

KURD - A Formalism for Shallow Post Morphological Processing

Michael Carl, Antje Schmidt-Wigger and Munpyo Hong

email: carl,antje,munpyo@iai.uni-sb.de

Institut für angewandte Informationsforschung,
Martin-Luther-Straße 14, 66111 Saarbrücken
Germany

1 Introduction

In most NLP applications an input text undergoes a number of transformations until the desired information can be extracted from it. Typically, such transformations involve part of speech tagging, morphological analysis such as lemmatization or full derivational and compositional analysis, context dependent disambiguation of tagging results, multi-word recognition, shallow, partial or full syntactic parsing, semantic analysis and so on. It is not always evident what level of analysis should be involved. For instance, whether a certain task requires a full parse or whether some 'shallow' operations may be sufficient is often difficult to determine. The choice of the involved tools can be guided by the data or the requirements or premises of the goal to be reached. These considerations may depend on the availability of a grammatical model, the required standard of the results and processing time constraints. However, the optimization of this task remains an unresolved area until now.

The interest of the NLP community for 'shallow' processing has grown recently (cf. (Skut et al., 1997),(Abney, 1996)). In this paper, we describe a simple formalism (KURD) that is designed to perform some 'shallow' operations on morphologically analyzed texts. The output can be used directly, or be redirected to further (e.g. linguistic or statistic) processing.

Typical tasks for such shallow processing include

- **Tagging** (disambiguation of multiple morphological analysis) Often a set of simple rules that runs in a set order over the results of the morphological analysis is sufficient to disambiguate multiple analysis of a word due to its morphosyntactic context.
- **Syntax checking**
Grammatically erroneous sentences are detected by a set of rules describing common weak points such as missing punctuation marks or ill-formed agreement.

These rules sometimes have to describe full phrases, but in most cases checking a few analyzed words suffices to define the error.

- **Style checking**
Highly complex constructions or heavy phrases can disturb the reading and understanding process. To avoid this, style checkers can recognize such patterns so that the author can readjust his text for better communication.
- **Shallow parsing**
Shallow parsing can help to simplify the data before full parsing is undertaken. It recognizes syntactic phrases, mostly on the nominal level, and merges them together into one node which is then presented to the next processing level.
- **Segmentation**
The morphological analysis deals with words which are presented in texts. High level processing deals with units between the word level and the text level, mostly with sentences. Thus, sentence segmentation is a typical shallow process, but other subunits could be equally interesting.

The basic idea of the presented formalism is the following: in a set of rules, patterns are defined which are mapped onto the morphologically analyzed input strings. If the mapping is successful, modifications of the analysis are undertaken according to the specifications in the rule. To ensure expressiveness and ease of formulation of the rules, we have introduced some elements of unification based systems into the formalism.

2 Morphological Analysis

Morphological analysis¹ is the process of separating grammatical information and (a) stem(s) from the surface form of an input word. Lemmatization

¹In this section and in the paper we refer to MPRO as the analysis tool (Maas, 1996). MPRO is very powerful: it yields approx. 95% correct morphological analysis and lemmas of arbitrary German and English text.

generates from an input string a basic word form that does not contain inflectional information. A lemma together with the grammatical information is thus equivalent to the surface form of the word. In addition, morphological decomposition can be carried out by the morphological processor. Recognition of composition and derivation yields knowledge about the internal structure of the word.

Morphological information and the value of the lemma are represented in the form of sets of attribute/operator/values ($A \text{ op } V$) which we will refer to as feature bundles (FB s). Beside morphological analysis and lemmatization, sentence segmentation is performed by the morphological processor. The output is thus a sentence descriptor SD that contains multiple Word Descriptors WD . The distinction between WD s and deeper embedded FB s is useful later in this paper due to the important functional difference. The formal definition of a SD is as follows:

Sentence Descriptor SD :

```

SD ::= WD , ... , WD .
WD ::= FB
FB ::= { AVS } ; ... ; { AVS }
AVS ::= A op V , ... , A op V
V ::= ATM | FB | VAR
ATM ::= atom ; ... ; atom
VAR ::= ' _ ' followed by any string
A ::= any alpha numeric string
op ::= = | ^=

```

A WD may consist of two types of disjunctive representation (local or complex disjunction) in a number of different levels. Local disjunction is an alternation of atomic values, complex disjunction is an alternation of complex features (FB). Which of the disjunctive representations is chosen depends on one hand on the expressive requirements (i.e. no feature dependencies can be expressed with local disjunctions) and on the other hand on the linguistic presumptions of the morphological analysis.

Word descriptor “der”:

$$\left\{ \begin{array}{l} \text{lu=d_art, c=w, sc=art, fu=def} \\ \text{agr} = \left\{ \begin{array}{l} \text{gen=f,} \\ \text{nb=sg,} \\ \text{case=d;g} \end{array} \right\}; \left\{ \begin{array}{l} \text{gen=m,} \\ \text{nb=sg,} \\ \text{case=n} \end{array} \right\}; \left\{ \begin{array}{l} \text{nb=plu,} \\ \text{case=g} \end{array} \right\} \end{array} \right\};$$

$$\left\{ \begin{array}{l} \text{lu=d_rel, c=w, sc=rel, fu=np,} \\ \text{agr} = \left\{ \begin{array}{l} \text{case=n,} \\ \text{g=m,} \\ \text{nb=sg} \end{array} \right\}; \left\{ \begin{array}{l} \text{case=g;d,} \\ \text{nb=sg,} \\ \text{g=f} \end{array} \right\} \end{array} \right\}$$

Both types of disjunction are shown in the the German article “der”. A first level of disjunction occurs on the level of the word descriptors. Different analyses (as a determiner (lu=d_art) and

as a relative pronoun (lu=d_rel)) are separated by a semicolon ‘;’. The second level of disjunction occurs in the feature “agr” that has a complex disjunction as its value. The feature “case” in the first complex disjunctive has a local disjunction (g;d) as its value. The word “der” has seven different interpretations which are melted together here by means of the two different types of disjunction.

Note that each attribute may only occur once in a FB . We presume that each attribute in a (morphological) FB expresses a different piece of information (it thus has a different type). As a consequence, we do not need variable binding between different attributes².

3 Feature declaration

Consistency of knowledge resources is of crucial importance in any computer system. In large applications and huge data bases facilities are required which efficiently control consistency to ensure complete recovery. Data typing is one way of doing so. Data typing, in addition, allows for efficient encodings.

We use feature declarations to describe the structure of the data and distinguish between two primitive classes of features and one complex class. A feature with (countable) infinitely many values is declared by the sign “?”.

Features with finitely many values must explicitly enumerate them in the feature declaration³. Complex features are defined recursively. Further, we allow mixed features which can have either complex or finite values. Features of class ‘infinite’ cannot be mixed.

Feature declaration

```

#ENTRY : {lu=#LU, agr=#AGR}
#LU : ?
#AGR : (nil; {gen=#GEN,nb=#NB,case=#CASE})
#GEN : (f; m; n)
#NB : (sg; pl)
#CASE : (n; g; d; a)

```

In the example, “#ENTRY” is a complex type (FB) that consists of two features: “lu” and “agr”. The feature “lu” is of type “#LU”. Since it can have any string as its value “#LU” is defined as a member of the class ‘infinite’. The feature “agr” is of mixed type: it can be complex or finite. As

²In many theories and formalisms (e.g. HPSG, CAT2 (Sharp and Streiter, 1995)) different attributes in a FB can be forced to always have the same values by assigning the same variable as their values (they share the same structure). However, these approaches allow structure sharing and variable binding only among the same types.

³Due to common computer hardware we restrict the number of different values to 32. This allows to encode each value as one bit; unification of two power-sets thus only requires one machine instruction.

a finite feature, “agr” may only take “nil” as an atomic value. As a complex feature “agr” consists of the finite features “g”, “nb” and “case”.

4 The Formalism

The formalism we shall describe in this section applies a set of rules in a predefined order to sentence descriptors *SD* thereby modifying selected word descriptors *WD*. The modified *SDs* are returned as a result. For each *SD*, each rule is repeatedly applied, starting from the first *WD*.

A rule essentially consists of a *description* part and an *action* part. The *description* consists of a number of *conditions* that must match successive *WD*. While matching the *description* part of a rule onto a *SD*, *WD* are marked in order to be modified in the *action* part. A rule fails if a *condition* does not match. In this case the *action* part of the rule is not activated. The *action* part is activated if all *conditions* are satisfied. *actions* may modify (change, delete, add) a *WD* or single features of it.

A *condition* of a rule can either match an *interval* or it can match a *count* of the *WD*. In the former case, one set of *tests* must be true. In the latter case two sets of *tests* must be true, one for an external *interval* and for counting an internal *interval*.

4.1 Some examples

German verbs have detachable prefixes that can be homonym with prepositions. Morphological analysis thus generates two interpretations for some detachable prefixes. However, the syntactic position of prefixes and prepositions within a sentence is different. While prepositions occur as the head in prepositional phrases and thus are always followed by a nominal phrase or a pronoun, detached prefixes occur at the end of the matrix sentence, thus followed by a punctuation mark or a coordinator. The following rule disambiguates a prefix at the end of a sentence, i.e. the interpretation as a preposition ($c=w, sc=p$) shall be deleted from the *WD*.

$$(1) \text{ Disambiguate_Prefix} = \\ \text{Ae} \left\{ c=w, sc=p \right\} \text{e} \left\{ c=vpref \right\}, \\ \text{a} \left\{ c=w, sc=punct; comma \right\} : \\ \text{Au} \left\{ c=vpref \right\}$$

The rule 1 consists of two *conditions* (separated by a comma) in the *description* part and one *act* in the *action* part. It illustrates the capacity of the formalism to express disjunction and conjunction at the same time. The first *condition* matches a preposition ($c=w, sc=p$) and a prefix ($c=vpref$). That is, the matched *WD* must be

ambiguous with respect to its category. Feature cooccurrences are required in the first *test*, where both features $c=w, sc=p$ must occur in a conjunction of the matched *WD*. The existence quantifier e preceding the *FB* means that there is an appropriate interpretation in the *WD*, i.e. there is a non-empty intersection of *FB* and *WD*. The second *condition* matches a end-of-sentence item ($sc=punct; comma$). Here, the all quantifier a requires the *WD* to be a subset of the *FB* i.e. there is no interpretation in the *WD* that is not mentioned in the *FB*.

A *WD* for which the first *condition* is true is marked by the marker “A”. The rule applies if both *conditions* are true. The *action* part has one *consequence* that consists of one *act*. The *WDs* which have been marked in the *description* part are unified with the *FB* ($\{c=vpref\}$) of the *act*. This results in the unambiguous identification of the prefix because the prepositional analysis is ruled out.

An example of a rule that disambiguates the agreement of a (German) noun phrase is given below (2). The rule can be paraphrased as follows: for all sequences of *WD* that have a unifyable agreement feature ($_AGR$) and that consist of an article ($c=w, sc=art$) followed by zero or more adjectives ($*c=adj$) followed by one noun ($c=noun$): unify the intersection of the agreement ($_AGR$) into the respective features of the marked word descriptors.

$$(2) \text{ Disambiguate_Noun_Phrase} = \\ \text{Ae} \left\{ c=w, sc=art, agr=_AGR \right\}, \\ * \text{Aa} \left\{ c=adj, agr=_AGR \right\}, \\ \text{Ae} \left\{ c=noun, agr=_AGR \right\} : \\ \text{Au} \left\{ agr=_AGR \right\}$$

The *description* part of rule (2) has three *conditions*. Each *condition* matches an *interval* of the *WDs*. The second *condition* can possibly be empty since it has the kleene star *scope* (“*”). All *WDs* for which the *test* is true are marked by the marker “A” and thus undergo the same *act* in the *action* part.

The formalism allows the use of variables (e.g. $_AGR$) for the purpose of unification. *WDs* can only be modified by instantiations of variables i.e. variable bindings may not be transferred into the *WD*. Each time a rule is activated, the variables are reinitialized.

The rule (2) matches a noun phrase thereby disambiguating the agreement. With slight changes, the output of the rule can be turned into a shallow parse:

$$(3) \text{ Parse_Noun_Phrase} = \\ \text{Ae} \left\{ c=w, sc=art, agr=_AGR \right\},$$

*Aa {c=adj, agr=_AGR},
 +Be {c=noun, agr=_AGR} :
 Ak{ }, Br {c=np, agr=_AGR}

The operator “r” in rule (3) replaces the category value in the noun node by a new one (c=np). The determiner node (c=w, sc=art) and all adjective nodes (c=adj) are removed (‘killed’) (Ak{ }) from the sentence descriptor such that only the NP node is printed as a result.

Style checking has often to deal with the complexity⁴ of a phrase. Therefore, it makes use of another type of rules where the presence of a number of word interpretations in a certain count is checked. For instance, in technical texts it may be advisable not to have more than eight words before the finite verb. The rule (4) unifies an appropriate warning number into the first finite verb analysis if more than eight words have occurred before it.

(4) Verb_Position =
 e {wnrr=1, vtyp~=fiv},
 8e {sc~=comma;cit;slash} |
 a {vtyp~=fiv}{c~=verb},
 Aa {c=verb, vtyp=fiv} :
 Au {warning='405'}

The first condition matches the first WD in a sentence (wnrr=1) if it has an interpretation different from a finite verb (vtyp~=fiv). The second condition is a count that matches a sequence of WDs other than finite verbs. This is expressed by the external test following the vertical bar. The internal test (the part before the vertical bar) counts the number of words in the count different from punctuation marks and slashes (sc~=comma;cit;slash). The count is true if eight or more such internal tests are true. The motivation for the third condition is to put the marker “A” on the finite verb such that it can be unified with the warning in the action part. The warning can be used by further tools to select an appropriate message for the user.

4.2 Formal Definition

The formal definition of rule syntax is given below:

⁴The complexity of a phrase is a function of different parameters such as its length, the number of lexical elements, the complexity of its structure. The definitions differ from one author to the next. In our calculation of complexity, only length and number of lexical elements are taken into account.

Definition of rule:

rule ::= name '=' descr ':' action
 descr ::= condition₁ ',' condition₂ ...
 condition ::= interval | count
 interval ::= [scope][marker] test₁ test₂ ...
 count ::= num interval_{int} '|' interval_{ext}
 test ::= quantifier FB

action ::= conseq₁ ',' conseq₂ ...
 conseq ::= marker act₁ act₂ ...
 act ::= operator FB

scope ::= ^ | + | * | -
 marker ::= A | ... | Z
 num ::= 0 | ... | 99
 operator ::= k | u | r | d
 quantifier ::= e | a

Whether a rule applies (i.e. its action part is executed) depends on whether its conditions could have successfully been matched. In order to successively match conditions onto a sentence descriptor SD, a word description pointer (WDP) is set to an initial position in the SD⁵. We do not foresee backtracking or multiple solution generation. Thus, each condition matches the longest possible sequence of WDs and, once matched, other segmentations are not considered.

In case a condition is of type interval, the WDP is increased dependent on the outcome of the test. If all tests match the word descriptor, the WDP is increased by one. The length of the interval (number of times the WDP is increased in a condition) depends on the scope of the interval and the outcome of the tests. In accordance with many linguistic formalisms we distinguish between four scopes.

- ^ The interval matches one optional word.
The interval always is TRUE.
The WDP is increased by one if all tests return true.
- * The interval matches zero or more words. (Kleene star)
The interval always is TRUE.
The WDP is increased by one until at least one test returns false.
- + The interval matches at least one word.
The interval is TRUE if all tests are true for the first WD.
The WDP is increased by one as long as all tests are true.
- The interval matches one and only one word.
This is the default value for the scope.
The interval is TRUE if all tests are true.
The WDP is increased by one if all tests return true.

⁵This is normally the word number. It is then successively increased by one until the end of the sentence each time the rule is re-applied.

A *test* maps a *FB* onto a *WD*. Whether a *test* is true or false depends on the *quantifier* of the *test*. The *existence quantifier* “e” and the *all quantifier* “a” are implemented as follows:

- e The *test* is true if there is a non-empty subset between the *FB* and the current *WD*. The *FB* describes a possible interpretation of the current *WD*. The *test* is true if there is at least one interpretation in the current *WD* that is unifiable with *FB*.
- a The *test* is true if the current *WD* is a subset of the *FB*. The *FB* describes the necessary interpretation of the current *WD*. The *test* is true if the *FB* subsumes all interpretations of the current *WD*.

All *conseq* of the *action* part are executed if the *descr* part matches. The *acts* of a *conseq* apply to the marked *WD*. The following operators are currently implemented:

- k kills the marked *WD*.
- u unifies *FB* into the marked *WD*.
- r replaces the values of the features in the marked *WDs* by those of *FB*.
- d deletes the specified features in *FB* from the marked *WD*.

Apart from an *interval*, a *condition* can consist of a *count*. The length of a *count* is controlled by a set of *external tests* (*interval_{ext}*), i.e. the right border of the *count* is either the end of the phrase or a *WD* where one of the external tests is *FALSE*. The outcome of a *count* (whether it is *TRUE* or *FLASE*) is controlled by a set of internal tests (*interval_{int}*). For a *count* to be true, at least the specified *number* of internal tests must be true.

5 An Application

In this section we want to describe an application of the formalism for style checking of technical documents. The application has been developed and tested for a car manufacturing environment (Haller, 1996).

In technical documentation, the quality of the text in terms of completeness, correctness, consistency, readability and user-friendliness is a central goal (Fottner-Top, 1996). Therefore completed documents undergo a cycle of correction and re-editing. As our experiments in this production process have shown, 40% of re-editions in technical documents are motivated by stylistic considerations (compared to corrections of orthographies, syntax, content or layout).

On the basis of the observed re-editions, stylistics guidelines have been formulated, like:

1. Do not use compounds made up of three or more elements.
2. Do not use the passive voice, if the agent is expressed.
3. Long coordinations should be represented in lists.

The compilation of these guidelines has influenced to some degree the architecture of KURD. Most scientists correlate the readability of a sentence with its complexity, defined often by length, number of content words and/or structural embedding. Whereas such information is not common in NLP applications, its calculation can be modeled in KURD through the *count* mechanism.

The basic idea of the use of the formalism for style checking is exemplified by rule (4): a morphosyntactic pattern is recognized by a specific rule unifying a warning number into the marked *WD*. This number triggers an appropriate message in further processing steps that signals the use of an undesirable formulation. As a result, the user can ameliorate that part of the text.

For better results, the style checking application makes use of the disambiguating power of KURD; i.e. some tagging rules (e.g. rule (1)) precede the application of the style rules.

The system contains at its present stage a number of 36 style warnings which are expressed by 124 KURD rules, an average of 3 to 4 rules for each style problem. The warnings can be classified as follows (for examples, see above):

1. **One word warnings** (10 types of warnings): These warnings can either recognize the complex internal structure of compound words, or forbid the use of a certain word. In this last task, style checking moves towards checking against the lexicon of a Controlled Language. This task should not be over-used, a lexically driven control mechanism seems to be more adequate.
2. **Structure-linked warnings** (19 types of warnings): These warnings react to complex syntactic structures and trigger the proposal of a reformulation to the writer. They are therefore the most interesting for the user and for the rule writer.
3. **Counting warnings** (7 types of warnings): These warnings measure the complexity of a sentence or of a sub-phrase by counting its elements. Complexity is a central topic in the readability literature, but it does not allow to trigger a concrete reformulation proposal to the user.

Most structure-linked warnings require more than one KURD rule. This is due to the fact that the pattern to be recognized can occur in different

forms in the text. As shown by the following example (5), two rules would be necessary to detect the 'Future II' in German, because word order of verbal phrases in main sentences differs from that in subordinate clauses.

(5) Der Mann wird schnell gekommen
The man will quickly come
sein.
be.

Er weiß, daß der Mann schnell
He knows, that the man quickly
gekommen sein wird.
come be will.

For recursive phenomena KURD's flat matching approach is somewhat inconvenient.

In example 6, rule 2 applies to *die Werkzeuge*, although the article *die* should in fact be linked to *Arbeiter*, while *Werkzeuge* stands in a plural.

(6) die Werkzeuge herstellenden Arbeiter
the tools building workers

To handle such problems, one can try to enumerate the elements which can be contained between the two borders of a pattern to be recognized. But this approach mostly yields only approximate results because it does not respect the generative capacity of language.

However, most of the style warnings can easily be implemented in KURD, as the according pattern can still often be recognized by one or two elements at its borders.

The system has been tested against an analyzed corpus of approx. 76,000 sentences. More than 5,000 sentences to be ameliorated were detected by KURD. 757 of them were selected manually to control the validity of the rules of warning class (2) and (3): In 8% (64), application was wrong. In these cases, syntactical structure could not adequately be described in the KURD formalism. These 8%, however, have to be re-estimated to reflect even a better result. They do not cover sentences selected by simple rules such as those of class (1). Rules of this class are responsible for 20% of the automatically detected sentences to be ameliorated, but cannot apply wrongly.

In another test, a text of 30 pages was annotated by a human corrector and the KURD style checker. The results were compared. About 50% of the human annotations were also annotated by the computer with a comparable amelioration proposal. 35% resist an automatic diagnosis, either because the recursive structure could not adequately be modeled by the style checking rules, or because the information calculated by the morphological analysis was not sufficient (i.e. no semantic information is available). By formulating

new style rules, 65% of recall could be achieved. The precision of the style checker, on the other hand, seems a critical point. The checker produces three times more automatic warnings than the human corrector. This is mainly due to the 'Counting rules', because the count limits were often too low. The choice of acceptable limits is still under discussion.

It has been shown that pattern recognition could be a valuable means for applications needing at least basic information on syntactic structures and that KURD could be a tool for realizing these applications.

References

Steven Abney. 1996. Partial Parsing via Finite-State Cascades. In *Proceedings of the ESSLLI '96 Robust Parsing Workshop 4*.

Claudia Fottner-Top. 1996. Workshop: Erstellung von verständlicher und benutzerfreundlicher technischer Dokumentation. Working paper, Institut für Technische Literatur, München.

Johann Haller. 1996. MULTILINT, A Technical Documentation System with Multilingual Intelligence. In *Translating and the Computer 18*, London. Aslib, The Association for Information Management, Information House.

Heinz-Dieter Maas. 1996. MPRO - Ein System zur Analyse und Synthese deutscher Wörter. In Roland Hausser, editor, *Linguistische Verifikation, Sprache und Information*. Max Niemeyer Verlag, Tübingen.

Randall Sharp and Oliver Streiter. 1995. Applications in Multilingual Machine Translation. In *Proceedings of The Third International Conference and Exhibition on Practical Applications of Prolog, Paris, 4th-7th April*. URL: <http://www.iai.uni-sb.de/cat2/docs.html>.

Wojciech Skut, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. 1997. An Annotation Scheme for Free Word Order Languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP)*, Washington, D.C.