Shallow Post Morphological Processing with KURD

Michael Carl and Antje Schmidt-Wigger
email: carl,antje@iai.uni-sb.de
Institut für Angewandte Informationsforschung,
Martin-Luther-Straße 14, 66111 Saarbrücken
Germany

Abstract

In this paper we describe a constraint based formalism that manipulates sequences of morphological analyses in order to Kill, Unify, Replace or Delete parts of the structure. We compare the formalism to a similar approach (CGP) and describe two applications.

1 Introduction

In 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 analyses such as lemmatization or full derivational and compositional analyses, context-dependent disambiguation of tagging results, multi-word recognition, shallow, partial or full syntactic parsing, semantic analyses 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 tools can be guided by the data or the requirements and the prerequisites 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. (Karlsson, 1990),(Abney, 1996), (Declerck and Klein, 1997)). In this paper, we describe a simple formalism (KURD[1]) that is designed to perform some 'shallow' operations on morphologically analyzed texts. The output can be used directly, or be redirected to further processing.

Typical tasks for such shallow processing include

- **Tagging** (disambiguation of multiple morphological analyses)
  Often a set of simple rules that runs in a set order over the results of the morphological analyses 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.
- **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.
- **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.

[1] KURD is an acronym representing the first letters of the implemented *actions*: K(ill)-U(nify)-R(eplace)-D(elete)

Michael Carl and Antje Schmidt-Wigger (1998) **Shallow Post Morphological Processing with KURD.** In D.M.W. Powers (ed.) *NeMLaP3/CoNLL98: New Methods in Language Processing and Computational Natural Language Learning,* ACL, pp 257-265.

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 analysis[2] is the process of separating grammatical information and (a) stem(s) from the surface form of an input word. Lemmatization 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, lemma 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 op V) which we will refer to as feature bundles (FBs). 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 WDs. The distinction between WDs and deeper embedded FBs is useful later in this paper due to the important functional difference. The formal definition of a SD is as follows:

Sentence Descriptor SD:

$$
\begin{array}{rcl}
SD & ::= & WD \, , \cdots , \, WD \, . \\
WD & ::= & FB \\
FB & ::= & \{ \, AVS \, \} \; ; \; \cdots \; ; \; \{ \, AVS \, \} \\
AVS & ::= & A \;\; op \, V \, , \cdots , A \;\; op \;\; V \\
V & ::= & ATM \;\; | \;\; FB \, | \;\; VAR \\
ATM & ::= & atom \;\; ; \;\; \cdots \;\; ; \; atom \\
VAR & ::= & \text{'\_' followed by any string} \\
A & ::= & \text{any alpha-numeric string} \\
op & ::= & \texttt{=} \;\; | \;\; \texttt{~=}
\end{array}
$$

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 the 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 assumptions of the morphological analysis.

Word descriptor WD "der":

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

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

Both types of disjunction are shown in the representation of 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`", which has a complex disjunction as its value. The feature "`case`" in the first complex disjunctor 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 we do not need variable binding between different attributes of the same FB[3]. because we presume that each attribute in a (morphological) FB expresses a different piece of information (it thus has a different type).

3 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

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

[3] In 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 equal types.

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*s. While matching the *description* part of a rule onto a *SD*, *WD*s 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 (Kill, Unify, Replace or Delete) 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 one for a *count* of an *internal interval*.

3.1 Some examples

German verbs have detachable prefixes that can be homonyms to prepositions. Morphological analysis thus generates two interpretations for such a string. 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) Disambiguate_Prefix =
Ae $\left\{\texttt{c=w,sc=p}\right\}$ e $\left\{\texttt{c=vpref}\right\}$,
a $\left\{\texttt{c=w,sc=punct;comma}\right\}$:
Au $\left\{\texttt{c=vpref}\right\}$

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* is expected to be ambiguous with respect to its category. Feature cooccurrences are required in the first *test*, where both features c=w and sc=p must occur in conjunction in (at least) one interpretation 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* consists of one *test* only. The *FB* matches an 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* which is not an end-of-sentence item.
A *WD* for which the first *condition* is true is marked by the marker "A". The rule applies if the second *condition* is true for the following *WD*. The *action* part has one *consequence* that consists of one *act*. The *WD* which has been marked in the *description* part is 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=_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=_AGR}) into the respective features of the marked word descriptors.

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

The *description* part of rule (2) has three *conditions*. Each *condition* matches an *interval* of the *WD*s. The second *condition* can possibly be empty since it has the kleene star *scope* ('*'). All *WD*s 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. *WD*s 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) Reduce_Noun_Phrase =
Ae $\left\{\texttt{c=w,sc=art,agr=\_AGR}\right\}$,
*Aa $\left\{\texttt{c=adj,agr=\_AGR}\right\}$,
+Be $\left\{\texttt{c=noun,agr=\_AGR}\right\}$:
Ak{}, Br $\left\{\texttt{c=np,agr=\_AGR}\right\}$

The operator "r" in the second *conseqence* of the 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') by means of the kill operator `Ak{}` from the sentence descriptor such that only the *NP* node is printed as a result.

Style checking has often to deal with the complexity[4] 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 $\left\{\texttt{wnrr=1,vtyp\textasciitilde=fiv}\right\}$,
8e $\left\{\texttt{sc\textasciitilde=comma;cit;slash}\right\}$ |
a $\left\{\texttt{vtyp\textasciitilde=fiv}\right\}$;$\left\{\texttt{c\textasciitilde=verb}\right\}$,
Aa $\left\{\texttt{c=verb,vtyp=fiv}\right\}$:
Au$\left\{\text{warning='405'}\right\}$

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 *WD*s other than finite verbs. This is expressed by the external *test* (`{vtyp~=fiv};{c~=verb}`) following the vertical bar. The internal *test* (`{sc~=comma;cit;slash}`), e.g. the part before the vertical bar counts the number of words in the *count* different from punctuation marks and slashes. 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.

3.2 Formal Definition

The formal definition of rule syntax is given below:

Definition of rule:

rule	::=	*name*'=' *descr* ':' *action*
descr	::=	$condition_1$ ',' $condition_2$ ...
condition	::=	*interval* | *count*
interval	::=	[*scope*][*marker*] $test_1$ $test_2$ ...
count	::=	*num* $interval_{int}$ '|' $interval_{ext}$
test	::=	*quantifier FB*
action	::=	$conseq_1$ ',' $conseq_2$ ...
conseq	::=	*marker* act_1 act_2 ...
act	::=	*operator FB*
scope	::=	∧ | + | * | -
marker	::=	A | ... | Z
num	::=	0 | ... | 99
operator	::=	k | u | r | d
quantifier	::=	e | a

Whether or not a rule applies (i.e. its *action* part is executed or not) depends on whether its *conditions* match. Each *condition* matches the longest possible sequence of *WD*s and, once matched, other segmentations are not considered. We do not foresee backtracking or multiple solution generation. The length of an *interval* 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* matches zero or more words.
- + The *interval* matches at least one word.
- - The *interval* matches one and only one word. This is the default value for the scope.

[4] The 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.

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 unifyable 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*uences of the *action* part are executed if the *descr*iption part matches. The *acts* of a *conseq*uence 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 SD or a WD where one of the *external tests* is *false*. The outcome of a *count* (whether it is *true* or *false*) 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.

4 Related Work

In order to compare KURD with other postmorphological processing systems, one can distinguish between the formalisms' design, the implementation of a grammar and the tasks for which the system is designed. Most such comparisons (e.g. (Abney, 1996)) are based on processing time, accuracy and recall, which in fact do not differentiate between the strength of the formalism and the strength of the grammar actually implemented.
In this section we want to compare the capacities of KURD to another formalisms by describing its formal characteristics for each possible step in the chain of NLP application. Two concrete applications will be presented in the following section.

Similar to KURD, CGP of the 'Helsinki' project (cf. (Karlsson, 1990)) is a system working on morphologically analysed text that contains lexical ambiguities. KURD and CGP are somewhat alike with respect to the basic assumptions on steps one would need to disambiguate morphological descriptions: an ambiguous word (WD) is observed in its context. If necessary it has to be acertained that the context itself is not ambiguous. In a fitting context the disambiguation operation is triggered. The realization of these assumptions in the two formalisms differs in the following features:

In KURD ...

- a rule definition is based on pattern matching of a specific context, in which the action's focus is than selected.
- the scope of disambiguation is fixed by means of markers. This allows more than one operation to be defined in the marked scope (WDs) at a time, and the same operation to be applied to more than one word (WD).
- the context of an operation and the operation itself are defined in separate parts of the rule. Each part may contain a distinct set of features while in CGP, all features specified for the focused word are subject to the same disambiguation.
- variable binding is supported. Multiple interpretations of several words can be disambiguated by unification as exemplified in rule (2). In CGP, rule batteries are necessary for this task, and disambiguation of the combination of features of more than two WD is not possible.
- unbounded dependencies can be modeled by means of intervals. We are not sure whether these can be modeled in CGP by means of relative positions.

In CGP ...

- the focus of the rule is positioned before the left- and rightward context is described.

- one can look backwards in a context. This is not always possible in KURD due to underspecification in the morphological input.
- one can define sets of features. In KURD, this can be modeled by means of feature disjunction; thus more freedom in KURD, but less consistency.
- one can discard a reading when the context is NOT realised. In KURD, these possibility can only be modeled using two rules and a meta-feature.
- there is a specific clause boundary mode. In KURD, clause boundaries have to be enumerated as simple features.

To summarize the comparison, backward looking seems basically the only difference with which CGP has an advantage over KURD in terms of expressiveness, while variable binding gives KURD advantage over CGP. In terms of user-friendliness, the systems choose two different directions. In KURD the use of markers and rule separation into a description part and an action part may reduce the number of rules, while CGP allows for the simplification of rules by means of sets or the clause boundary mode.

The next step in processing moves from the treatment of words towards the treatment of word groups i.e. to parsing. Traditional parsers are full parsers building all possible deep parse trees over the flat input structure. Weaker models, usually referred to as 'shallow parsers' (cf. (Karlsson and Karttunen, 1997)), allow for partial parses, for trees of depth of one or for one result only. The output data structure of a parser is generally a bracketed structure which preserves the original morphological flat structure inside the output structure. Some shallow parsers, however such as CGP, assign syntactic functions to the words of a sentence and renounce the representation of the dependency structure.

Parsing with KURD results in a one level representation where the nodes (WD) can be enriched with information concerning their syntactic functions. The insertion of brackets is not supported in KURD but recognized phrases can be reduced to one node if they are part of higher level phrases. Also recursivity of language has to be approximated by means of iterative, multiple application of (not necessarily the same) rule set. Thus KURD has to be classified as a typical shallow parsing system, also allowing for partial parsing.

The last step in the NLP processing chain is a practical application of the linguistic knowledge for a specific task. The next section describes such an application of KURD for style checking. It does not rely on a full disambiguation and syntactic tagging of the morphological analysis. Disambiguation is undertaken only when necessary. We believe that 100% disambiguation is too expensive for a rule based system[5] especially when it has to be adapted to each new text type. In the next section, we show that good results can also be obtained on ambiguous input.

5 Style checking

In this section we want to describe an application of KURD 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, stylistic guidelines have been formulated, such as:

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

The compilation of these guidelines has influenced the architecture of KURD to a certain extent. Most scientists correlate the readability of a sentence with its complexity, defined often by length,

[5] CGP contained 400 rules for 90% disambiguation quality (cf. (Karlsson, 1990)). In order to reach nearly 100%, this number increased up to 1100 rules... cf. (Karlsson and Karttunen, 1997)

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 using 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 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 warning): These warnings can either recognize the complex internal structure of compound words, or forbid the use of a certain word. For the latter 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 warning):
These warnings react to complex syntactic structures and trigger the proposition 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 warning): 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 (see footnote 5), but it does not allow the triggering of a concrete reformulation proposition 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 bare 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 have been easily implemented in KURD, as the appropriate 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), the warnings had been applied incorrectly. In these cases, syntactic structure could not adequately be described in the KURD formalism. These 8%, however, only reflects the erroneous results of warning classes 2 and 3. They do not cover sentences selected by simple rules such as those of class 1. Rules of warning class 1 are responsible for 20% of the automatically detected sentences to be ameliorated. These rules do never apply incorrectly.
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 proposition. 35% resisted 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 was available). By writing new style rules, a 65% recall could be achieved. The precision of the style checker, on the other hand, seems to be 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.

6 Chunk Reduction and Refinement

In the framework of the CBAG[6] module (cf. (Carl, 1998) in this volume) KURD is used in several components. CBAG is an example based translation engine whose aim it is to be used as a stand-alone Example Based Machine Translation system (EBMT) or to be dynamically integrated as a front-end into a Rule Based Machine Translation system.

The CBAG module is divided into three submodules:

- The Case Base Compilation module (CBC) compiles a set of bilingual SD equivalences into a case base thereby inducing case abstractions from the concrete SD. Case abstractions ensure a greater recall and are thus needed for a better coverage of the system.

- The Case Based Analysis module (CBA) decomposes and reduces an input SD into a set of chunks according to the cases in the case base. Reduced sequences are more likely to match a case in the case base because they are shorter abstractions from the original sequence of WDs.

- The Case Based Generation module (CBG) re-generates sequences of target language WDs from the reduced chunks. In the refinement process lexical and grammatical information are merged together into WDs.

[6] CBAG stands for Case Based Analysis and Generation

KURD is used for two different tasks in these modules. In the CBC module and in the CBA module, KURD performs chunk reduction and in the CBG module, KURD performs chunk refinement.

In order to do chunk reduction, the input SD is first decomposed into a sequence of chunks according to the entries in the case base. KURD reduces those chunks which match a case in the case base into one chunk descriptor according to the schema of rule 3.

In the refinement phase, KURD merges lexical and grammatical information which is extracted from two different sets of cases. These rules use all types of operators that are available in KURD.

7 Implementation

The KURD formalism is implemented in C and compilable under gcc. It runs on sparc workstations and is currently ported to PC (with gcc).

8 Conclusion

In this paper we have presented a constraint-based formalism (KURD) that manipulates morphological analysis in order to kill, unify, replace or delete parts of the structure. The formalism realizes a pattern matching approach that is suitable for shallow and/or partial NLP.

First, we give a formal definition of the data structure and of the formalism and discuss a few example rules in order to present the capacities of the formalism.

KURD is then compared to another similar formalism (CGP) and it is found that both formalisms have a comparable expressiveness. Whereas in KURD the use of variables and markers makes the rule writing easier, CGP allows for the simplification of rules by means of sets or the clause boundary mode.

Two applications of KURD are presented. In two large-scale experiments it could be shown that style-checking can be realized by KURD with a reasonable result. In a small experiment it is shown that KURD can be used for shallow parsing and refinement in a MT application.

9 Acknowledgement

We would like to thank Munpyo Hong and Cath Pease for valuable comments.

References

Steven Abney. 1996. Partial Parsing via Finite-State Cascades. In *Proceedings of the ESSLLI '96 Robust Parsing Workshop4*.

Michael Carl. 1998. A constructivist approach to MT. In *Proceedings of NeMLaP*, Sydney.

Thierry Declerck and Judith Klein. 1997. Ein Email-Korpus zur Entwicklung und Evaluierung der Analysekomponente eines Terminvereinbarungssystems. In *Konferenzbeiträge der 6.Fachtagung der DGfS-CL*.

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.

Fred Karlsson and Lauri Karttunen. 1997. Subsentential processing. In G.B. Varile and A. Zampolli, editors, *Survey of the State of the Art in Human Language Technology*, volume Vol. XII+XIII of *Linguistica Computazionale*. Giardini Editori e Stampatori, Pisa.

Fred Karlsson. 1990. Constraint grammar as a framework for parsing running text. In *COLING-90*, volume 3, pages 168–173.

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.