of sentences, the sentence structure itself should be identified. This step is also common in all human languages. In case of agglutinative languages, like Hungarian, morphological processing provides a lot of information expressed by syntax in other, less-inflectional languages. Syntactic processing of highly inflectional languages, therefore, can use a lot of syntactic information identified by the morphological subsystem. It is needed, because the agglutinative nature of morphology results in free phrase order on the sentence level.

In the first part of the paper we introduce Humor, a morphological description formalism and algorithm inspired by an agglutinative language, Hungarian. In the second part of the paper we describe a syntactic parsing formalism and algorithm inspired by Hungarian, a free-phrase-order language, and show the way how it is used in the Hungarian–English MetaMorpho machine translation system.

1. Morphological analysis inspired by Hungarian, an agglutinative language

A crucial requirement for doing machine translation, machine-aided translation, information retrieval or computational information processing of texts of agglutinative and highly inflectional languages, in general, is an efficient computational morphology. Having done the morphological processing of the word-forms of sentences, the sentence structure itself should be basically identified. In case of agglutinative languages, like Hungarian, morphological processing provides a lot of information expressed by syntax in other, less-

inflectional languages. Syntactic processing, therefore, can use a lot of syntactic information already identified by the morphological system. It is needed, because the rich case system of Hungarian results in free phrase order on the sentence level. We describe here an approach to morphological analysis called Humor (High-speed Unification Morphology), which successfully copes with problems of agglutinative and other (highly) inflectional languages very effectively.

1.1 A morphological description formalism

Our approach to computational morphology belongs to the 'item-and-arrangement" paradigm. The method itself uses lexicons of allomorphs and adjacency restrictions between them. A feature system and a unification algorithm—or better said: checking a relation called unfiability —is also used, as described later. Concatenation points between morphs are defined by continuation classes, and the basic idea behind the algorithmic description is—instead of a real-time analysis of every morpho-phonological attribute—that there are no more active operations in this approach to computational morphology. Due to its minimal memory requirements even the first versions of the system (since 1991) could effectively run even on early PC's (Prószéky & Kis 1999).

Treatment of written word-forms is a starting point to a large set of computational linguistic applications. Since the introduction of two-level morphology by Koskenniemi (1983) morphological processing has been widely known as the first step of processing running texts. Here we stress the importance of Koskenniemi's dissertation, because in the same year another important book was published by Winograd (1983) on natural language processing which does not mention the term "morphology" at all. This book was meant as a general introduction to computational linguistic applications but dealt mainly with English.

In general, we can say that a morphological analyzer is able to

- identify the lexical form (stem) of the actual word-form and its part-of-speech:
  kesztyű > kesztyű[N]
- list the inflectional features of the actual word-form:
  kesztyűimet > kesztyű[N][PSsg1i][ACC]
- show the derivational suffixes attached to the actual word-form:
  tartónak > tart@ó[N][DAT]
- show the eventual compounding boundaries in the actual word-form:
  kesztyűtartóba > kesztyű|tart@ó[N][INE]

Generation of word-forms is a morphological synthesis producing the actual surface form of a lexical stem combined with given morpho-syntactic features (1).

(1) kesztyű[N][PSsg2i][ACC] > *kesztyűidet*

Morphology always meets phonology via a phonology–morphology interface, but an orthography–morphology interface should also be applied in computational linguistics. In the following examples we can see two morpho-phonological phenomena: application of vowel harmony (2) and vowel lengthening (3), while an orthography-motivated treatment of gemination of the Hungarian digraph *'ly'* is shown by (4):

(2) nyelv + vAl → nyelv + vel → *nyelvvel*
(3) technológia + vAl → technológiá + val → *technológiával*
(4) bagoly + vAl → bagolly + al → *bagollyal*

In the generative grammar formalism not only surface and underlying forms are used, but many layers in between them. The most famous morphological system without intermediate levels is two-level morphology, but the approach introduced in this paper is also belongs here. The entire algorithm shows features similar to the hypothesis according to which most segments of word-forms in agglutinative languages are handled as a whole un-analyzed construction by native speakers, instead of parsing them on-the-fly. Psycholinguists use the term "Gestalt" to refer to this phenomenon (Bryant & Miikkulainen 2001), but the idea is also known in the theoretical linguistic literature: "a psycholinguistic argument for treating (some) ending sequences as wholes comes from the observation that children acquiring inflectional languages seldom make errors involving the order of morphemes in a word." (Bybee 1985) Another source is Karlsson (1986): "The endings and entries are often listed as wholes, especially in close-knit combinations. Such combinations are often subject to bi-directional dependencies that are hard to capture otherwise". A good example is the linguistic tradition handling number and person combinations of Hungarian conjugation (Tompa 1962), where, for example, *-unk*[PSpl1] is treated as a single morph, although its *-n* refers to first person and *-k* means plural. There are several ways in which lexical forms of words may be constructed: full listing, minimal listing, methods with unique lexical forms and methods with phonologically distinct stem variants (Karlsson 1986). Full listing does not need rules at all, but it is implausible for agglutinative languages. Minimal listings need a quite large rule system in the case of highly inflectional languages, although their lexicons are relatively small. In methods based on unique lexical forms allowing diacritics and morpho-phonemes paradigms are represented by a single base form (Koskenniemi 1983; Abondolo 1988). This is one of the reasons why it is very difficult to add new entries to the lexicons automatically in real language technology environments. Our approach is close to the minimal listing methods, but fewer rules are needed. Finally, the representation presented here regards phonologically distinct bound variants of a base form as separate stems. Actual two-level and some other morphological descriptions apply similar methods in order to cope with morphotactic problems that cannot be treated phonologically in an elegant way.

There are two known important variants of this method: one using technical stems, that is, orthographically motivated strings that linguists do not consider

'real' stem variants (Karttunen 1981), and the other using real allomorphs (Karlsson 1986). Thus, Humor lexicons contain stem allomorphs (which were generated by the learning phase mentioned above) instead of single stems. A morphological parser need not be directly concerned with the derivation of allomorphs from their base forms, for example, it does not matter how *technológiá-* is derived from *technológia-*, for example before accusative *-t.* This phenomenon—a consequence of the orthographical system—is handled by the off-line linguistic process of Humor, which makes the analysis much faster. This method is close to the lexicon compilation used in finite-state models. Relations among allomorphs of the same base form (e.g. *technológia-, technológiá-*) are, however, rather important not only for theoretical linguists, but also for further syntactic processing. According to the above principle, the lexicon of stem allomorphs contain even non-lexical forms, e.g. *bagoly- bagolly-, bagly-, ..., kesztyű-, ..., nyelv-, ..., tart-, ..., technológia-, technológiá-, ...* The lexicon of suffix allomorphs is generated exactly the same way: *-k, -ak, -ek, -ok, -ök, ..., -t, -at, -et, -ot, -öt, ..., -nak, -nek, ..., -val, -vel, ...* Concatenation of stem allomorphs and suffix allomorphs is licensed with the help of the following two factors: continuation classes defined by paradigm descriptions, and classes of surface allomorphs belonging to them. The approach is somewhat similar to the two-level descriptions' continuation classes (Koskenniemi 1983). A simplified description of Hungarian continuation classes are shown below, where % indicates here the starting state, and $ indicates ending (or accepting) states:

```
START:%
        PREFIX          -> STEM_REQUIRED
        STEM1           -> STEM1_PASSED
STEM_REQUIRED:
        STEM1           -> STEM1_PASSED
STEM1_PASSED:$
        STEM2           -> AFFIXES_POSSIBLE
        DERIV_AFF       -> INFL_AFF_POSSIBLE
        INFL_AFF        -> END
AFFIXES_POSSIBLE:$
        DERIV_AFF       -> INFL_AFF_POSSIBLE
        INFL_AFF        -> END
INFL_AFF_POSSIBLE:$
        INFL_AFF        -> END
END:$
```

In fact, there are more concatenation points in the description of a single Hungarian word form for handling eventual prefixes and compounding. Recent implementations of Humor define continuation classes with the help of a so-called *meta-dictionary*, in fact, a finite-state automaton.

Because segmentation of a word-form in Humor is based on surface patterns, typical sequences of suffix morphemes are analyzed as a whole. For example, the Hungarian nominal ending string *-amé* (PersSg1+Poss) consisting of *-am* (PersSg1) and *-é* (Poss) is in fact a complex affix but handled as an atomic symbol

in Humor. The string *amé* is generated from *am+é* separately, in an earlier development phase by a dedicated utility. The generator is able to make a finite set of affix sequences from a description of possible continuation classes. Suffix combinations beginning with the same morpheme are considered equivalent because the only relevant pieces of information come from the suffix that immediately follows the stem. For example, from the point of view of the preceding stem, for example, *ház-*, morpheme combinations like *-am*[PersSg1], *-amé*[PersSg1][Poss], *-amnak* [PersSg1][Dat], *-amének*[PersSg1][Poss][Dat] behave as the suffix *-am* itself (Table 1).

Word-form	**Humor's real-time segmentation**	**Humor's output segmentation**
házam	*ház + am*	*ház + am*
házamé	*ház + amé*	*ház + am + é*
házamnak	*ház + amnak*	*ház + am + nak*
házaménak	*ház + aménak*	*ház + am + é + nak*

Table 1.

Therefore, every affix array is represented by its starting affix. We can say that there is an equivalence relation on the set of affix arrays. Each equivalence class and each paradigm is given an abstract name, that is, each existing set of equivalence classes can have its own abstract name.

Table 2 shows a partial morphotactic paradigm of nominals. Features (morpho-phonological properties) are used to characterize both stem and suffix allomorphs. For instance, the stem *piros-* belongs to the paradigm that can be described by the set of Feature=Value pairs (5).

(5) *piros-* [Cat=Nom, Pos=Adj, Deriv=Abstr, Deg=Comp]
-abb [Cat=Nom, Pos=Adj, Deg=Comp]

Checking 'appropriateness' is based on unification, or, strictly speaking, checking unifiability of the adequate features of stems and suffixes. The word-form *pirosabb* is morphotactically licensed by the unifiability of the two structures: the feature 'Deg' occurs in both with the same value (6)

			Stems				
	Features= +/- Values		**Cat=Nom**				
			Pos=N		**Pos=Adj**		**Pos=Num**
		Morph	*hal-*	*ház-*	*piros-*	*magas-*	*nyolc-*
Affixes	**Nbr=Pl**	*-ak*	+	+	+	+	+
	Deriv=Adj	*-i-*	-	+	-	-	-
	Deriv=Abstr	*-ság-*	-	-	+	+	-
	Deriv=Multipl	*-szor*	-	-	-	-	+

	Deg=Comp	*-abb-*	-	-	+	+	-

Table 2.

(6) INPUT: *pirosabb*
ANALYSIS: *piros*[Adj] + *abb*[Comp]
OUTPUT: piros[Adj][Comp]

The most important advantage of this feature-based method is that possible paradigms and morpho-phonological types need not be defined previously; only the classification criteria have to be clarified. Here you find some criteria used in the morpho-phonological description of Hungarian:

		α = +	α = –
1	α nominal	nominal	verb
2	α frontness	front vowel	back vowel
3	α roundness	rounded vowel	unrounded vowel
4	α ACC	has accusative	does not have accusative
5	α ACCvowel	with vowel *(-Vt)*	without vowel *(-t)*
6	α PL	has plural	does not have plural
7	α PLvowel	with vowel *(-Vk)*	without vowel *(-k)*
8	α lexical	lexical form	non-lexical form
9	α PERS	has personal suffix	does not have personal suffix
10	α DAT	has dative	does not have dative
11	α INS:ß	has instrumentalis	does not have instrumentalis
12	α ÁS	has derivation *–ás/-és*	does not have derivation *-ás/-és*
…	…	…	...

Table 3.

Since the number of these criteria is around a few dozen (in case of a language with rather complicated morphology), the number of theoretically possible paradigm classes is several million or more. According to the general linguistic practice, about 10-20 orthogonal properties are chosen which produce 2^{10}-2^{20} possible classes. In the reality, most of these potential combinations are hypothetical. That is, there are many empty classes. We have got an n-dimensional „morphological periodic table" based on the actual set of binary features describing potential, not-yet-existing forms as well. If we have 12 features, as in the description of Hungarian shown by Table 3, altogether we get 2^{12}=4096 potential vectors, but in the actual Hungarian morphology only 2792 are used out of them. We must not forget that not all the original morpho-phonological features are independent: for example, the vowel V in the Hungarian plural (-Vk) and V in the accusative ending (-Vt) show the following phonological interdependencies: -V_it → -Vk, but –V_ik → {V_it, 0t}. It means that if the accusative form of *ház* is *házat,* then we know that *házak* is its plural. On the other hand, the plural of *kép* is *képek* and plural of *bér* is *bérek*, but the accusative forms are *képet* and *bért,* respectively. Thus, the vowel of the accusative morph cannot be calculated on the

basis of the vowel of the plural morph.

1.2 The process of the morphological analysis

The morphological analysis process is as follows. The continuation matrix provides concatenation licensing based on paradigm descriptions. The columns represent the paradigms themselves, the rows show the suffix category types by which the word to be analyzed can be followed.

The Feature=Value pairs control concatenation licensing based on unifiability checking introduced earlier. In the actual descriptions we use binary features, so 0 or 1 represent '–' or '+' values of the feature represented by the *n*-th column. A small segment of the internal lexical database is shown by Table 5. Allomorphs are shown in the first column, followed by the (right and left) continuation classes and the Feature=Value pairs. The last column shows the part-of-speech of the morph, preceded by the lexical form of the actual allomorph.

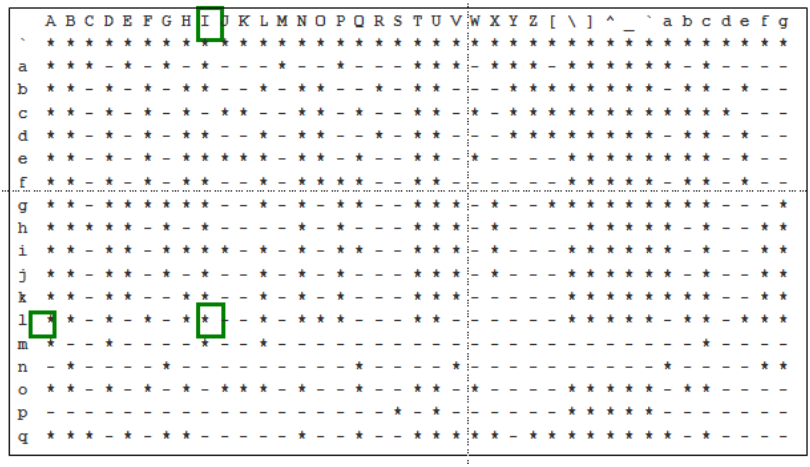


Table 4.

Continuation class information says that *bokrot* can be analyzed as a potential concatenation of stem *bokr-* and terminal suffix *-ot*. The second column of Table 5 says that *bokr-* belongs to paradigm 'H' and type of suffix -ot is 'l'. We see an asterisk in the paradigm matrix at the coincidence point of column 'H' and row 'l' (Table 4), meaning that paradigm 'H' of *bokr-* contains suffix type 'l' of *-ot*. According to this, the algorithm can go on. According to Table 5, *-ot* offers four features to unifiability check: 1st, 2nd, 3rd and 5th columns with values 101 an 1. These four Feature=Value pairs according to Table 3 are the following: Nominal='+', Front='–', Rounded='+' and Has-Accusative='+'. Allomorph *bokr-* fulfills all the requirements, namely Table 5 says that its right feature vector is

10111010 01000010, which has 101 and 1 on the first three and the fifth positions. The unfiability check is positive, so the actual form is licensed by our description. The output is the lexical form of the actual allomorph followed by the part-of-speech of both the stem and the suffix: *bokrot*=bokor[N][ACC].

```
bokor,          G,101..... .0.00010, `,........ ...,....,bokor,          bokor,          N
bokorbab,       B,10111111 11000011, `,........ ...,....,bokorbab,       bokorbab,       N
bokorrózsa,     C,100..... ...00011, `,........ ...,....,bokorrózsa,     bokorrózsa,     N
bokorrózsá,     D,10000100 11100011, `,........ ...,....,bokorrózsá,     bokorrózsa,     N
bokorugró,      A,10100100 11101011, `,........ ...,....,bokorugró,      bokorugró,      ADJ
bokr,           H,10111010 01000010, `,........ ...,....,bokr,           bokor,          N
bokros,         B,10010010 10011010, `,........ ...,....,bokros,         bokros,         ADJ
bokros,         B,10110010 10010010, `,........ ...,....,bokros,         bokros,         N
bokrosod,       A,00011010 10000000, `,........ ...,....,bokrosod,       bokrosodik,     V
bokrosodás,     B,10110010 11000010, `,........ ...,....,bokrosodás,     bokrosodás,     N
bokréta,        C,100..... ...00010, `,........ ...,....,bokréta,        bokréta,        N
bokrétaünnep,B,11011010 11000011, `,........ ...,....,bokrétaünnep,bokrétaünnep,N
bokrétá,        D,10000100 11100010, `,........ ...,....,bokrétá,        bokréta,        N
bokrétás,       B,10010010 10001010, `,........ ...,....,bokrétás,       bokrétás,       ADJ
...
at,             A,00000000 00000000, 1,100.1... ...,....,at,             at,             ACC
et,             A,00000000 00000000, 1,110.1... ...,....,et,             et,             ACC
ot,             A,00000000 00000000, 1,101.1... ...,....,ot,             ot,             ACC
t,              A,00000000 00000000, 1,1...0... ...,....,t,              t,              ACC
öt,             A,00000000 00000000, 1,111.1... ...,....,öt,             öt,             ACC
```

Table 5.

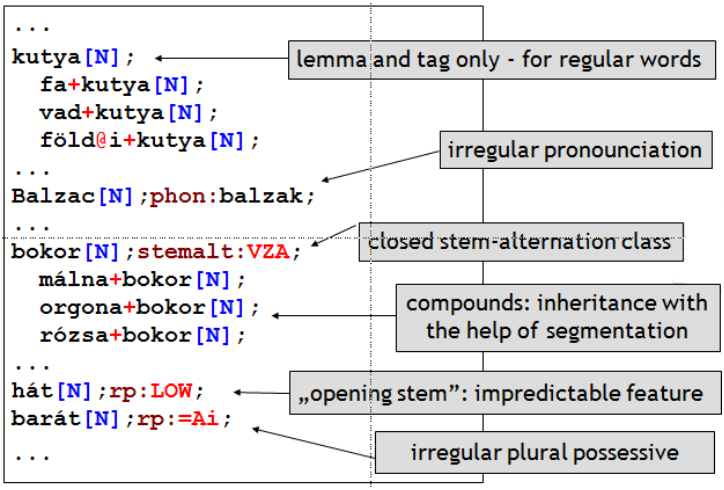


Table 6.

The above tables are rather difficult to produce by hand and there are a lot of potential sources of errors. For this reason, Novák developed an additional system to Humor (Novák & Wenszky 2007) which made it possible to start with basic lexical forms instead of allomorphs, and a special program generates the internal Humor formalism shown by Table 4 and Table 5. A sketch of this formalism (from Novák & Wenszky 2007) is shown by Table 6.

An excerpt from the real Humor lexicon (in human readable format) is shown below:

szó [+nominal +noun +lexical –front –rounded –PL –PERS +ACC –ACCvowel +DAT +INS:V]
[]
szav [+nominal +noun – lexical –front –rounded +PL +PLvowel +PERS –ACC +DAT –INS]
[+nominal +noun + lexical +front –rounded +PL +PLvowel +ACC –ACCvowel +DAT +INS:S]
ak [+nominal –front –rounded +PL +PLvowel]
[+nominal –front –rounded –PL –PERS +ACC +ACCvowel +DAT +INS:K]
ek [+nominal +front –rounded +PL +PLvowel]
[+nominal +front –rounded –PL –PERS +ACC +ACCvowel +DAT +INS:K]
at [+nominal –front –rounded +ACC +ACCvowel]
[]
et [+nominal +front –rounded +ACC +ACCvowel]
[]
nak [+nominal –front +DAT]
[]
nek [+nominal +front +DAT]
[]
val [+nominal –front +INS:V]
[]
vel [+nominal +front +INS:V]
[]
kal [+nominal –front +INS:K]
[]

With the help of the above lexicon we can see how unifiability checking is done. In case of failed constructions (marked by an asterisk before the word form) the critical features that cannot be unified are underlined:

szó [+nominal +noun **+lexical** –front –rounded –PL –PERS +ACC –ACCvowel +DAT +INS:V]

szav* [+nominal +noun **–lexical –front –rounded +PL +PLvowel +PERS –ACC +DAT –INS]

szó+nak [**+nominal** +noun +lexical **–front** –rounded –PL –PERS +ACC –ACCvowel **+DAT** +INS:V]
[**+nominal –front +DAT**]

szó+val [**+nominal** +noun +lexical **–front** –rounded –PL –PERS +ACC –ACCvowel +DAT **+INS:V**]
[**+nominal –front +INS:V**]

szav+val* [+nominal** +noun –lexical **–front** –rounded +PL +PLvowel +PERS –ACC +DAT **–INS**]
[**+nominal –front +INS:V**]

szó+vel* [+nominal** +noun +lexical **–front** –rounded –PL –PERS +ACC –ACCvowel +DAT **+INS:V**]
[**+nominal +front +INS:V**]

szav+ak+at [**+nominal** +noun –lexical **–front –rounded +PL +PLvowel** +PERS –ACC +DAT –INS]
[**+nominal –front –rounded +PL +PLvowel**]
[**+nominal –front –rounded** –PL –PERS **+ACC +ACCvöwel** +DAT +INS:K]
[**+nominal –front –rounded +ACC +ACCvowel**]

szav+ak+kal [**+nominal** +noun –lexical **–front –rounded +PL +PLvowel** +PERS –ACC +DAT –INS]
[**+nominal –front –rounded +PL +PLvowel**]
[**+nominal –front** –rounded –PL –PERS +ACC +ACCvowel +DAT **+INS:K**]
[**+nominal –front +INS:K**]

Some examples for the analysis:

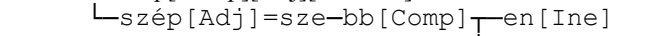
játszunk = játszik[V][Pl1]
└─játszik[V]=játsz─unk[Pl1]

játszottam = játszik[V][PastSg1] |
játszik[V][DefPastSg1] |
játszik[V][PPartic][PsSg1]
└─játszik[V]=játsz┬─ottam[PastSg1]
├─ottam[DefPastSg1]
└─ott[PPartic]─am[PsSg1]

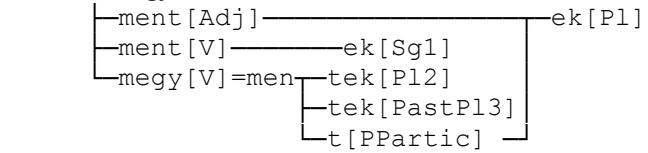
barátokért = barát[N][Pl][Cau]
└─barát[N]─ok[Pl]─ért[Cau]

könyveimet = könyv[N] [PsSg1[Acc]

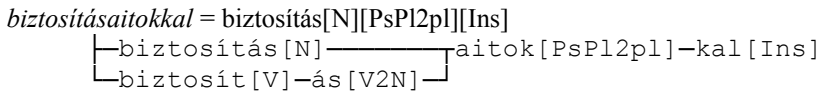
szebben = szép[Comp][Adj][Ine] |
szép[Comp][Adj][Sup]|
szép[Comp][Adj][EssMod]



mentek = ment[Adj][Pl] |
ment[V][Sg1] |
megy[V][Pl2] |
megy[V][PastPl3] |
megy[V][PPartic]



biztosításaitokkal = biztosítás[N][PsPl2pl][Ins]



There are only a few general, reversible morphological systems that are suitable for more than a single language. The most well-known approach is the two-level morphology (Koskenniemi 1983) and its modifications (Karttunen 1993, Beesley & Karttunen 2003). There are some computational morphological description systems showing some features in common with Humor—like paradigmatic morphology (Calder 1989), or the Paradigm Description Language (Anick & Artemieff 1992)—but they don't have large-scale implementations. Two-level morphology is a reversible, orthography-based system that has several advantages from a linguist's point of view. Namely, the morpho-phonemic/graphemic rules can be formalized in a general and very elegant way. It also has computational advantages, but the lexicons contain entries with extra symbols and other sophisticated elements in order to produce the necessary surface forms. Non-linguist users need an easy-to-extend dictionary into which words can be inserted easily. The lexical entries of Humor consist of surface characters only, which makes adding new entries to the system rather easy.

2. Syntax and translation algorithm inspired by Hungarian, a free-phrase-order language

MorphoLogic's English-Hungarian and Hungarian-English machine translation software is based on the MetaMorpho MT engine. This engine was developed by MorphoLogic specifically for the purposes of the machine translation project, and some of its features were inspired by the difficulties inherent in analyzing Hungarian input. In the first part of this chapter we will give the reader an overview of MetaMorpho technology, and in subsequent subsections we will go on to describe how certain phenomena characteristic of the Hungarian language were successfully handled with it.

2.1 An overview of the MetaMorpho technology

MetaMorpho (Novák, Tihanyi & Prószéky 2008) is a rule-based system that uses immediate transfer (Prószéky 2006), which means that any constituent captured by the parser has its translation (or at least one possible translation) ready at the moment of analysis. The analysis itself is based on a monolithic grammar described in a unified formalism called *mmo*, which comprises the whole lexicon, the set of usual context-free rules needed to capture regularities in the source language, and also a number of 'patterns'—some of which may be less linguistically motivated—describing linguistic phenomena that do not easily lend themselves to compositional analysis or translation. The same *mmo* rules that we use to analyze the input contain the target-language equivalent or possible alternative equivalents.

The technology encompasses a parser-generator called Moose, the high-level *mmd* language and a converter to transform it into low-level *mmo* rules, and the Humor morphological analyzer and generator described in the first part of the paper. The capabilities of the MetaMorpho system are best explained in terms of the *mmd* language.

2.2 The syntactic description formalism

The core of MetaMorpho technology is an enhanced context-free grammar, which is represented as a set of *mmd* rules. All rules have a unique identifier, an analysis-side, a generation-side, and possibly a list of *kills*, which will be explained later. The simple *mmd* rule below exemplifies the basic syntax (7).

The analysis-side, which is the second line, immediately below the rule identifier, is very much like an ordinary CF rule. On the left hand side, there is a single non-terminal category symbol, which is followed by an equal sign, which in turn is followed by a list of terminal and/or non-terminal category symbols

separated by plus signs. Obviously, the analysis-side rules are used by the parser to find valid tree representations of the input constituent structure.

(7) *NP=DET+NX:12345678-1
HU.NP[det=YES, def=DET.def] = DET(art=YES) + NX(allowdet=YES)
EN.NP(HU.DET.def=YES) = DET[lex="the"] + NX
EN.NP(HU.DET.def=NO) = DET[lex="a"] + NX
!12345678-2, 12345678-3

Below the analysis-side, there are one or more generation-side rules. These rules also follow the general context-free pattern. They represent the target-language equivalents of the source-language constituent, and are invoked at generation time. TL non-terminals are generated by the same rules that created the corresponding SL node. The choice between multiple generation-side rules in a particular instance is governed by the left hand side conditions.

As is apparent from the example (7), all categories have a set of attributes or *features* as they are usually referred to in our terminology. The list of categories and their features are defined separately in an XML-based format. Attributes fall into three types: *symbolic*, *string*, and *pointer.* Both analysis-side and generation-side rules may check the values of any of these features and may also assign values to them. Application of the rules is subject to meeting the conditions imposed on feature values. Conditions are given in parentheses right next to the category symbols, while value assignments appear inside square brackets.

Symbolic features have a finite set of permissible values, and they are typically used to carry grammatical (syntactic or morphological) information, e.g. number, person, gender, subcategory codes, etc. *String features* typically store the lexical content of most categories. They are predominantly used by lexicon-level rules simply to identify the lexical item. VP patterns or other higher level rules may also refer to string features to impose lexical constraints on their complements. The concept of p*ointer features* is MetaMorpho technology's most valuable addition to the basic context-free attribute grammar underlying it. These features can store entire constituents (together with their analysis). Passing such pointers up to parent elements during analysis and back down during generation allows us to move constituents great distances in translation. Features, as expected, may be assigned constant values, and more importantly, they may inherit the value of another feature that belongs to a child or parent element. In such a way practically any amount of structural and grammatical information can be percolated through an inheritance chain. Checking feature values is not limited to the equality and non-equality operators. A so-called *compatibility operator* may be defined for any pair of symbolic features. Also, the values assigned to symbolic features may be computed by user-defined binary operators.

The last line in (7) contains a list of rule identifiers preceded by an exclamation mark. In MetaMorpho terminology, these are called *kills.* If a rule can be successfully applied to a given range of input tokens, and it contains *kills,* then the

rules so referenced are overridden if they are to be applied to the exact same range. This mechanism allows us to create very simple rules for general cases without having to worry about exceptions, and then list more specific special cases as separate rules. By *killing* the general rules with the more specific ones, the superfluous analyses are suppressed.

Mmd rules are created by our linguists to define the grammar and the lexicon of the MT system, while *mmo* is a slightly less human-readable form that can be directly complied into the binary representation required by the parser. A converter is used to syntax-check and transform high-level *mmd* descriptions into low-level *mmo* rules. There is not much formal difference between *mmd* and *mmo,* in fact as far as syntax is concerned, all *mmo* rules are valid *mmd* rules. The opposite is not necessarily true because the *mmd* language allows some extra features such as shorthand notations for manipulating large groups of features and some *meta-features* that influence the conversion process. Such group identifiers are converted to the whole set of features they represent, while meta-features are stripped from the *mmo* output when they are no longer needed.

The reason why we decided to have the higher-level *mmd* description on top of *mmo* is efficiency. Especially in the case of lexicon-level rules, but also in many other cases, a rule has a lot of predictable content while the amount of unique, non-predictable information is very little in comparison. In an *mmd* rule, only the non-predictable part must be present, the rest is supplied by the converter, therefore an immense amount of typing can be spared. The converter is not restricted to adding missing feature checks and assignments to a rule. It may even generate a large number of related rules based on a single *mmd* description. For example the converter is responsible for creating the active and passive forms of an English VP from a single common *mmd* definition. As can be seen from this, some linguistic knowledge may be encoded in the converter itself. While it may deduct from the purity of the MetaMorpho system that not all the regularities of language are captured in the formal grammar, the simplicity of performing certain transformations with the aid of a computer program rather than the limited capabilities a CF transducer, and the amount of redundancy that can be avoided this way outweigh other considerations.

2.3 Parsing and translating Hungarian with MetaMorpho

Hungarian syntax has a number of phenomena which are difficult to handle with the usual tools based on context-free grammars, which are best suited to configurational languages such as English. In the following chapters we will discuss how the special features of MetaMorpho technology enabled us to tackle problems like free phrase order within the sentence, long distance dependencies, the prevalence of zero constituents, and the translation of nominal predicates.

While lexical phrases in Hungarian have a well-defined word order and are easily handled by well-established methods, sentence structure is much more

complicated. Grammatical functions such as subject, object, etc. and argument structure are not identified by structural position but solely by case endings and/or post-positions. There do seem to be some fixed positions inside a clause, but these have to do with logical, semantic and pragmatic rather than syntactic functions. The basic Hungarian finite clause follows the pattern: Topic—Focus—Predicate. The topic is the pragmatically most salient element in the predication, which can be any constituent, even a free adverbial. The focus position is obligatorily occupied by constituents with negative polarity or 'restrictive' meaning, and may also express contrastive meaning, similarly to the structure that traditional grammars call the 'cleft sentence' in English. The verb, or an auxiliary, when one is present, is always at the left edge of the predicate, but the rest of the verb's complements and any free modifiers may follow in just about any order. In summary, without going into now irrelevant details, it may be said that for the purposes of MT, Hungarian clauses can be regarded as having absolutely free phrase order.

The random order of sentence constituents is not the only problem we have to face. In surface structure we can also have a number of different disjointed constituents. It is not unusual for the possessor in possessive NP constructions to leave the NP and move into topic position. So the following two sentences are nearly identical in meaning (8).

(8) *Ellopták Mari pénztárcáját*.
Ellopták $_{NP}$[Mari pénztárcáját].
steal[PAST][PL3] Mary[NOM] wallet[ACC][possSG3]
"They stole Mary's wallet."
Marinak ellopták a pénztárcáját.
[Marinak] ellopták $_{NP}$[a pénztárcáját].
Mary[GEN/DAT] steal[PAST][PL3] the wallet[ACC][possSG3]
"Mary had her wallet stolen."

Complement clauses also have a tendency to be extraposed to the right edge of the sentence, leaving a pronominal antecedent behind. Finally, relative clauses may also move to the right edge, while the phrases they modify remain in their original position. As a result, a single logical constituent may show up on the surface in two or even three separate pieces, which for the purposes of argument identification and translation have to be matched to each other.

The number of surface constituents may also be less than expected. Hungarian subjects and objects can be realized as phonologically zero elements. Since the verb is inflected for subject and object agreement, the pronominal subject and object can be recovered from the verb ending (9). Depending on the analysis one chooses, it may be argued that sometimes even other complements of the verb may be zero constituents. In this case, an incorporated adverbial particle, usually referred to as the *verb modifier*, allows us to infer the presence and the grammatical case of the covert constituent. Zero pronouns in Hungarian are

illustrated below by (9) and (10).

(9) *Szeretlek.*
love[PRES][SG3][objSG2]
"I love you."

(10) *A könyvet beleraktam a táskába.*
The book[ACC] into-put[PAST][SG1] the bag[INE]
"I put the book in the bag."
Beleraktam a könyvet.
into-put[PAST][SG1] the book[ACC]
"I put the book in it."

Pronouns are not the only syntactic category that may be realized as phonologically zero elements. In sentences with a nominal predicate, the copula may or may not be overt depending on tense and mood. Compare for example the present and past tense of *The grass is green* in Hungarian (11).

(11) *A fű zöld.*
the grass[NOM] green
"The grass is green."
A fű zöld volt.
The grass[NOM] green be[PAST][SG3]
"The grass was green."

In order to translate verb phrases adequately, one has to identify the complete argument structure, the verb and its complements cannot be translated in isolation. Therefore, once the individual clause constituents have been parsed, somehow they have to be put together as the right lexical item. We call the *mmd* rules that describe Hungarian verbal phrases and the corresponding English translations *VP patterns*. Here is an actual example (12).

(12) *VP=találkozik:15577
HU.VP = SUBJ + TV(:lex="találkozik") + COMPL#1(pos=N, case=INS, abstract=NO)
EN.VP = SUBJ + TV[lex="meet"] + COMPL#1

The fundamental problem is how to make such a VP pattern match an almost random sequence of phrases in a scenario where the arguments of the verb may appear in any order, sometimes as disjointed constituents, sometimes they do not appear at all, even the head of the VP may be missing in some special constructions, and there may be any number of sentence-level adverbial modifiers interspersed.

The simple straightforward solution to identifying *VP patterns* would be to generate all possible configurations from the *mmd* rule in (6) using the *mmd* to *mmo* converter. Unfortunately, however, the enormous number of mmo rules thus

obtained would not be manageable. There are tens of thousands of VP patterns in our lexicon. Given the number of possible permutations of sentence constituents combined with the possibility of one or more of them being zero, and also counting some other quirks of Hungarian grammar that we have not even mentioned, we may well end up having several hundred and maybe more rules for each VP pattern. The resulting grammar file would take up gigabytes at least and the parsing process despite all optimizations would slow down drastically.

Another approach could be combining the parser with a software module that can somehow access the VP pattern descriptions and the pool of existing partial analyses, and using a set of algorithms designed for the purpose, create the correct VP nodes and place them back in the analysis space. This solution is not impracticable, but taking full advantage of the tools provided by the MetaMorpho framework, we came up with a solution that does not require external tinkering with the parsing process and can be implemented with the *mmd* language alone. The basic idea is that a CF parser itself can be used as a simple computer, and the grammar can serve as a program that can be executed on it. We obviously do not have the space here to develop this theory, but suffice it to say that with user-defined operators it is possible to define an arithmetic over symbolic features used as variables, and a set of recursive rules and some features functioning as status indicators or flags for exit conditions can serve as execution loops. Such techniques enabled us to implement all necessary algorithms within the grammar itself.

The main goal was to find a way to identify VP patterns using a single *mmo* rule only. To do so, we used the mmd-to-mmo converter to create a 'canonical form' of the pattern and made sure that regardless of the accidents of surface form, the sentence constituents appear to the *mmo* VP rule to be in the same 'canonical form'. Let us briefly show how we achieved that.

We introduced a purely technical category named VPP and a set of core grammar rules that operate on it recursively. Starting from the verb and then going first right and then left, these rules gather all the constituents one by one and store them in an ordered list of pointer features. During this process the constituents which are potential arguments of the verb are sorted according to an arbitrarily determined ordering based on their grammatical case and some other syntactic features. This ordering is part of the 'canonical form', which makes permuting the constituents unnecessary, thereby solving the problem of free phrase order. It should be noted that generating all permutations of the arguments is also possible using such 'grammar-programs', but less efficient. Disjointed constituents are 'reunited' during this process by placing them in associated pointer features after checking for compatibility. At generation time the constituent that stores the head of a disjointed phrase is passed the extraposed parts via these associated pointers.

In order for the checking of arguments to be a uniform process, zero constituents must be supplied. A simple CF grammar is incapable of handling an element that is simply not there, but using a trick with pointers we can overcome this difficulty as well. The idea is to use the entire clause to represent the zero element whose existence can be inferred from it. We build an ARG (the category that represents

an argument) on top of the VPP, and store the VPP in a pointer. Then we build a VPP on top of the ARG, restore its original attributes form the pointer, and store the faked ARG in the appropriate argument pointer. The end result is a VPP which contains the zero argument in exactly the same manner as any other overt constituent, namely a pointer. The concept is illustrated by the following pair of simplified rules.

(13) HU.ARG[zero=YES, Vpp<-VPP] = VPP(zeroarg=NO)
EN.ARG = PRON
HU.VPP[Arg<-ARG, zeroarg=YES, :features=ARG.Vpp->:features] = ARG[zero=YES]
EN.VPP = VPP{HU.ARG.Vpp}

During the 'VPP stage' we may also perform transformations to adjust for passive, factitive or participial constructions, so that all these may finally be represented as the 'canonical form'.

We still haven't accounted for how we differentiate arguments from free modifiers. Case marked NPs and phrases containing post-positions may generally both serve either as complements of the verb or as free adverbials. We could of course have alternative analyses for these constituents, but the number of possible combinations of arguments and modifiers may result in unnecessarily high memory requirements during parsing. So taking advantage of multiple generation-side lines in an *mmd* rule, we create *polymorphic constituents*. Instead of determining the role of such NPs before VP pattern identification, all of them have an ADVP layer above the NP. ARGs are then created from such ADVPs with multiple generation-side rules. Depending on a left hand side condition, these rules translate the phrase either as an adverbial or as a simple noun phrase.

VP patterns do not check the arguments directly. Instead, the constraints are copied into a set of features, and once again technical rules take over and check the arguments one by one based on the contents of these features. Whenever these rules encounter a constituent that turns out not to be an argument, the rest of the potential arguments are shifted while the current constituent is assigned modifier status. At generation time the constituent is translated accordingly.

Although giving a full account of how we handled the more difficult aspects of Hungarian sentence structure would have been far beyond the scope of the present paper, we hope that we have managed to give at least an interesting insight into our work.

3. Conclusion

The first part of the paper has described a morphological system inspired by an agglutinative language, Hungarian. The solution belongs to the ‘item-and-arrangement” paradigm. The method itself uses lexicons of allomorphs and

adjacency restrictions between them. A feature system and a unification algorithm —or better said: checking a relation called unifiability—is also used. Concatenation points between morphs are defined by continuation classes, and the basic idea behind the algorithmic description is—instead of a real-time analysis of every morpho-phonological attribute—that there are no more active operations in this computational morphology. Due to its minimal memory requirements even the first versions of the system could effectively run even on early PC's (since 1991).

In the second part of the paper we have dealt with the syntactic processing of Hungarian texts and the way it is used in the Hungarian–English MetaMorpho machine translation system. First the sentence is segmented into terminal symbols, and the morphological analyzer determines all the needed morpho-syntactic attributes of these symbols. The syntactic parser analyzes the input sequence and if it is recognized as a correct sentence, comes up with one or more root symbols on the source side. When the whole input is processed and no applicable patterns remain, the target equivalent is read top-down from the root symbols by firing the target pattern corresponding to the source pattern that created the edge at parse time. This solution—a sort of "immediate transfer"—uses no separate transfer steps or target transformations. In the remaining part of the chapter we show how certain linguistic phenomena of Hungarian are described with the help of this formalism.

Implementations of both algorithms run in machine translation, proofing tools, intelligent search and comprehension assistance applications.

4. References

Abondolo, D. M. 1988, *Hungarian Inflectional Morphology*. Budapest: Akadémiai

Anick, P. & Artemieff, S. 1992, A High-level Morphological Description Language Exploiting Inflectional Paradigms. *Proceedings of COLING-92*, Nantes, France, 67–73

Beesley, K. R., Karttunen, L. 2003, *Finite State Morphology*. CSLI Studies in Computational Linguistics. Stanford:CSLI

Bryant, B.D., Miikkulainen, R. 2001, From Word Stream to Gestalt: A Direct Semantic Parse for Complex Sentences. *AI Technical Report* TR-AI98-274, Austin:University of Texas

Bybee, J. L. 1985, *Morphology. A Study of the Relation Between Meaning and Form.* Amsterdam: John Benjamins

Calder, J. 1989, Paradigmatic Morphology. *Proceedings of 4th Conference of EACL 89*. Manchester, United Kingdom, 58-65

Karlsson, F. 1986, A Paradigm-based Morphological Analyzer. *Papers from the Fifth Scandinavian Conference of Computational Linguistics*, Helsinki:University of Helsinki, 95-112

Karttunen, L., Root, R. and Uszkoreit, H. 1981, Morphological Analysis of Finnish by Computer. *Proceedings of the 71st Annual Meeting of the SASS.* Albuquerque, New Mexico.

Karttunen, L.1993, Finite-State Lexicon Compiler. *Technical Report*. ISTL-NLTT-1993-04-02. Palo Alto: Xerox PARC

Koskenniemi, K. 1983, *Two-level Morphology: A General Computational Model for Word-form Recognition and Production*. Publications No.11, Helsinki:University of Helsinki

Novák, A. 2008, Language Resources for Uralic Minority Languages. In: *Proceedings of the SALTMIL Workshop "Collaboration: Interoperability Between People in the Creation of Language Resources for Less-resourced Languages"*. Marrakech, 27–32

Novák, A., Wenszky N. 2007, Mire jó és hogyan készül egy számítógépes morfológia. In: Alberti G., Fóris Á. (eds.): *A mai magyar formális nyelvtudomány műhelyei.* Budapest:Nemzeti Tankönyvkiadó, 157–169

Novák, A., Tihanyi L., Prószéky, G. 2008, The MetaMorpho Translation System. In: *Proceedings of the Third Workshop on Statistical Machine Translation.* Columbus, 111–114

Prószéky, G. 2006, Translating While Parsing. In: Suominen, M. et al (eds.) *A Man of Measure (Festschrift in Honour of Fred Karlsson on his 60th Birthday).* Turku:The Linguistic Association of Finland, 449-459

Prószéky, G., Földes A. 2005, Between Understanding and Translating: A Context-Sensitive Comprehension Tool. *Archives of Control Sciences* 15(4), 637-644

Prószéky, G. Kis B. 1999, A Unification-based Approach to Morpho-syntactic Parsing of Agglutinative and Other (Highly) Inflectional Languages. *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics.* College Park, 261-268

Prószéky, G., Kis, B., Pál, M. 2004, Specialties and Applications of Hungarian Proofing Tools. *Pre-Proceedings of the 1st International Workshop on Proofing Tools and Language Technologies*. Patras:Conference and Cultural Centre, 113–116

Tompa, J. (ed.) *1962, A mai magyar nyelv rendszere*. Budapest: Akadémiai

Winograd, T. 1983, *Language as a Cognitive Process*. Reading:Addison-Wesley

Wołosz, R. 2005, *Efektywna metoda analizy i syntezy morfologicznej w jezyku polskim.* Warszawa:Akademicka Oficyna Wydawnicza EXIT

Appendix: Glosses for Hungarian words used in the examples

Hungarian	POS	English
barát	N	friend
bagoly	N	owl

bér	N	wages/pay
biztosít	V	assure/insure
biztosítás	N	insurance
bokor	N	bush
hal	N	fish
hát	N	back
ház	N	house
játszik	V	play
kép	N	picture
kesztyű	N	glove
kesztyűtartó	N	glove box
könyv	N	book
kutya	N	dog
magas	ADJ	high/tall
megy	V	go
ment	ADJ	safe/free from sth
ment	V	save
nyelv	N	language
nyolc	NUM	eight
piros	ADJ	red
szép	ADJ	beautiful
szó, szavak	N	word, words
tart	V	hold
tartó	N	holder/container
technológia	N	technology