Nonmonotonic Sorts for Feature Structures

Mark A. Young
Artificial Intelligence Laboratory
The University of Michigan
1101 Beal Ave.
Ann Arbor, MI 48109
marky@caen.engin.umich.edu

Abstract

There have been many recent attempts to incorporate defaults into unification-based grammar formalisms. What these attempts have in common is that they all lose one of the most desirable properties of feature systems: namely, presentation order independence. This paper describes a method of dealing with defaults that retains order independence. The method works by making a strong distinction between strict and default information. The addition of nonmonotonic sorts allows default information to be carried in the feature structure while retaining a simple, deterministic unification operation. Monotonic feature structures are rederived through a satisfaction relation that is abstract in that it depends only on the ordering information for sorts.

Introduction

There have been many recent attempts to incorporate defaults into unification-based grammar formalisms. (Shieber 1987) suggests *add conservatively* and (1986) *overwrite* as default mechanisms for PATR-II. (Kaplan 1987) describes *priority union* for LFG. (Bouma 1990) describes a default unification operation for a generic UG. (Carpenter 1991) compares skeptical and credulous methods for default unification, and describes a method of default inheritance.

None of these schemes retain the property of presentation order independence. That is, $(F \sqcap_D G) \sqcap_D H$ may be very different from $(F \sqcap_D H) \sqcap_D G$ (where $\sqcap_D$ is the default unification operator). Since unification grammars are meant to be declarative, this is a big step backward. To make sure that the "right" answer comes out, these mechanisms are restricted in their use (usually to just the lexicon) and even there require that specific procedures be followed.

This paper describes an effort to put defaults into a declarative framework. Our object is to provide a theoretically "clean" mechanism—one that produces the right answers without any procedural entanglements. Feature structures should be simple bundles of information, and unification should combine them without any loss (or gain) of information. The method of nonmonotonic sorts meets these requirements.

This paper is organized as follows. We first describe our favorite way of defining feature structures. The method does not rely on this particular definition, but having a concrete system to work with will simplify the presentation. We then give a simple example that illustrates the way defaults are often used in unification systems. Thereafter we define nonmonotonic sorts and nonmonotonically sorted feature structures, and show how they apply to the example. We finish up with a slightly more involved example and a discussion of extensions to the method.

Notation

Feature structures are bundles of information. Each structure has a sort (drawn from a finite set $\mathcal{S}$), and a (possibly empty) set of attributes (drawn from a finite set $\mathcal{F}$). Each attribute has a value, which is again a feature structure. One feature structure may be the value of more than one attribute, so the whole structure is best thought of as a rooted, labeled di-graph. More precisely:

Definition 1 *A* feature structure *is a tuple* $\langle Q, q, \delta, \Theta \rangle$ *where*

- Q *is a finite set of nodes,*
- $q \in Q$ *is the root node,*
- $\delta : Q \times \mathcal{F} \rightarrow Q$ *is a partial* feature value *function that gives the edges and their labels, and*
- $\Theta : Q \rightarrow \mathcal{S}$ *is a* sorting *function that gives the labels of the nodes.*

This structure must be connected.

Although we prefer this definition, the notion of nonmonotonic sorts is not tied to it: any similar notion of feature structures will do. It is usual to require that the graph be acyclic, and not unusual to require that Θ be defined only for sink nodes.

We will assume that there is a partial order, $\prec$, defined on $\mathcal{S}$. This ordering is such that the greatest lower bound of any two sorts is unique, if it exists. In

VERB template
<past tense suffix> *isa* +te
<past participle prefix> *isa* ge+
<past participle suffix> *isa* +t

spiel lex VERB

MIXED-VERB template VERB
<past participle suffix> *isa* +en

mahl lex MIXED-VERB

STRONG-VERB template MIXED-VERB
<past tense suffix> *isa* $\emptyset$

zwing lex STRONG-VERB
<past tense stem> *isa* zwang
<past participle stem> *isa* zwung

Figure 1: Example Lexicon Entries

VERB template
<past tense suffix> *default* +te
<past participle prefix> *isa* ge+
<past participle suffix> *default* +t

spiel lex VERB

MIXED-VERB template VERB
<past participle suffix> *isa* +en

mahl lex MIXED-VERB

STRONG-VERB template MIXED-VERB
<past tense suffix> *isa* $\emptyset$

zwing lex STRONG-VERB
<past tense stem> *isa* zwang
<past participle stem> *isa* zwung

Figure 2: Example Lexicon with Defaults

other words, $\langle \mathcal{S} \cup \{\bot\}, \prec \rangle$ is a meet-semilattice (where $\bot$ represents inconsistency or failure). This allows us to define the most general unifier of two sorts as their greatest lower bound, which write as $a \wedge_{\mathcal{S}} b$. We also assume that there is a most general sort, $\top$, called *top*. The structure $\langle \mathcal{S}, \prec \rangle$ is called the *sort hierarchy*.

The main operation carried out on feature structures is unification. We write $F \sqcap G$ for the most general unifier of F and G. Since we are working with sorted feature structures, we will require a method of sorted unification (Walther 1988). Note that the restriction on the sort hierarchy will ensure that most general unifiers are unique if they exist at all.

Defaults and the Lexicon

Defaults most commonly appear in the lexicon of natural language systems. The formation of verb tenses is particularly suited for the use of defaults, and we will be using examples of tense formation to illustrate our method.

We assume that our lexicon is made up of a system of templates and lexical entries. These objects are arranged in an inheritance hierarchy, with templates corresponding to non-terminals of the grammar and lexical entries to terminals. Figure 1 shows a simple hierarchy for a subset of German verbs. There are three kinds of verbs indicated: weak, mixed, and strong. The VERB template holds information about weak verbs. It specifies that the suffix for the past tense is *+te*, and that the past participle prefix and suffix are *ge+* and *+t* respectively. The MIXED-VERB template indicates that it inherits information from VERB, and that the past participle suffix for mixed verbs is *+en*.

The lexical entry *mahl* is an instance of the MIXED-VERB template. It is intended that *mahl* will get *+te* as a past tense suffix and *ge+* and *+en* as past participle affixes. However, if we use only strict unification, we will find that the information specified for *mahl* is inconsistent. In particular, we get *+en* as a past participle suffix from MIXED-VERB, and *+t* from VERB. Since strict unification cannot deal with this situation, we need a form of default unification to save us. (We could stick to strict unification if we simply replicated the required information in every template. This would not only lose meaningful generalizations, but would also lead to much larger lexicons.)

Current methods to deal with defaults introduce an asymmetric "default unification" operation. Systems using these methods might start at the top of the hierarchy and work down toward the lexical entries, *replacing* inconsistent information as it is found. Or they might start at the lexical entries and work their way up, *conserving* information by ignoring conflicts. These conflict resolution methods are by their very nature sensitive to order of presentation. There is very much a notion of "what we knew before" and "what we are being told now." Such methods cannot hope to provide a declarative way of dealing with defaults. For that we need a whole new way of looking at the matter.

Nonmonotonic Sorts

To allow defaults in our feature structures, we introduce the notion of nonmonotonic sorts (NSs). The intuition behind a nonmonotonic sort is that it encodes both strict and default information, and it keeps the distinction between the two kinds of information. We cannot be sure what sort a NS will finally turn out to be. That is, a NS is a description that may be satisfied by various sorts.

Looking back at Fig. 1, we can see that the suffixes *+te* and *+t* are both default information. Each is contradicted by information lower in the hierarchy. The suffixes *+en* and $\emptyset$ and the prefix *ge+* are, on the other hand, strict information. They are not contradicted by information lower in the hierarchy. We make this information explicit in Fig. 2.

Nonmonotonic sorts are defined in terms of the

monotonic sorts they describe. That is, we start with the same hierarchy of basic sorts, $\langle \mathcal{S}, \prec \rangle$, with the same sort unification operation, $\wedge_S$ defined on $\mathcal{S}$. A nonmonotonic sort occurs when we say that a feature structure is "strictly of this sort" and "by default of that sort." The intent of Fig. 2 is that the past participle suffix of *mahl* will be strictly *+en* and only by default *+t*. Since the two are incompatible, *+en* will overrule *+t*, and *mahl* will get the correct suffix regardless of the order templates are scanned.

Once formed, NSs can be freely combined, leading to NSs with multiple strict and default sorts. The strict parts can be combined into a single sort through sort unification. Default parts cannot be so combined, however. This is because the two pieces of information—that this FS is by default of sort a and that it is also by default of sort b—are two independent pieces of information which may be independently overruled. The default parts must be kept separate, not only from the strict parts, but also from each other. But default sorts can be made more specific by adding the information that they must be subsumed by the strict information.

Definition 2 *A* nonmonotonic sort *is a pair* $\langle f, \Delta \rangle$ *where* $f \in \mathcal{S}$, *and* $\Delta \subseteq \mathcal{S}$ *such that for each* $d \in \Delta$, $d \prec f$.

We write $\mathcal{N}$ for the set of nonmonotonic sorts.

Note that a nonmonotonic sort can be made from any sort f, and subset D of $\mathcal{S}$. Those elements of D that are inconsistent with f can be discarded. The remaining sorts can then be sort-unified with copies of f, discarding duplicates. Nonmonotonic sort unification uses a similar method to combine the information from two NSs without loss or gain.

Definition 3 *The* nonmonotonic sort unifier *of nonmonotonic sorts* $\langle f_1, \Delta_1 \rangle$ *and* $\langle f_2, \Delta_2 \rangle$ *(written* $\langle f_1, \Delta_1 \rangle \wedge_{\mathcal{N}} \langle f_2, \Delta_2 \rangle$*) is the nonmonotonic sort* $\langle f, \Delta \rangle$ *where*

- $f = f_1 \wedge_S f_2$, *and*
- $\Delta = \{ d \wedge_S f \mid d \in \Delta_1 \cup \Delta_2 \wedge (d \wedge_S f) \prec f \}$.

The nonmonotonic sort unifier is undefined if $f_1 \wedge_S f_2$ *is undefined.*

The definition of $\wedge_{\mathcal{N}}$ allows defaults to be eliminated in two ways (neither of which changes the information present). If a default sort is inconsistent with new strict information, it is dropped (we follow the convention that if $d \wedge_S f$ is undefined, then $d \wedge_S f \prec f$ is false). Also, new strict information may be more specific than a default sort; the redundant default is then dropped.

Going back to our example above, the past participle suffix in VERB is of nonmonotonic sort $\langle \top, \{+t\} \rangle$, while that of MIXED-VERB is $\langle +en, \{\} \rangle$. Their nonmonotonic sort unifier is $\langle +en, \{\} \rangle$ since $+t \wedge_S +en$ is undefined.

Theorem 1 *Nonmonotonic sort unification is a commutative and associative operation.*

Proof: The method is defined simply in terms of sort unification and set union, both of which are commutative and associative. The proof that NS unification retains these properties is straightforward (though tedious). □

Because of this, we may meaningfully speak of *the* nonmonotonic unifier of a set of nonmonotonic sorts, and not worry about the order that they are presented in.

We said earlier that we could not be sure what sort a NS would turn out to be. We can, however, give the range of possible values. First we define a weaker notion of *explanation*. This notion is borrowed from Theorist (Poole *et al.* 1986). In that default reasoning system, a belief can be explained by showing that it follows from what is known together with a consistent set of hypotheses. In our case, what is known is the strict information, and the possible hypotheses are the default sorts. A set of hypotheses is consistent if its sort unification exists. Thus we have:

Definition 4 *A set* D *is said to be a* theory *of a NS* $\langle f, \Delta \rangle$ *if* $D \subseteq \Delta$ *and* $(\bigwedge_S D)$ *is defined. A theory* D explains *a sort* t *if* $t = f \wedge_S (\bigwedge_S D)$.

The reason for including f in the definition of explanation is that D may be empty (in which case we take $\bigwedge_S D$ to be $\top$). We will say that a NS, n, explains a sort, t, if there is a theory of n which explains t. The only sort explained by $\langle +en, \{\} \rangle$ is *+en* itself. On the other hand, $\langle \top, \{+t\} \rangle$ explains both $\top$ and *+t*.

Theorem 2 *For each sort* t *that* $\langle f, \Delta \rangle$ *explains, there is a unique* maximal *theory for* t, *equal to the union of all theories for* t.

Proof: Given any two theories for t, say D_1 and D_2, t is the greatest lower bound of $\{f\} \cup D_1$ as well as the greatest lower bound of $\{f\} \cup D_2$. Thus t is the greatest lower bound of $\{f\} \cup D_1 \cup D_2$. Thus $D_1 \cup D_2$ is a theory for t. This is easily extended to finitely many theories. □

This allows us to associate a unique "best" theory with each explained sort.

The set of sorts that $\langle f, \Delta \rangle$ explains is in general too large for our purposes. The idea of a default is that it is true unless proven false. Therefore, we want to restrict ourselves to sorts that include as many defaults as possible.

Definition 5 *A sort* t *is a* solution *for a NS* $n = \langle f, \Delta \rangle$ *if it is explained by a maximal theory of* n. *That is,*

- $M \subseteq \Delta$ *and* $t = f \wedge_S (\bigwedge_S M)$,
- *for all* $D \subseteq \Delta$ *such that* $M \subset D$, $(\bigwedge_S D)$ *is not defined.*

The solutions for $\langle +en, \{\} \rangle$ and $\langle \top, \{+t\} \rangle$ are *+en* and *+t* respectively. There may be multiple solutions to a single NS, because default sorts can be individually consistent with the strict sort while being inconsistent with each other. We say that a sort *satisfies* a NS n if

s is subsumed by some solution for n. Thus a solution for n is also a most general satisfier for n.

Theorem 3 *The solutions of a NS n are the informationally maximal sorts explained by n. That is, if t is a solution of n then there is no $s \prec t$ such that n explains s.*

Proof: Let t be a solution for $n = \langle f, \Delta \rangle$ and let T be the maximal theory for t. Assume there is an $s \prec t$ explained by n and let D be some theory of n that explains s. We then have:

$$\begin{aligned} f\wedge_S(\bigwedge_S(D \cup T)) &= f\wedge_S(\bigwedge_S D)\wedge_S(\bigwedge_S T) \\ &= (f\wedge_S(\bigwedge_S D))\wedge_S(f\wedge_S(\bigwedge_S T)) \\ &= s\wedge_S t \\ &= s \end{aligned}$$

Obviously $T \subset (D \cup T)$ (equality would require $s = t$ as well), so T fails to be a maximal theory of n. That means t is not a solution: a contradiction. □

Feature Structures with Defaults

We have now said just about everything we want to say about nonmonotonic sorts. Nonmonotonically sorted feature structures (NSFSs) are easy to define, given what has gone before. You can think of a NSFS as a FS that is decorated with NSs instead of sorts. While this view makes them easier to understand, it should be remembered that they are more properly thought of as descriptions of feature structures. In particular, each NSFS stands for its set of solutions.

Definition 6 *A* nonmonotonically sorted feature structure *is a tuple $\langle Q, q, \delta, \Omega \rangle$ where Q, q, and δ are defined as for feature structures, and Ω is a nonmonotonic-sorting function, $\Omega : Q \rightarrow \mathcal{N}$.*

Unification of NSFSs is carried out as for monotonic FSs, but with nonmonotonic sort unification replacing sort unification. Since nonmonotonic sort unification is order-independent, nonmonotonic unification is also free of ordering effects.

The notions of explanation, and solution are easily extended to NSFSs. A node's sort information is local to it, and so we only need to look at each node separately.

Definition 7 *A NSFS, $F = \langle Q, q_0, \delta, \Omega \rangle$,* explains *a feature structure, C, if and only if*

- *$C = \langle Q, q_0, \delta, \Theta \rangle$, and*
- *$\forall q \in Q.\Omega(q)$ explains $\Theta(q)$.*

C is a solution for F if $\Theta(q)$ is a solution for $\Omega(q)$ for each $q \in Q$.

A feature structure is said to *satisfy* a nonmonotonically sorted feature structure, F, if and only if it is subsumed by some solution for F.

VERB template
  <past tense suffix> *default* +ed
  <past participle suffix> *default* +ed

call lex VERB

DDOUBLE template VERB
  <past tense suffix> *default* +d+
  <past participle suffix> *default* +d+

nod lex DDOUBLE

STRONG template VERB
  <past tense suffix> *isa* $\emptyset$
  <past participle suffix> *isa* +en

beat lex STRONG

forbid lex DDOUBLE,STRONG
  <past tense stem> *isa* forbade

Figure 3: Multiple Defaults Lexicon

Given that we know how to find the solution(s) for a NSFS, it remains only to determine under what circumstances it is appropriate to do so. Since the purpose of the lexicon is to associate feature structures with lexical entries, it is clear that it is appropriate to take solutions at that level. Taking the solution at any other level would be an error. Nothing inherits from a lexical entry, so the information in a lexical entry cannot be overruled. Defaults appearing in templates, on the other hand, are likely to be overruled by information in lower templates and lexical entries.

Given these definitions, we can work out the full feature structures for the verbs given in Fig. 2. The lexical entry *mahl* gets from MIXED-VERB a strict value *+en* for the past participle suffix. From the VERB template it inherits a strict PP prefix, and a default PT suffix. The default PP suffix *+t* from VERB is incompatible with the strict PP suffix in MIXED-VERB, and so is dropped. There is only one solution to the resulting description, one with PT *mahlte* and PP *gemahlen*. Similarly, we get *spielte* and *gespielt* for the VERB *spiel*, and *zwang* and *gezwungen* for the STRONG-VERB *zwing*.

Structures with Multiple Sorts

The example given above was very simple, involving as it did a very flat sort structure. Fig. 3 uses a set of sorts with some more order. In particular, it represents a doubled final d with the *+d+* sort. (The example is only given to illustrate the method. It is not meant to represent an adequate linguistic analysis.) This combines with the *+ed* and *+en* sorts to make the *+ded* and *+den* sorts. The $\emptyset$ sort is incompatible with any other sort, and *+ed* is incompatible with *+en*. (See Fig. 4.)

The analysis for *call* is quite simple. It is a solution of the verb template, and so takes the default endings to yield *called* and *called*. The value for *beat* is the so-

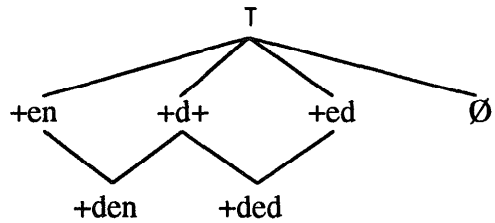

Figure 4: Simple Sort Hierarchy

lution to the STRONG template. Here the strict values for the suffixes overrule the defaults from VERB, and so the solution can be read directly from the template: *beat* and *beaten*.

The analysis for *nod* is not very complicated. The past tense suffix has two default values, *+ed* from VERB, and *+d+* from DDOUBLE, giving NS $\langle \top, \{+ed, +d+\}\rangle$. These are compatible, so both can be used in the solution, yielding a *+ded* suffix (*nodded*). The same reasoning exactly gives the (same) past participle, *nodded*.

The analysis for *forbid* is a little more complicated. Consider the past participle. The suffix here inherits three values: $\langle \top, \{+ed\}\rangle$ from VERB, $\langle \top, \{+d+\}\rangle$ from DDOUBLE, and $\langle +en, \{\}\rangle$ from STRONG. The *+en* is incompatible with the *+ed*, and so overrules it. The *+d+* is still consistent with the *+en*, and so is retained. The resulting NS is $\langle +en, \{+den\}\rangle$ (remembering to unify *+d+* with a copy of *+en*). The solution has *+den* as a suffix: *forbidden*. For the past tense, however, the $\emptyset$ suffix overrules both the *+ed* and *+d+*, meaning that we get $\langle \emptyset, \{\}\rangle$: *forbade*.

The analysis of *forbid* shows that nonmonotonic sorts can handle in a straightforward way what other systems require special methods for. Systems that conserve information have to inherit from STRONG before DDOUBLE so that the empty suffix for past tenses appears before the *+d+* (otherwise it would get **forbaded* as the past tense). Since there is no *a priori* reason to use this order, the system must give some way for the user to order templates other than simple inheritance order.

Systems that use replacement are even worse off for this example. These systems must start with VERB in order to correctly overrule the *+ed* suffixes. Given that, the information from STRONG must be added before that from DDOUBLE: otherwise the *+ed* and *+d+* sorts would be combined to form *+ded*, and then *+en* would overwrite them both (yielding **forbiden*). Conversely, the system must add the information from DDOUBLE *before* that from STRONG, lest the *+d+* overwrite the $\emptyset$ (giving **forbaded*). To solve this problem, the system must break one of DDOUBLE or STRONG into two parts, or must augment the default unification operator somehow.

Conclusion

In an effort to put defaults in a declarative framework, we have developed the concept of nonmonotonic sorts. Nonmonotonic sorts make a distinction between strict and default information. Maintaining this distinction allows us to define a simple unification operation that respects the information encoded in the nonmonotonic sorts. This operation allows us to use a single structure to keep track of multiple possible solutions in a completely deterministic way. Since the method is declarative, multiple inheritance with defaults can be carried out without regard to the order that information is presented.

The method as described above only deals with atomic information. The method can easily be extended to deal with path information (using a node instead of a sort). To say that two paths are by default equal means that the nodes at the ends of those paths are by default the same structure (as opposed to other systems, where it usually means that they have similar information by default). We are currently investigating the properties of such a system, including the question of how such unification tests would best be carried out (adding default equations leads to non-local effects in the NSFS).

We have already developed a version of the method that uses priorities on defaults sorts to choose between multiple solutions. The most natural system of priorities would mirror the structure of the lexicon. This extension allows for defaults appearing at lower levels of the lexicon to overrule those appearing at more general levels.

We are also investigating the effect of typing information (Carpenter 1992) on nonmonotonic sorts. Such information allows a limited form of inference to be carried out within a FS. As with default equations, this leads to non-local effects in the FS. In both cases, a solution FS may not be made up entirely of solution sorts, but will consist entirely of explained sorts.

References

Bouma, Gosse 1990. Defaults in unification grammar. In *Proceedings of the 1990 Conference of the Association for Computational Linguistics*. 165–172.

Carpenter, Bob 1991. Skeptical and credulous default unification with applications to templates and inheritance. In *Proceedings of Acquilex Workshop on Defaults and the Lexicon*.

Carpenter, Bob 1992. *The Logic of Typed Feature Structures*. Cambridge University Press. To appear.

Kaplan, Ronald 1987. Three seductions of computational linguistics. In *Linguistic Theory and Computer Applications*. Academic Press, London. 149–188.

Poole, David; Goebel, Randy; and Aleliunas, Romas 1986. Theorist: A logical reasoning system for defaults and diagnosis. Research Report CS-86-06, University of Waterloo.

Shieber, Stuart 1986. *An Introduction to Unification-Based Approaches to Grammar*, volume 4 of *CSLI Lecture Notes*. University of Chicago Press, Chicago.

Shieber, Stuart 1987. Separating linguistic analyses from linguistic theories. In *Linguistic Theory and Computer Applications*. Academic Press, London. 1–36.

Walther, Christoph 1988. Many-sorted unification. *Journal of the ACM* 35(1):1–17.