Chapter 26

Default inheritance and derivational morphology

Stefan Müller
Humboldt University Berlin

This paper is a contribution to the discussion whether argument structure constructions should be treated phrasally or lexically. While lexical models can explain the interaction between argument structure constructions and derivational morphology in a straightforward way, the analysis of this interaction is a desideratum for phrasal analyses. This paper deals with the question whether type hierarchies together with default inheritance can be used to describe derivational morphology. Given the challenges provided by Krieger & Nerbonne (1993) it seems impossible to do derivation without embedding (that is something like morphological phrase structure rules with mother/daughter relations or lexical rules/constructions with an input/daughter and an output/mother) and as will become clear the price for doing derivation with default inheritance is very high indeed.

1 Introduction

Goldberg (1995) and Goldberg & Jackendoff (2004) argue that Resultative Constructions like those in (1) are best described by phrasal rules that contribute the part of meaning that is specific to such resultative constructions.

(1) a. The pond froze solid.

[1] I thank Ann Copestake for discussion and John Bateman, Dorothee Beermann, Hans-Ulrich Krieger, and Andrew McIntyre for discussion and for comments on earlier versions of Müller (2006), research that is related to the present paper. An earlier version of this paper was presented at the 2nd International Workshop on Constraint-Based Grammar, which was held in 2005 in Bremen. I thank all participants for discussion.

While preparing this paper I extended the explanations and added and updated references. I thank Antonio Machicao y Priemer for detailed comments which resulted in many clarifications in the revised paper. The revised paper is 21 pages long, which is way outside of the page limit of the present collection. So I decided to publish the earlier version with all its shortcomings. The interested reader is referred to the extended version, which can be downloaded at `http://hpsg.fu-berlin.de/~stefan/Pub/default-morph.html`.

b. The gardener watered the flowers flat.

c. They drank the pub dry.

Authors who work in the framework of Construction Grammar (CxG) usually capture generalizations about Constructions in inheritance hierarchies (Kay & Fillmore 1999; Goldberg 2003). Goldberg assumes that generalizations regarding active, passive, and middle variants of the Resultative Construction can be expressed this way. However, there are other ways to realize Resultative Constructions and it is not obvious how such patterns should be treated. The German examples in (2) show, that resultative constructions and derivational morphology interact (Müller 2002; 2003; 2006):

(2) a. *-ung* nominalizations:
Leerfischung[2] 'empty.fishing', *Kaputterschließung*[3] 'broken.development', *Kaputtmilitarisierung*[4] 'broken.militarization', *Gelbfärbung*[5] 'yellow.dyeing'

b. *-er* nominalizations:
Totschläger[6] 'dead.beater' or 'cudgel', *SFB-Gesundbeter*[7] 'SFB.healthy.prayer',
Ex-Bierflaschenleertrinker[8] 'ex.beer.bottles.empty.drinker'

c. marginally in *Ge- -e* nominalizations:
Totgeschlage[9] 'beating.to.death'

So if all generalizations about resultative constructions are captured in inheritance hierarchies, the derivational facts should be covered that way too.

Krieger & Nerbonne (1993) showed that derivational morphology cannot be modeled using (simple) inheritance hierarchies since recursion as for example in *Vorvorvorversion* 'prepprepreversion' cannot be covered in inheritance networks (Krieger & Nerbonne 1993). Since information about the prefix *vor-* is contained in *Vorversion* inheriting a second time from *vor-* would not add anything. Secondly, in an inheritance-based approach to derivation, it cannot be explained why *undoable* has the two readings that correspond to the two bracketings in (3), since inheriting information in different orders does not change the result.

(3) a. [un- [do -able]]

b. [[un- do] -able]

Proponents of CxG often refer to default inheritance (Goldberg 1995; Michaelis & Ruppenhofer 2001) and Michaelis & Ruppenhofer (2001) explicitely suggest an analysis of derivational morphology that is based on default inheritance. For the class

[2] taz, 20-06-1996, p. 6.
[3] taz, 02-09-1987, p. 8.
[4] taz, 19-04-1990, p. 5.
[5] taz, 14-08-1995, p. 3.
[6] taz, Bremen, 24-05-1996, p. 24 and taz, Hamburg 21-07-1999, p. 22
[7] taz, 25-08-1989, p. 20.
[8] taz, 13/14-01-2001, p. 32.
[9] Fleischer & Barz (1995: 208).

of *be*-Verbs they assume the Applicative Construction which derives for instance *beladen* from *laden* by overriding incompatible information of the base verb by material inherited from the *be-* Construction (p. 59).

This paper deals with the question whether type hierarchies together with default inheritance can be used to describe derivational morphology. Given the challenges provided by Krieger & Nerbonne (1993) it seems impossible to do derivation without embedding and as will become clear the price for doing derivation with default inheritance is very high indeed.

A full account of derivational morphology has to explain the following facts:

- derivation may change the phonological form (*Les+bar+keit* 'readability');
- derivation may change the syntactic category:
  Les+bar+keit 'readability' = V $\rightarrow$ A $\rightarrow$ N;
- derivation changes the semantic contribution:
  lesen(x,y) $\rightarrow$ modal(lesen(x,y)) $\rightarrow$ nominal(modal(lesen(x,y))).

The crucial point that has to be captured by every analysis is that there are productive morphological patterns. This means that it is not sufficient to specify two types in a type hierarchy and introduce explicitly a new subtype of the latter two types. One could do this for instance for *Lesbarkeit*. The category of the verbal stem and of all affixes would be specified as a default and a subtype for *lesbar* and *Lesbarkeit* is stipulated. In such a setting, the values that override defaults are stipulated for all lexemes, they do not follow from any rule. This is not adequate since it does not capture the productive aspect of many morphology patterns. What is needed is some way to automatically compute subtypes that correspond to stems or words. This can be done by an automatic closure computation that introduces new types for all compatible types specified in a type hierarchy. Such online type computation was suggested by Kay (2002) for phrasal Constructions in CxG and by Koenig (1999) in the framework of HPSG. However, such online computation makes it necessary to specify the default information in appropriate ways and to have some way to determine automatically in which ways default information may be overridden. In the remainder of the paper, I show how a default-based analysis has to be set up in order to capture the productive aspects of derivational morphology. I use the default logic described in Lascarides & Copestake (1999). In this formalization default information is explicitely marked, so the hierarchy may be set up in a way that it is clear which information overrides which other information.

2 Derivational morphology with default inheritance

The following subsection deals with changes in phonology, Subsection 2.2 deals with changes in syntactic category, and Subsection 2.3 captures semantics.

2.1 Changes in phonology

Villavicencio (2000: 86–87) suggested a way to extend the length of a list by using default unification. The trick is to mark the end of a list as default information. This information maybe overridden by inheritance from another type that specifies a conflicting value for the list end. For instance, the stem *les* can be specified as follows:

(4)
$$\begin{bmatrix} \textit{les} \\ \textsc{phon-h} & \begin{bmatrix} \textit{ne-list} \\ \textsc{hd} & \textit{les} \\ \textsc{tl} & /\,\textit{e-list} \end{bmatrix} \end{bmatrix}$$

e-list stands for empty list and *ne-list* for non-empty list.

Combining this stem with the suffix *-bar* in (5a) yields the representation in (5b):

(5) a.
$$\begin{bmatrix} \textit{bar} \\ \textsc{phon-h} & \begin{bmatrix} \textit{ne-list} \\ \textsc{tl} & \begin{bmatrix} \textit{ne-list} \\ \textsc{hd} & \textit{bar} \\ \textsc{tl} & /\,\textit{e-list} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

b.
$$\begin{bmatrix} \textit{les} \wedge \textit{bar} \\ \textsc{phon-h} & \begin{bmatrix} \textit{ne-list} \\ \textsc{hd} & \textit{les} \\ \textsc{tl} & \begin{bmatrix} \textit{ne-list} \\ \textsc{hd} & \textit{bar} \\ \textsc{tl} & /\,\textit{e-list} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The PHON-H|TL value of (4) is overridden by the value in (5a). The new end of the list is in turn marked as default.

There is one little problem and one big problem with this approach. The little problem is that the elements in the PHON-H list are in the wrong order if the affix is a prefix. Consider the noun *Vorversion*. Since *Vor-* is a prefix it should appear before *Version*, but if we use the mechanism described above, affixes are appended at the end of the PHON-H list. This problem could be solved by making the values in the PHON-H list more complex: if new material is added at the end of the list, information about prefix/suffixhood is added as well. For a noun like *Anfahrbarkeit* as in *Die Anfahrbarkeit des Flughafens muß gewährleistet sein* 'The accessibility of the airport by car must be guaranteed', one would get ⟨ *fahr, an*-prefix, *bar*-suffix, *keit*-suffix ⟩. We would then use the following relational constraint that maps this list onto the actual phonological realization.

(6) compute_phon(⟨⟩, 1, 1).

compute_phon(
$$\left\langle \begin{bmatrix} \textit{prefix} \\ \textsc{phon} & 1 \end{bmatrix} \right\rangle \oplus 2,3,4$$
) if

compute_phon(2, ⟨1⟩ ⊕ 3,4).

compute_phon(
$$\left\langle \begin{bmatrix} \textit{suffix} \\ \textsc{phon} & 1 \end{bmatrix} \right\rangle \oplus 2,3,4$$
) if

compute_phon(2, 3 ⊕ ⟨1⟩,4).

The symbol '⊕' stands for *append*, a relation that concatenates two lists. `compute_phon` is defined recursively. The second and the third clause take one element – a prefix or a suffix, respectively – from the beginning of a list and then call `compute_phon` with a shorter list (2). The first clause ends the recursion. If the first argument contains an empty list, all affixes are processed and the second and the third argument are identified (1). When `compute_phon` is called initially the second argument is the root, e.g., *fahr* in *Anfahrbarkeit*. The list of affixes is passed to `compute_phon` as the first argument. If this list starts with a prefix, the PHON value of the prefix (1) is appended to the value in the second slot (3) and the result ⟨ 1 ⟩ ⊕ 3 is the second argument of the recursive call of `compute_phon`. The third clause is for suffixes and works parallel to the second clause, the only difference being that the phonology of the suffix is appended to the second argument at the end. By recursively working through the affix list in the first slot the list gets shorter while the list in the second slot gets longer. When the recursion ends due to exhaustion of the list of affixes, the phonological information is in the second slot. Clause one of `compute_phon` identifies the second and the third slot of the relational constraint. Since the third slot is just passed on in the second and third clause (4), the third slot will contain the result of the PHON computation.

The value that is determined by the relational constraint is declared to be the PHON value of the sign (1):

(7)
$$\begin{bmatrix} \textsc{phon} & \boxed{1} \\ \textsc{phon-h} & \begin{bmatrix} \textsc{hd} & [\textsc{phon}\ \boxed{2}] \\ \textsc{tl} & \boxed{3} \end{bmatrix} \end{bmatrix} \wedge \texttt{compute-phon}(\boxed{3},\boxed{2},\boxed{1})$$

`compute_phon` takes the phonology of the root (2) as its second argument and the remainder of the PHON-H list, which contains all the affixes, as its first argument (3).

This analysis also gets the bracketing problem in (3) right: for [un [do able]], we get ⟨ *do, able*-suffix, *un*-prefix ⟩ and for [[un do] able], we get ⟨ *do, un*-prefix, *able*-suffix ⟩. The phonology is the same in both cases, but the semantics differs. See Section 2.3 for the meaning representation.

While this solves the problem of ordering prefixes and suffixes, there remains an even bigger problem. This problem has to do with the question when the PHON value is determined. The computation of the PHON value has to happen at the interface between the lexicon and syntax, that is, at the moment when it is clear that no further affixes will be unified with the existing description. If one would have a PHON value for the stem *les*, this PHON value would also be part of the unification of *les* and *-bar*. If one computes the PHON value for *lesbar* one would get a conflict between the value of *les* and the computed value *lesbar*. It would not be an option to leave PHON values of stems underspecified and let the affix determine the PHON value of the whole construction since it is possible to have more than one affix.

2.2 Changes in part of speech

The change in part of speech can be explained in an analoguos fashion: the category information is stored in an auxiliary list (CAT-H) that is extended by the affix. A path equation is used to identify the last element of the auxiliary list with the actual category value (CAT). This path equation is specified to be default information. An affix can override this information with an explicit path inequation and add a new default path equation that points to a newly introduced element at the end of the auxiliary list.

2.3 Changes in semantics

For the meaning representation similar tricks can be applied, a difference being that we need embedding. Let us consider the noun *Vorversion*. The lexical item for *Version* is given in (8):

(8)
$$\begin{bmatrix} \textit{lex-version} & \\ \textsc{phon-h} & \begin{bmatrix} \textit{ne-list} \\ \textsc{hd} \;\; \textit{version} \\ \textsc{tl} \;\; /\, \textit{e-list} \end{bmatrix} \\ \textsc{sem} & \begin{bmatrix} \textit{ne-list} \\ \textsc{hd} \;\; \textit{version-rel} \\ \textsc{tl} \;\; /\, \textit{e-list} \end{bmatrix} \end{bmatrix}$$

This description says that the value of PHON-H is ⟨ *Version* ⟩ and that the value of SEM is ⟨ *version-rel* ⟩. The important part of the definition above is that the TL value is marked to be a default specification, i.e., this value may be overridden. Using lists for the representation of semantic information is also crucial for the mechanisms described below.

The following type for the prefix *Vor-* can be unified with the type *lex-version*. *pref-vor* contains information about the second element of the PHON-H list and the SEM list. The result of the unification of *lex-version* and *pref-vor* is given in (9b):

(9) a.
$$\begin{bmatrix} \textit{pref-vor} & \\ \textsc{phon-h} & \begin{bmatrix} \textsc{tl} & \begin{bmatrix} \textsc{hd} \;\; \textit{vor} \\ \textsc{tl} \;\; /\, \textit{e-list} \end{bmatrix} \end{bmatrix} \\ \textsc{sem} & \begin{bmatrix} \textsc{hd} & \boxed{1} \\ \textsc{tl} & \begin{bmatrix} \textsc{hd} & \begin{bmatrix} \textit{vor-rel} \\ \textsc{arg1} \;\; \boxed{1} \end{bmatrix} \\ \textsc{tl} & /\, \textit{e-list} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

b.
$$\begin{bmatrix} \textit{lex-version} \wedge \textit{lex-vor} & \\ \textsc{phon-h} & \begin{bmatrix} \textit{ne-list} \\ \textsc{hd} \;\; \textit{version} \\ \textsc{tl} \;\; \begin{bmatrix} \textit{ne-list} \\ \textsc{hd} \;\; \textit{vor} \\ \textsc{tl} \;\; /\, \textit{e-list} \end{bmatrix} \end{bmatrix} \\ \textsc{sem} & \begin{bmatrix} \textit{ne-list} \\ \textsc{hd} \;\; \boxed{1} \;\; \textit{version-rel} \\ \textsc{tl} \;\; \begin{bmatrix} \textit{ne-list} \\ \textsc{hd} \;\; \begin{bmatrix} \textit{vor-rel} \\ \textsc{arg1} \;\; \boxed{1} \end{bmatrix} \\ \textsc{tl} \;\; /\, \textit{e-list} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

So the prefix *Vor-* extends the PHON-H list and adds its phonological information. It also adds its semantic contribution and embeds the semantic contribution of the

type it was combined with (1). Of course the semantic representation in (9) is not satisfying, since the SEM list contains both a version-rel(X) and an embedded version-rel(X), but only the representation vor-rel(version-rel(X)) is appropriate for *Vorversion*; version-rel(X) is not. In the framework of *Minimal Recursion Semantics* (2005) pointers are used to identify the main semantic contribution of a linguistic object. This pointer would identify version-rel in the entry for *Version* and vor-rel in the lexical entry of *Vorversion*. The path equation pointing to the semantic contribution has to be specified as default information. An affix overrides this equation with a non-default inequation and adds a new default equation pointing to the meaning representation that it contributes.

3 Comments on defaults and recursion

Note that this analysis is basically a misuse of defaults. In classical knowledge representation defaults are used to say things like the following: birds have wings and they can fly. A penguin is a bird, it inherits the property of having wings from the super concept, but it overrides the property of being able to fly. In the analysis of derivation given above, a crucial property of a word, namely how it is pronounced, is overridden. This overriding can occur an unbounded number of times. This amounts to making a statement like the following: *Vorvorvorversion* is essentially *Version* except that it is pronounced differently and means something different.

Whether or not one sees ways around this problem, there is a more serious problem, namely that the types above do not account for *Vorvorversion*. To account for *Vorvorversion* we have to have a type that can be combined with structures that have two elements on their PHON list:

(10)
$$\begin{bmatrix} \textit{pref-vor-2} & \\ \text{PHON-H} & \begin{bmatrix} \text{TL}|\text{TL} & \begin{bmatrix} \text{HD} & \textit{vor} \\ \text{TL} & /\,\textit{e-list} \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{TL} & \begin{bmatrix} \text{HD} & \boxed{1} \\ \text{TL} & \begin{bmatrix} \text{HD} & \begin{bmatrix} \textit{vor-rel} \\ \text{ARG1} & \boxed{1} \end{bmatrix} \\ \text{TL} & /\,\textit{e-list} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

This means that we have to have infinitely many entries for each prefix, since we do not know the length of the lists of the stem the prefix combines with. For instance if *Vor-* combines with *An+kündig+ung* 'announcement', we would need a version of *Vor-* that attaches itself to the end of a three element list. To form *Vorvorankündigung* we need another *Vor-* that attaches to an even longer list.

One can imagine a way to fix this: instead of giving a fixed path to the end of the list in the definition of *pref-vor* or *pref-vor-2*, one could extend the formalism and allow for regular expressions in type declarations. The star after TL in the following

definition would mean that the feature description that follows it can be unified in after any sequence of TLs. Since the HD of the description following TL* is specified to be non-default information and all other PHON-H values are also non-default information, the expression following TL* can only be unified in at the very end of the list.

(11)
$$\begin{bmatrix} \textit{pref-vor} & \\ \textsc{phon-h} & \begin{bmatrix} \textsc{tl}^* & \begin{bmatrix} \textsc{hd} & \textit{vor} \\ \textsc{tl} & \textit{/ e-list} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Thus we get *Vorversion* and *Vorankündigung*. In addition to the PHON-H specification in (11), we would need a similar expression for the computation of the semantics.

But note what such an extension would lead to: if we unify the structure in (9b) with a prefix *Vor-* that contains a regular expression as the one above, we do not get a unique result. One possible result of unification would be the structure in (9b) itself, i.e., TL* is expanded as TL and another possible result would be the structure that corresponds to *Vorvorversion*. This is the case where TL* is expanded as TL|TL. This means that if we "apply" the prefix *Vor-* to *Vorversion* we get two results, one being spurious.

It could be argued that the spurious unification result mentioned above does not do any harm if we use a closure computation, since elements computed twice are not represented twice in the closure. But note that we have two independent regular expressions: one for extending the PHON-H list and one for extending the SEM list. Therefore using such regular expressions would not only result in spurious unification results it would also result in unwanted structures, for instance in a structure with PHON value *Vorversion* and meaning vor-rel(vor-rel(version-rel(X))). To fix this, one would have to introduce another extension of the formalism that allows one to use a certain expansion of a regular expression at various places in a feature description.

4 Conclusion

This discussion has shown that default inheritance can be used to model derivation only at a very high cost. To achieve this we need:

- an auxiliary list in which the affixes are collected in the order of application;
- a special marking of the elements in the list that indicates whether the item is a prefix or a suffix;
- a complex relational constraint that walks through the auxiliary list and computes the actual phonological form;
- regular expressions in type definitions that basically break everything we know about unification;

- variables that help us to use the same regular expression in a feature description;
- and some sort of automatic unification of lexical types to get recursion.
- In addition to all this additional machinery, we have misused the concept of defaults.

I consider this too high a price to pay. If one compares this approach to the simplicity of embedding constructions like those usually used in lexical rule-based approaches, it is clear which approach should be preferred. Such constructions can state whether they are prefix or suffix constructions simply by putting constraints on the order in which the phonology of the embedded object and the phonology contributed by the construction are concatenated.

Concluding this paper, we can state that derivation cannot be done without embedding in a reasonable way, the techniques developed here may be used to implement other things in inheritance networks, though.

The analysis is implemented in the LKB system (Copestake 2002) and the code of the implementation is available at `http://hpsg.fu-berlin.de/~stefan/Pub/default-morph.html`.

References

Copestake, Ann. 2002. *Implementing typed feature structure grammars* (CSLI Lecture Notes 110). Stanford, CA: CSLI Publications.

Copestake, Ann, Daniel P. Flickinger, Carl J. Pollard & Ivan A. Sag. 2005. Minimal recursion semantics: an introduction. *Research on Language and Computation* 4(3). 281–332.

Fleischer, Wolfgang & Irmhild Barz. 1995. *Wortbildung der deutschen Gegenwartssprache.* 2nd edn. Tübingen: Max Niemeyer Verlag.

Goldberg, Adele E. 1995. *Constructions: a Construction Grammar approach to argument structure* (Cognitive Theory of Language and Culture). Chicago/London: The University of Chicago Press.

Goldberg, Adele E. 2003. Words by default: the Persian complex predicate construction. In Elaine J. Francis & Laura A. Michaelis (eds.), *Mismatch: form-function incongruity and the architecture of grammar* (CSLI Lecture Notes 163), 117–146. Stanford, CA: CSLI Publications.

Goldberg, Adele E. & Ray S. Jackendoff. 2004. The English resultative as a family of constructions. *Language* 80(3). 532–568.

Kay, Paul. 2002. An informal sketch of a formal architecture for Construction Grammar. *Grammars* 5(1). 1–19.

Kay, Paul & Charles J. Fillmore. 1999. Grammatical Constructions and linguistic generalizations: the What's X Doing Y? Construction. *Language* 75(1). 1–33.

Koenig, Jean-Pierre. 1999. *Lexical relations* (Stanford Monographs in Linguistics). Stanford, CA: CSLI Publications.

Krieger, Hans-Ulrich & John Nerbonne. 1993. Feature-based inheritance networks for computational lexicons. In Briscoe, Copestake & de Paiva (eds.), *Inheritance, defaults, and the lexicon*, 90–136. Cambridge University Press.

Lascarides, Alex & Ann Copestake. 1999. Default representation in constraint-based frameworks. *Computational Linguistics* 25(1). 55–105.

Michaelis, Laura A. & Josef Ruppenhofer. 2001. *Beyond alternations: a Constructional model of the German applicative pattern* (Stanford Monographs in Linguistics). Stanford, CA: CSLI Publications.

Müller, Stefan. 2002. *Complex predicates: verbal complexes, resultative constructions, and particle verbs in German* (Studies in Constraint-Based Lexicalism 13). Stanford, CA: CSLI Publications.

Müller, Stefan. 2003. Solving the bracketing paradox: an analysis of the morphology of German particle verbs. *Journal of Linguistics* 39(2). 275–325.

Müller, Stefan. 2006. Phrasal or lexical constructions? *Language* 82(4). 850–883.

Villavicencio, Aline. 2000. The use of default unification in a system of lexical types. In Erhard W. Hinrichs, Walt Detmar Meurers & Shuly Wintner (eds.), *Proceedings of the ESSLLI-2000 Workshop on Linguistic Theory and Grammar Implementation*, 81–96. Birmingham, UK.