A LOGICAL SEMANTICS FOR NONMONOTONIC SORTS

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

Abstract

Suppose we have a feature system, and we wish to add default values in a well-defined way. We might start with Kasper-Rounds logic, and use Reiter's example to form it into a default logic. Giving a node a default value would be equivalent to saying "if it is consistent for this node to have that value, then it does." Then we could use default theories to describe feature structures. The particular feature structure described would be the structure that supports the extension of the default theory. This is, in effect, what the theory of nonmonotonic sorts gives you. This paper describes how that theory derives from what is described above.

INTRODUCTION

There have been many suggestions for incorporating defaults into unification-based grammar formalisms (Bouma 1990; Bouma 1992; Carpenter 1991; Kaplan 1987; Russell *et al.* 1992; Shieber 1986; Shieber 1987). Each of these proposes a non-commutative, non-associative default unification operation that combines one structure representing strict information with another representing default information. When presented with a set of structures, the result depends on the order in which the structures are combined. This runs very much against the unification tradition, in which any set has a unique most general satisfier (if a satisfier exists at all).

A method that is free of these ordering effects was presented in (Young 1992). The method of *nonmonotonic sorts* (NSs) allows default labels to be assigned at any time, and used only in the absence of conflicting information. NSs replace the more traditional labels on feature structures to give nonmonotonically sorted feature structures (NSFSs). These structures can be combined by an associative and commutative unification operation. FSs are rederived from NSFSs by taking a solution—an operation defined in terms of information present in the NSFS.

The original presentation of nonmonotonic sorts provided only a description of their operation and an informal description of their meaning. In this paper, we present a logical basis for NSs and nonmonotonically sorted feature structures (NSFSs). NSFSs are shown to be equivalent to default theories of default logic (Reiter 1980). In particular, we show how nonmonotonic sort unification is equivalent to finding the smallest default theory that describes both NSFSs; and also how taking a solution for a NSFS is the same as finding an extension for that theory.

FEATURE SYSTEMS

Unification-based grammar formalisms use formal objects called *feature structures* to encode linguistic information. We use a variant of the standard definition. Each structure has a sort (drawn from a finite set $\mathcal{S}$), and a (possibly empty) set of attributes (drawn from a finite set $\mathcal{F}$).

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

- Q *is a finite set of nodes,*
- $r \in Q$ *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.

It is not unusual to require that these structures also be acyclic. For some systems Θ is defined only for sink nodes (PATR-II, for example). Fig. 1 shows a standard textual representation for a FS.

We sometimes want to refer to substructures of a FS. If $\mathcal{A}$ is a feature structure as described above, we write $\mathcal{A}/f$ for the feature structure rooted at $\delta(q, f)$. This feature structure is defined by $Q' \subseteq Q$, the set of nodes that can be reached from $\delta(r, f)$. We will use the letter p (possibly subscripted) to represent paths (that is, finite sequences from $\mathcal{F}^*$). We will also extend δ to have paths in its second

<subj agr person> *isa* 3rd
<subj agr number> *isa* singular
<subj agr> = <pred agr>
<pred actor> = <subj>
<pred rep> *isa* sleep
<pred tense> *isa* present

Figure 1: Textual Feature Structure: "Uther sleeps."

$TRUE$
$FALSE$
a where $a \in \mathcal{S}$
$p_1 \doteq p_2$ where each $p_i \in \mathcal{F}^*$
$f : \phi$ where $f \in \mathcal{F}$ and $\phi \in FML$
$\phi \wedge \psi$
$\phi \vee \psi$

Figure 2: SFML: the domain of sorted logical formulas.

position, with the notion of iterated application of δ.

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 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*.

KASPER-ROUNDS LOGIC

(Kasper 1988) provides a logic for describing feature structures. Fig. 2 shows the domain of these logical formulas. We use the standard notion of satisfaction. Let $\mathcal{A} = \langle Q, r, \delta, \Theta \rangle$.

1. $\mathcal{A} \models TRUE$ always;
2. $\mathcal{A} \models FALSE$ never;
3. $\mathcal{A} \models a \iff \Theta(r) \preceq a$;
4. $\mathcal{A} \models p_1 \doteq p_2 \iff \delta(r, p_1) = \delta(r, p_2)$;
5. $\mathcal{A} \models f : \phi \iff \mathcal{A}/f$ is defined and $\mathcal{A}/f \models \phi$;
6. $\mathcal{A} \models \phi \wedge \psi \iff \mathcal{A} \models \phi$ and $\mathcal{A} \models \psi$;
7. $\mathcal{A} \models \phi \vee \psi \iff \mathcal{A} \models \phi$ or $\mathcal{A} \models \psi$

Note that item 3 is different than Kasper's original formulation. Kasper was working with a flat sort hierarchy and a version of FSs that allowed sorts only on sink nodes. The revised version allows for order-sorted hierarchies and internal sorted nodes.

NONMONOTONIC SORTS

Figure 3 shows a lexical inheritance hierarchy for a subset of German verbs. The hierarchy specifies

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

spiel lex VERB

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

mahl lex MIDDLE-VERB

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

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

Figure 3: Example Lexicon with Defaults

strict (*isa*) and default (*default*) values for various suffixes. If we ignore the difference between strict and default values, we find that the information specified for the past participle of *mahl* is inconsistent. The MIDDLE-VERB template gives *+en* as the suffix, while VERB gives *+t*. The declaration of the latter as a default tells the system that it should be dropped in favour of the former. The method of nonmonotonic sorts formalizes this notion of separating strict from default information.

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

The first element, s, represents the strict information. The default sorts are gathered together in Δ. We write $\mathcal{N}$ for the set of nonmonotonic sorts.

Given a pair of nonmonotonic sorts, we can unify them to get a third NS that represents their combined information.

Definition 3 *The* nonmonotonic sort unifier *of nonmonotonic sorts* $\langle s_1, \Delta_1 \rangle$ *and* $\langle s_2, \Delta_2 \rangle$ *is the nonmonotonic sort* $\langle s, \Delta \rangle$ *where*

- $s = s_1 \wedge_{\mathcal{S}} s_2$, *and*
- $\Delta = \{ d \wedge_{\mathcal{S}} s \mid d \in \Delta_1 \cup \Delta_2 \wedge (d \wedge_{\mathcal{S}} s) \prec s \}$.

The nonmonotonic sort unifier is undefined if $s_1 \wedge_{\mathcal{S}} s_2$ *is undefined. We write* $n_1 \wedge_{\mathcal{N}} n_2$ *for the NS unifier of* n_1 *and* n_2.

The method strengthens consistent defaults while eliminating redundant and inconsistent ones. It should be clear from this definition that NS unification is both commutative and associative. Thus we may speak of *the* NS unifier of a set of NSs, without regard to the order those NSs appear. Looking back to our German verbs example, the past participle suffix in VERB is $\langle \top, \{+t\} \rangle$, while that of MIDDLE-VERB is $\langle +en, \{\} \rangle$. The lexical entry for *mahl* gets their nonmonotonic sort unifier, which is $\langle +en, \{\} \rangle$. If $+t \wedge_{\mathcal{S}} +en$ *had* been defined, and equal to, say, $+ten$, then the NS unifier of $\langle\top, \{+t\}\rangle$ and $\langle +en, \{\}\rangle$ would have been $\langle +en, \{+ten\}\rangle$.

Once we have nonmonotonic sorts, we can create nonmonotonically sorted feature structures (NSFSs) by replacing the function $\Theta : Q \rightarrow \mathcal{S}$ by a function $\Omega : Q \rightarrow \mathcal{N}$. The nodes of the graph are thus labeled by NSs instead of the usual sorts. NSFSs may be unified by the same procedures as before, only replacing sort unification at the nodes with nonmonotonic sort unification. NSFS unification, written with the symbol $\sqcap_N$, is associative and commutative.

NSFSs allow us to carry around default sorts, but has so far given us no way to apply them. When we are done collecting information, we will want to return to the original system of FSs, using all and only the applicable defaults. To do that, we introduce the notions of explanation and solution.

Definition 4 *A sort t is said to be* explained *by a nonmonotonic sort $\langle s, \Delta\rangle$ if there is a $D \subseteq \Delta$ such that $t = s \wedge_S (\bigwedge_S D)$. If t is a maximally specific explained sort, then t is called a* solution *of n.*

The solutions for $\langle +en, \{\}\rangle$ and $\langle\top, \{+t\}\rangle$ are $+en$ and $+t$ respectively. The latter NS also explains $\top$.

Note that, while D is maximal, it's not necessarily the case that $D = \Delta$. If we have mutually inconsistent defaults in Δ, then we will have more than one maximal consistent set of defaults, and thus more than one solution. On the other hand, strict information can eliminate defaults during unification. That means that a particular template can inherit conflicting defaults and still have a unique solution—provided that enough strict information is given to disambiguate.

NSFS solutions are defined in much the same way as NS solutions.

Definition 5 *A FS $\langle Q, r, \delta, \Theta\rangle$ is said to be* explained *by a NSFS $\langle Q, r, \delta, \Omega\rangle$ if for each node $q \in Q$ we have $\Omega(q)$ explains $\Theta(q)$. If $\mathcal{A}$ is a maximally specific explained FS, then $\mathcal{A}$ is called a* solution.

If we look again at our German verbs example, we can see that the solution we get for *mahl* is the FS that we want. The inconsistent default suffix $+t$ has been eliminated by the strict $+en$, and the sole remaining default must be applied.

For the generic way we have defined feature structures, a NSFS solution can be obtained simply by taking NS solutions at each node. More restricted versions of FSs may require more care. For instance, if sorts are not allowed on internal nodes, then defining an attribute for a node will eliminate any default sorts assigned to that node. Another example where care must be taken is with typed feature structures (Carpenter 1992). Here the application of a default at one node can add strict information at another (possibly making a default at the other node inconsistent). The definition of NSFS solution handles both of these cases (and others) by requiring that the solution be a FS as the original system defines them. In both of these cases, however, the work can be (at least partially) delegated to the unification routine (in the former by allowing labels with only defaults to be removed when attributes are defined, and in the latter by propagating type restrictions on strict sorts).

What is done in other systems in one step has been here broken into two steps—gathering information and taking a solution. It is important that the second step be carried out appropriately, since it re-introduces the nonmonotonicity that we've taken out of the first step. For a lexicon, templates exist in order to organize information about words. Thus it is appropriate to take the solution of a lexical entry (which corresponds to a word) but not of a higher template (which does not). If the lexicon were queried for the lexical entry for *mahl*, then, it would collect the information from all appropriate templates using NSFS unification, and return the solution of that NSFS as the result.

DEFAULT LOGIC

The semantics for nonmonotonic sorts is motivated by default logic (Reiter 1980). What we want a default sort to mean is: "if it is consistent for this node to have that sort, then it does." But where Reiter based his DL on a first order language, we want to base ours on Kasper-Rounds logic. This will require some minor alterations to Reiter's formalism.

A *default theory* is a pair (D, W) where D is a set of default inferences and W is a set of sentences from the underlying logic. The default inferences are triples, written in the form

$$\frac{\alpha : \mathsf{M}\,\beta}{\gamma}$$

Each of the greek letters here represents a wff from the logic. The meaning of the default inference is that if α is believed and it is consistent to assume β, then γ can be believed.

Given a default theory (D, W), we are interested in knowing what can we believe. Such a set of beliefs, called an *extension*, is a closure of W under the usual rules of inference combined with the default rules of inference given in D. An extension E is a minimal closed set containing W and such that if $\alpha : \mathsf{M}\,\beta/\gamma$ is a default, and if $\alpha \in E$ and β consistent with E then $\gamma \in E$ (that is, if we believe α and β is consistent with what we believe, then we also believe γ).

Reiter can test a formula for consistency by testing for the absence of its negation. Since Kasper-Rounds logic does not have negation, we will not be able to do that. Fortunately, we have do have our own natural notion of consistency—a set of formulas is consistent if it is satisfiable. Testing a set of Kasper-Rounds formulas for consistency thus simply reduces to finding a satisfier for that set.

Formally, we encode our logic as an information system (Scott 1982). An information system (IS) is a triple $\langle A, C, \vdash \rangle$ where A is a countable set of "atoms," C is a class of finite subsets of A, and $\vdash$ is a binary relation between subsets of A and elements of A. A set X is said to be *consistent* if every finite subset of X is an element of C. A set G is *closed* if for every $X \subseteq G$ such that $X \vdash a$, we have $a \in G$. Following the style used for information systems, we will write $\overline{G}$ for the closure of G.

In our case, A is the wffs of SFML (except *FALSE*), and C is the class of satisfiable sets. The entailment relation encodes the semantics of the particular unification system we are using. That is, we have

$$\Gamma \vdash \beta \text{ if } \forall F.F \models \bigwedge \Gamma \Rightarrow F \models \beta.$$

For instance,

$$p_1 \doteq p_2, p_2 \doteq p_3 \vdash p_1 \doteq p_3$$

represents the transitivity of path equations.

DEFAULT KASPER-ROUNDS LOGIC

In the previous section we described the generic form of default logic. We will not need the full generality to describe default sorts. We will restrict our attention to closed precondition-free normal defaults. That is, all of our defaults will be of the form:

$$\frac{:\mathsf{M}\,\beta}{\beta}.$$

We will write D_β as an abbreviation for this default inference. Here β stands for a generic wff from the base language. Even this is more general than we truly need, since we are really only interested in default sorts. Nevertheless, we will prove things in the more general form.

Note that our default inferences are closed and normal. This means that we will always have an extension and that the extension(s) will be consistent if and only if W is consistent. These follow from our equivalents of Reiter's theorem 3.1 and corollaries 2.2 and 2.3.

Let's consider now how we would represent the information in Fig. 3 in terms of Kasper-Rounds default logic. The strict statements become normal KR formulas in W. For instance, the information for MIDDLE-VERBs (not counting the inheritance information) is represented as follows:

$$(\{\}, \{\mathit{past} : \mathit{participle} : \mathit{suffix} : +en\})$$

The information for VERB will clearly involve some defaults. In particular, we have two paths leading to default sorts. We interpret these statements as saying that the path exists, and that it has the value indicated by default. Thus we represent the VERB template as:

$$\begin{aligned} D &= \{D_{past:tense:suffix:+te}, \\ &\quad D_{past:participle:suffix:+t}\}, \\ W &= \{\mathit{past} : \mathit{tense} : \mathit{suffix} : \top, \\ &\quad \mathit{past} : \mathit{participle} : \mathit{suffix} : \top, \\ &\quad \mathit{past} : \mathit{participle} : \mathit{prefix} : ge+\} \end{aligned}$$

Inheritance is done simply by pair-wise set union of ancestors in the hierarchy. Since the entry for *mahl* contains no local information, the full description for it is simply the union of the two sets above.

$$\begin{aligned} D &= \{D_{past:tense:suffix:+te}, \\ &\quad D_{past:participle:suffix:+t}\}, \\ W &= \{\mathit{past} : \mathit{tense} : \mathit{suffix} : \top, \\ &\quad \mathit{past} : \mathit{participle} : \mathit{suffix} : \top, \\ &\quad \mathit{past} : \mathit{participle} : \mathit{prefix} : ge+, \\ &\quad \mathit{past} : \mathit{participle} : \mathit{suffix} : +en\} \end{aligned}$$

We can then find an extension for that default theory and take the most general satisfier for that formula. It is easy to see that the only extension for *mahl* is the closure of:

$$\begin{gathered} \mathit{past} : \mathit{tense} : \mathit{suffix} : +te, \\ \mathit{past} : \mathit{participle} : \mathit{suffix} : +en, \\ \mathit{past} : \mathit{participle} : \mathit{prefix} : ge+ \end{gathered}$$

The default suffix $+t$ is not applicable for the past participle due to the presence of $+en$. The suffix $+te$ is applicable and so appears in the extension.

DKRL AND NONMONOTONIC SORTS

In the previous section we defined how to get the right answers from a system using default sorts. In this section we will show that the method of nonmonotonic sorts gives us the same answers. First we formalize the relation between NSFSs and default logic.

Definition 6 *Let $\mathcal{D} = \langle Q, r, \delta, \Omega \rangle$ be a nonmonotonically sorted feature structure. The* default theory *of D is*

$$\begin{aligned} DT(\mathcal{D}) = & (\{D_{p:t} \mid \Omega(\delta(r,p)) = \langle s, \Delta \rangle \wedge t \in \Delta\}, \\ & \{\{p_1, p_2\} \mid \delta(r, p_1) = \delta(r, p_2)\} \\ & \cup \{p : s \mid \Omega(\delta(r,p)) = \langle s, \Delta \rangle\}) \end{aligned}$$

The default part of $DT(\mathcal{D})$ encodes the default sorts, while the strict part encodes the path equations and strict sorts.

Theorem 1 *The FS $\mathcal{A}$ is a solution for the NSFS $\mathcal{D}$ if and only if $\{\phi \mid \mathcal{A} \models \phi\}$ is an extension of $DT(\mathcal{D})$.*

Because we are dealing with closed normal default theories, we can form extensions simply by taking maximal consistent sets of defaults. This, of course, is also how we form solutions, so the the solution of a NSFS is an extension of its default theory.

We now need to show that NSFS unification behaves properly. That is, we must show that nonmonotonic sort unification doesn't create or destroy extensions. We will write $(D_1, W_1) =_\Delta (D_2, W_2)$ to indicate that (D_1, W_1) and (D_2, W_2) have the same set of extensions. We will do this by combining a number of intermediate results.

Theorem 2 *Let (D, W) be a closed normal default theory.*

1. *If $\alpha \wedge \beta \Leftrightarrow \gamma$, then $(D, W \cup \{\alpha \wedge \beta\}) =_\Delta (D, W \cup \{\gamma\})$.*
2. *If $W \cup \{\beta\}$ is inconsistent, then $(D \cup \{D_\beta\}, W) =_\Delta (D, W)$.*
3. *If $W \vdash \beta$, then $(D \cup \{D_\beta\}, W) =_\Delta (D, W)$.*
4. *If $W \vdash \alpha$ and $\alpha \wedge \beta \Leftrightarrow \gamma$, then $(D \cup \{D_\beta\}, W) =_\Delta (D \cup \{D_\gamma\}, W)$.*

The formulas α and β represent the (path prefixed) sorts to be unified, and γ their (path prefixed) greatest lower bound. The first part deals with strict sort unification, and is a simple consequence of the fact that (D, W) has the same extensions as $(D, \overline{W})$. The next two deal with inconsistent and redundant default sorts. They are similar to theorems proved in (Delgrande and Jackson 1991): inconsistent defaults are never applicable; while necessary ones are always applicable. The last part allows for strengthening of default sorts. It follows from the previous three. Together they show that nonmonotonic unification preserves the information present in the NSFSs being unified.

Theorem 3 *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be NSFSs. Then $DT(\mathcal{D}_1 \sqcap_N \mathcal{D}_2) =_\Delta DT(\mathcal{D}_1) \cup DT(\mathcal{D}_2)$ (using pairwise set union).*

DISCUSSION

Most treatments of default unification to date have been presented very informally. (Bouma 1992) and (Russell *et al.* 1992), however, provide very thorough treatments of their respective methods. Bouma's is more traditional in that it relies on "subtracting" inconsistent information from the default side of the unification. The method given in this paper is similar to Russell's method in that it relies on consistency to decide whether default information should be added.

Briefly, Bouma defines a default unification operation $A \sqcup! B = (A - B) \sqcup B$, where $A - B$ is derived from A by eliminating any path that either gets a label or shares a value in B. In the lexicon, each template has both "strict" and "default" information. The default information is combined

```
A template
        <f> isa a
        <g> default b
B template
        <f> default c
        <g> isa d
C lex A B
```

Figure 4: Multiple Default Inheritance

with the inherited information by the usual unification. This information is then combined (using $\sqcup!$) with the strict information to derive the FS associated with the template. This FS is then inherited by any children of the template.

Note that the division into "strict" and "default" for Bouma is only local to the template. At the next level in the hierarchy, what was strict becomes default. Thus "defaultness" is not a property of the information itself, as it is with NSs, but rather a relation one piece of information has to another.

The method described in (Russell *et al.* 1992) also divides templates into strict and default parts[1]. Here, though, the definitions of strict and default are closer to our own. Each lexical entry inherits from a list of templates, which are scanned in order. Starting from the lexical entry, at each template the strict information is added, and then all consistent defaults are applied.

The list of templates that the lexical entry inherits from is generated by a topological sort of the inheritance hierarchy. Thus the same set may give two different results based on two different orderings. This approach to multiple inheritance allows for conflicts between defaults to be resolved. Note, however, that if template A gets scanned before template B, then A must not contain any defaults that conflict with the strict information in template B. Otherwise we will get a unification failure, as the default in A will already have been applied when we reach B. With NSs, the strict information will always override the default, regardless of the order information is received.

The treatment of default information with NSs allows strict and default information to be inherited from multiple parents. Consider Fig. 4. Assuming that the sorts do not combine at all, the resulting FS for lexical entry C should be

$$\begin{bmatrix} f : a \\ g : d \end{bmatrix}$$

The two methods mentioned above would fail to get any answer for C: one default or the other would

[1]There may actually be multiple strict parts, which are treated as disjuncts, but that is not pertinent to the comparison.

be applied before the other template was even considered. In order to handle this example correctly, they would have to state C's properties directly.

One advantage of both Bouma and Russell is that exceptions to exceptions are allowed. With nonmonotonic sorts as we have presented them here, we would get conflicting defaults and thus multiple answers. However, it is straight-forward to add priorities to defaults. Each solution has a unique set of defaults it uses, and so we can compare the priorities of various solutions to choose the most preferred one. The priority scheme can be any partial order, though one that mirrored the lexical inheritance hierarchy would be most natural.

Another advantage that both might claim is that they deal with more than just default sorts. However, the theorems we proved above were proved for generic wffs of Kasper-Rounds logic. Thus any formula could be used as a default, and the only question is how best to represent the information. Nonmonotonic sorts are a concise and correct implementation of the kind of default inheritance we have defined here.

CONCLUSION

This paper has shown how the method of nonmonotonic sorts is grounded in the well-established theories of Kasper-Rounds logic and Reiter's default logic. This is, to our knowledge, the first attempt to combine Reiter's theory with feature systems. Most previous attempts to fuse defaults with feature structures have relied on procedural code—a state of affairs that is highly inconsistent with the declarative nature of feature systems. Methods that do not rely on procedures still suffer from the necessity to specify what order information is received in.

It seems to us that the major problem that has plagued attempts to add defaults to feature systems is the failure to recognize the difference in kind between strict and default information. The statement that the present participle suffix for English is '+ing' is a very different sort of statement than that the past participle suffix is '+ed' *by default*. The former is unassailable information. The latter merely describes a convention—that you should use '+ed' *unless you're told otherwise*. The method of nonmonotonic sorts makes this important distinction between strict and default information. The price of this method is in the need to find solutions to NSFSs. But much of the cost of finding solutions is dissipated through the unification process (through the elimination of inconsistent and redundant defaults). In a properly designed lexicon there will be only one solution, and that can be found simply by unifying all the defaults present (getting a unification failure here means that there is more than one solution—a situation that should indicates an error).

The semantics given for NSs can be extended in a number of ways. In particular, it suggests a semantics for one kind of default unification. It is possible to say that two values are by default equal by giving the formula $D_{p_1 \doteq p_2}$. This would be useful in our German verbs example to specify that the past tense root is by default equal to the present tense root. This would fill in roots for *spiel* and *mahl* without confounding *zwing*. Another extension is to use a prioritized default logic to allow for resolution of conflicts between defaults. The natural prioritization would be parallel to the lexicon structure, but others could be imposed if they made more sense in the context.

References

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

Bouma, Gosse 1992. Feature structures and nonmonotonicity. *Computational Linguistics* 18(2):183–203.

Carpenter, Bob 1991. Skeptical and credulous default unification with applications to templates and inheritance. In *Default Inheritance Within Unification-Based Approaches to the Lexicon.*

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

Delgrande, James P and Jackson, W Ken 1991. Default logic revisited. In *Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning.* 118–127.

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

Kasper, Bob 1988. *Feature Structures: A Logical Theory with Applications to Language Analysis.* Ph.D. Dissertation, University of Michigan, Ann Arbor.

Reiter, Ray 1980. A logic for default reasoning. *Artificial Intelligence* 13:81–132.

Russell, Graham; Ballim, Afzal; Carroll, John; and Warwick-Armstrong, Susan 1992. A practical approach to multiple default inheritance for unification-based lexicons. *Computational Linguistics* 18(3):311–337.

Scott, Dana 1982. *Domains for Denotational Semantics,* volume 140 of *Lecture Notes in Computer Science.*

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.

Young, Mark 1992. Nonmonotonic sorts for feature structures. In *National Conference on Artificial Intelligence*, San Jose, California. 596–601.