A Type-Driven Method for Compacting *MMorph* Resources

Hans-Ulrich Krieger and Feiyu Xu
Language Technology Lab
German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
{krieger|feiyu}@dfki.de

Abstract

This paper describes an offline compaction method that removes both redundancies and spurious ambiguities from *MMorph* lexical databases. The described technique increases the efficiency of systems using *MMorph*, since it shrinks the size of the lexicons and comes up with fewer readings for a morphological form. Our approach not only is interesting to *MMorph*, but also to other lexicons, which build on an attribute-value representation of lexical information. The compaction method is part of the *SProUT* shallow processing system.

1 Motivation

In multilingual text processing systems, performance is heavily affected by the size and reusability of objects that encode linguistic knowledge. Thus, building compact lexical databases supporting efficient operations is an important research and engineering task.

Our compaction method has been developed as part of the *SProUT* system (Shallow Processing with Unification and Typed feature structures), a platform for the construction of multilingual shallow text processing systems (Becker *et al.* 02). The system consists of linguistic processing resources for seven languages (Chinese, English, French, German, Japanese, Italian, Spanish), and provides a grammar development and testing environment. In *SProUT*, typed feature structures (TFSs) are used as the uniform data interchange format between components and therefore, morphological information in *SProUT* is represented via TFSs. Transduction rules in *SProUT* do not rely on simple atomic symbols, but instead on TFSs, where the left-hand side of a rule is a regular expression over TFSs, representing the recognition pattern, and the right-hand side is a sequence of TFSs, specifying the output structure. Consequently, equality of atomic symbols is replaced by unifiability of TFSs and the output is constructed using TFS unification w.r.t. a type hierarchy (see section 3.1 for an informal explanation).

Morphological resources are built on top of the full form lexical databases of *MMorph*. However, many lexical entries possess spurious ambiguities within *MMorph*. When integrating *MMorph* lexicons as they are, a runtime system might have a serious space problem, and in particular, performs redundant unifications. This paper describes an approach, which compacts *MMorph* resources by replacing several readings through a compact reading, by deleting redundant readings, and by substituting specialized readings through more general ones, using type generalization and subsumption checking. These techniques go hand in hand with a moderate enlargement of the original type hierarchy.

2 *MMorph*

MMorph has been developed at ISSCO/Geneva (Petitpierre & Russell 95). It provides an environment for the development and compilation of lexical databases, and allows to access the results of morphological analysis. The full form lexicons used in our application are available for five languages, viz., English (approx. 200,000 entries), German (830,000), French (225,000), Italian (330,000), and Spanish (570,000). Initial lexical entries are collected from various sources (ISSCO, Sardic, and the Web). In addition, many regular entries are automatically derived from already existing ones by means of extended two-level rules (Koskenniemi 84).

Entries in an *MMorph* full form database relate word forms to their base forms and their morphosyntactic descriptions (MSDs), which are sets of flat feature-value pairs. In *MMorph*, most lexical entries have more than one inflectional reading; e.g., in the original German lexicon, the average number of readings is 3.2. We have observed that there are many redundant readings among these ambiguities. Figure 1 demonstrates an extreme case, the German word "evaluierten" (*to evaluate, evaluated*), which has 11 readings. The MSD of the first reading assigns, e.g., the attribute `person` the value `1|3`, meaning `1` *or* `3`, where `|` serves as a syntactic delimiter, indicating a disjunctive description. When carefully studying the MSDs and comparing them to each other, we discover that several entries are redundant, e.g., reading 3, 4, 6,

7, 9, and 10 are specific cases of reading 11 and can therefore be deleted.

3 Compacting *MMorph*

Before presenting the details of the compaction algorithm, additional technical information is required.

3.1 Background

Central to TFSs is an operation which combines the information from two feature structures w.r.t. a type hierarchy: unification. The resulting unique single structure contains the information provided by the input structures, but nothing more. If the input structures contain conflicting information, unification is said to be failed.

Informally, a feature structure can be seen as a collection of feature-value pairs, where a feature expresses a functional (linguistic) property and the value of a feature might again be a feature structure (or an atom), thus we allow for recursive embeddings. An important characteristic of feature structures is that they provide coreference constraints, meaning that two features share exactly one common value. This concept allows for the transport of information and is exhaustively used in unification-based grammars, where features on the left-hand side (LHS) of a grammar rule share values with other features on the right-hand side (RHS).

Feature structures can also be given a type, which ultimately leads to TFSs. First of all, a type can be seen as a compact abbreviation for a TFS, supporting clarity and easy modifiability of descriptions (type definition). Furthermore, types can be arranged in a type hierarchy, allowing multiple inheritance of information from all supertypes; for more information on this theme, see (Carpenter 92).

Since *MMorph* entries in the *SProUT* shallow processor are translated into TFSs, it is essential to guarantee that TFS unification is an efficient operation: firstly, unifiability is used by the *SProUT* interpreter during the matching phase of the LHS of a rule and secondly, unification is employed during structure building on the RHS (see (Becker *et al.* 02) for more information). Efficiency is addressed on the feature term level by a lazy-copying unifier which is a variant of (Emele 91) and during type unification by a sophisticated greatest lower bound (GLB) caching mechanism, based on a bit-vector encoding of types (Kiefer *et al.* 99).

The complexity of computing a new GLB is linear in the number of types (in the best case even logarithmic), according to a method introduced by (Aït-Kaci *et al.* 85). The idea here is to establish an injection γ between the original type hierarchy $\mathcal{T}$ and another partial order $\mathcal{B}$ (of bit-vectors) which allows a faster computation of GLBs. In case that $\mathcal{T}$ is a lower semilattice (or a bounded complete partial order, BCPO), which is guaranteed to be the case for our initial type hierarchy, the inverse mapping γ^{-1} is also an injection. The computation of a GLB for two type s, t then reduces to the computation of

$$\gamma^{-1}(\gamma(s) \otimes \gamma(t))$$

whereas $\otimes$ denotes the bit-wise AND operation on the two bit-vectors $\gamma(s)$ and $\gamma(t)$. Given a bit-code c, $\gamma^{-1}(c)$ is given by

$$\gamma^{-1}(c) = \lceil \{s \in \mathcal{T} \mid \gamma(s) \leq c\} \rceil$$

meaning that $\gamma^{-1}(c)$ is the set of maximal elements from $\mathcal{T}$, whose codes are less than c. In case $\mathcal{T}$ is a BCPO, this set consists of a single unique element, as explained above.

The cached items (pairs of types mapping onto their GLBs) can be retrieved even in constant time on the average, due to a technique that is used in compiler technology when mapping a multi-dimensional array onto the (one-dimensional) main memory. Assume that a type s is represented by an integer $id(s)$ and that the set of types is given by $\mathcal{T}$. The GLB of two types s and t can then be realized by a (hash) table lookup, whereas the unique key (an integer) is computed as

$$id(s) + |\mathcal{T}| \times id(t)$$

and the corresponding value is exactly $\mathrm{GLB}(s, t)$.

We note here that type hierarchies which are not bounded complete (i.e., there exist pairs of types which do not have a unique maximal lower bound, i.e., a GLB) can be transformed into order-preserving BCPOs by using a completion method devised in (Aït-Kaci *et al.* 89). As stated above, type hierarchies in *SProUT* are always BCPOs, due to an offline application of the *flop* preprocessor of *PET* (Callmeier 00), which performs (besides other things) the completion of arbitrary type hierarchies.

The TFS unifier above is part of the JTFS package, a freely available implementation of TFSs (Krieger 02). JTFS reads in a binary representation of a typed unification grammar (the output of the *flop* preprocessor), including type hierarchy and lexicon, and builds up the objects in main memory. JTFS supports a dynamic extension of the type hierarchy at run time to

1. `Verb[mode=indicative vform=fin tense=imperfect number=plural person=1|3 particle_verb=none ...]`
2. `Verb[mode=subjunctiveII vform=fin tense=imperfect number=plural person=1|3 particle_verb=none ...]`
3. `Adjective[gender=masc number=singular case=gen|acc degree=pos spelling=unchanged stts_open=adja]`
4. `Adjective[gender=neutrum number=singular case=gen degree=pos spelling=unchanged stts_open=adja]`
5. `Adjective[gender=masc|fem|neutrum number=plural case=dat degree=pos spelling=unchanged stts_open=adja]`
6. `Adjective[gender=masc number=singular case=gen|dat|acc degree=pos spelling=unchanged stts_open=adja]`
7. `Adjective[gender=fem|neutrum number=singular case=gen|dat degree=pos spelling=unchanged stts_open=adja]`
8. `Adjective[gender=masc|fem|neutrum number=plural case=nom|gen|dat|acc degree=pos spelling=unchanged ...]`
9. `Adjective[gender=masc number=singular case=gen|dat|acc degree=pos spelling=unchanged stts_open=adja]`
10. `Adjective[gender=fem|neutrum number=singular case=gen|dat degree=pos spelling=unchanged stts_open=adja]`
11. `Adjective[gender=masc|fem|neutrum number=singular case=nom|gen|dat|acc degree=pos spelling=unchanged ...]`

Figure 1: The 11 readings for *evaluierten*. We only display the morphosyntactical features. The format of the entries is determined by *MMorph*. The entries result from a full form *MMorph* dump.

allow for unknown words. Other operations, such as subsumption and equivalence checking, fast unifiability testing, deep copying, path selection, feature iteration, and different printers are available.

3.2 Restrictions

Due to the fact that the lazy-copying TFS unifier in *SProUT* only provides conjunctive descriptions, we translated the original *MMorph* lexicons into disjunctive normal form (DNF) to have a first running system. For instance, the disjunctive reading 11 in figure 1 resulted in $3 \times 1 \times 4 \times 1 \times 1 = 12$ (conjunctive) readings. In general, moving to DNF is not a bad idea and must not lead to a degradation of efficiency as (Kiefer *et al.* 99) have shown for several large HPSG grammars (they even gained a speedup of a factor of two as a result of using a lazy-copying unifier, instead of a non-lazy disjunctive unifier).

However, one can have the best of the two worlds, at least in our setting here: a lazy-copying unifier plus disjunctive descriptions which, however, are encoded as additional types in the original type hierarchy. In general, expressing a disjunctive feature constraint, e.g.,

$$[\,f \;\; s\,] \vee [\,f \;\; t\,]$$

through an equivalent feature constraint employing a new disjunctive type, e.g.,

$$[\,f \;\; s\_or\_t\,]$$

is not always possible, due to a theoretical and a very practical fact.

3.2.1 Coreferences

The difference between type and token identity might get lost in case a coreference is located inside a TFS, which is imploded into a new type; e.g., if the value under feature f in

$$\begin{bmatrix} f & [\,g & \boxed{1} s\,] \\ h & \boxed{1} & \end{bmatrix}$$

would be substituted by type t, representing TFS

$$[\,g \;\; s\,]$$

we could no longer distinguish the above description from

$$\begin{bmatrix} f & [\,g & s\,] \\ h & s & \end{bmatrix}$$

since both are now of the form

$$\begin{bmatrix} f & t \\ h & s \end{bmatrix}$$

Clearly, we can maintain the coreference $\boxed{1}$ by reduplicating feature g, say, on top of the TFS, calling it g'

$$\begin{bmatrix} f & [\,g & \boxed{1} s\,] \\ h & \boxed{1} & \\ g' & \boxed{1} & \end{bmatrix}$$

so that the substitution would yield

$$\begin{bmatrix} f & t \\ h & \boxed{1} s \\ g' & \boxed{1} \end{bmatrix}$$

This, however, will result in a global reformulation of all other TFSs which are suspect to potentially unify with the above structure and by making sure that g' is a new feature, not used anywhere before.

3.2.2 Combinatorics

In case that the number of appropriate values for a given feature is large (e.g., the KEY feature in HPSG grammars) or even potentially infinite (e.g., the morphological string form), it would not be a good idea to represent all possible combinations of values through a type hierarchy. Assuming, we have n different values for a given feature, the number of possible combinations (= number of new types) is $2^n - n - 1$ (the n original types are already there, plus the bottom type $\perp$, always expressing inconsistent knowledge).

However, the above two points are not applicable here. Firstly, *MMorph* entries do not specify coreference constraints (only implicit through atomic values which can be seen as always coreferent). Secondly, we employ the above power set construction only for single features which comes up with a relatively small number of types, but do not mould several features and their values into a single type (at least not at the moment).

3.3 Compaction Method

Given a full form database, containing entries such as the example in figure 1, we store information for the same word form (example: *evaluierten*) in an index structure of the following form (POS abbreviates part-of-speech):

word form	$\rightarrow$	POS_1	$\rightarrow$	stem_{11}	$\rightarrow$	set of MSDs
				...		...
				stem_{1m}	$\rightarrow$	set of MSDs
		⋮		⋮		⋮
		POS_n	$\rightarrow$	stem_{n1}	$\rightarrow$	set of MSDs
				...		...
				stem_{nk}	$\rightarrow$	set of MSDs

An MSD (i.e., a set of flat feature-value pairs) is encoded as a table of the following form:

feature_1	$\rightarrow$	set of appropriate values
...		...
feature_l	$\rightarrow$	set of appropriate values

Given a set of MSDs M for a word form, the compacting method applies the following operations to arbitrary $m_1, m_2 \in M$, until M remains constant (i.e., until a fixpoint is reached):

1. **equality test**
   if $m_1 = m_2$, then remove m_1 from M.
2. **subsumption test**
   if the set of values for the features in m_1 is a subset of values of features in m_2, then remove m_1 from M (m_2 is more general than m_1).
3. **set union**
   if m_1 differs from m_2 at only one feature f, then merge the two values, remove m_1 from M, and replace the value of f in m_2 by v, where $v := m_1@f \cup m_2@f$ denotes the union of the two sets (generalize m_1 and m_2).

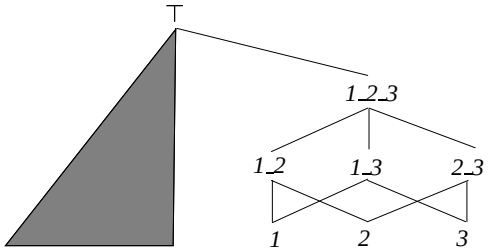
3.4 Generating a Type Hierarchy

During the analysis of a full form lexicon, we collect

1. POS information together with their features and
2. features together with their appropriate values.

From 1. and 2., we generate type definitions (i.e., a type hierarchy, plus appropriateness specifications).

For example, the PERSON feature in German or English has the three values 1, 2, and 3. The type hierarchy will then be (we omit $\perp$):



The type definitions are generated as $\mathcal{TDL}$ statements (Krieger & Schäfer 94) and are processed by the *flop* preprocessor of *PET* (see section 3.1), so that they can be incorporated into the *SProUT* system:

```
1_2_3 :< *top*.
1_2 :< 1_2_3.
1_3 :< 1_2_3.
2_3 :< 1_2_3.
1 := 1_2 & 1_3.
2 := 1_2 & 2_3.
3 := 1_3 & 2_3.
```

The new types names are generated by a power set construction of the appropriate values for the features of interest. The specific form of the type names originates from a lexicographical ordering of the symbols (so that we have, e.g., `1_2`, but not `2_1`). Given the natural order $\subseteq$ between elements of the power set, we can easily generate the above order of type definition statements. E.g., since $\{1,2\} \subseteq \{1,2,3\}$ is the case, we know that the definition for `1_2_3` must come before `1_2`. And since intersection corresponds to the GLB, we know, for instance, that `1` must inherit from `1_2` and `1_3`: $\{1,2\} \cap \{1,3\} = \{1\}$.

Given the MSDs (see figure 1), we generate further type definitions to represent inflectional information. There will be a type definition for each POS, consisting of features we are interested in (i.e., only a subset

of the *MMorph* features) and which are restricted by their most general type (e.g., 1_2_3 in the example), e.g.,

```
infl_verb :=
  infl & [PERSON 1_2_3,
          PARTICLE_VERB sep_part_...,
          TENSE imperfect_present,
          STTS_OPEN adja_adjd,
          NUMBER plural_singular,
          SPELLING new_old_unchanged,
          MODE imperative_indicative_...,
          VFORM fin_inf_infzu_prp_psp].
```

We note here that by adding new types, the GLB computation will still has a constant time complexity on the average, due to the table lookup technique described in section 3.1.

4 Results

After applying the compaction method to the German lexicon in DNF, the average number of readings has been reduced from 5.8 (in DNF) to 1.6 (with additional types), whereas the original German *MMorph* lexicon had 3.2 readings on the average (recall that the original *MMorph* entries employed atomic disjunctions). The most drastic improvements are obtained for adjectives: 4.0 (original lexicon) vs. 1.7 readings (compacted lexicon). The size of the new lexicon is less than one third of the old in DNF: 0.86 GByte vs. 0.25 GByte. Only 195 type definitions are produced by the above method for the German lexicon. Overall, the average speedup measured for the German named entity grammars in *SProUT* was about a factor of 3.

The approach described here has even a repercussion on the original *MMorph* data base, since the compacted lexicons can be retranslated, thus helping to remove spurious ambiguities from *MMorph*. At the moment, we do not merge several features into a new (super-)feature (e.g., PERSON-NUMBER or PERSON-NUMBER-TENSE). It is worth considering this option, since it will further lower the ambiguity rate which will have a direct impact on runtime performance.

A related approach, although performed entirely manually, has been conducted by Dan Flickinger. (Flickinger 02) reports on experiments with a large HPSG grammar, which originally contained feature structure disjunctions (see section 3.2). By going to conjunctive descriptions (as we did), by introducing additional types, and by changing the grammar (in his approach: by hand), Flickinger achieved significant performance gains: parsing was approx. four times faster and required three times less space. He also combined several features and their values into new super-features and super-values. We are currently investigating the impact of such a packing of morphosyntactical information in *SProUT*. The described automated compaction method can be easily extended to handle such super-features/-values.

References

(Aït-Kaci *et al.* 85) Hassan Aït-Kaci, Robert Boyer, and Roger Nasr. An encoding technique for the efficient implementation of type inheritance. Technical Report AI-109-85, MCC, Austin, TX, 1985.

(Aït-Kaci *et al.* 89) Hassan Aït-Kaci, Robert Boyer, Patrick Lincoln, and Roger Nasr. Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems*, 11(1):115–146, January 1989.

(Becker *et al.* 02) Markus Becker, Witold Drożdżyński, Hans-Ulrich Krieger, Jakub Piskorski, Ulrich Schäfer, and Feiyu Xu. SProUT—Shallow processing with unification and typed feature structures. In *Proceedings of the International Conference on Natural Language Processing, ICON-2002*, 2002.

(Callmeier 00) Ulrich Callmeier. PET—A platform for experimentation with efficient HPSG processing. *Natural Language Engineering*, 6(1):99–107, 2000.

(Carpenter 92) Bob Carpenter. *The Logic of Typed Feature Structures*. Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, 1992.

(Emele 91) Martin Emele. Unification with lazy non-redundant copying. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 323–330, 1991.

(Flickinger 02) Dan Flickinger. On building a more efficient grammar by exploiting types. In S. Oepen, D. Flickinger, J. Tsuji, and H. Uszkoreit, editors, *Collaborative Language Engineering. A Case Study in Efficient Grammar-based Processing*, pages 1–17. CSLI Publications, 2002.

(Kiefer *et al.* 99) Bernd Kiefer, Hans-Ulrich Krieger, John Carroll, and Rob Malouf. A bag of useful techniques for efficient and robust parsing. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics, ACL-99*, pages 473–480, 1999.

(Koskenniemi 84) Kimmo Koskenniemi. A general computational model for word-form recognition and production. In *Proceedings of the 10th International Conference on Computational Linguistics, COLING-84*, pages 178–181, 1984.

(Krieger & Schäfer 94) Hans-Ulrich Krieger and Ulrich Schäfer. $\mathcal{TDL}$—A type description language for constraint-based grammars. In *Proceedings of the 15th International Conference on Computational Linguistics, COLING-94*, pages 893–899, 1994.

(Krieger 02) Hans-Ulrich Krieger. JTFS—a Java implementation of typed feature structures. Technical Report, DFKI, 2002.

(Petitpierre & Russell 95) Dominique Petitpierre and Graham Russell. MMORPH—The Multext Morphology Program, 1995. Multext Deliverable 2.3.1. ISSCO, University of Geneva.