Technical Report

UCAM-CL-TR-238
ISSN 1476-2986

Number 238

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Proceedings of the ACQUILEX Workshop on Default Inheritance in the lexicon

Edited by Ted Briscoe, Ann Copestake, Valeria de Paiva

October 1991

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

https://www.cl.cam.ac.uk/

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

https://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Proceedings of the ACQUILEX Workshop on Default Inheritance in the Lexicon

Computer Laboratory
University of Cambridge
New Museums Site
Pembroke Street
Cambridge CB2 3QG

The ACQUILEX Esprit BRA (Basic Research Action) research project is concerned with the acquisition and representation of lexical information from machine readable dictionaries for use in natural language processing. The Cambridge group of the ACQUILEX project organised a Workshop on Default Inheritance in April 1991, the main purpose of which was to review approaches to default inheritance for lexical organisation and representation. The emphasis from ACQUILEX's point-of-view was in implementing a practical system capable of supporting substantial lexicons, based on existing proposals to incorporate (default) inheritance into a unification-based framework similar to DATR (Gazdar and Evans, 1989) and HPSG (e.g. Carpenter, 1990).

The Workshop consisted of two days of talks, where theoretical and implementational issues on default inheritance were discussed, as well as a last day of demonstrations of implemented systems. Papers from several European colaborative projects on the topics of the Workshop were presented - see enclosed list of titles and affiliations. The Cambridge ACQUILEX group presented and demonstrated the ACQUILEX lexical knowledge base (LKB) system and provided a tutorial on use of the software. The TFS system of the project POLYGLOSS and the system ELU of the group at ISSCO were also discussed and demonstrated.

Many thanks to all the participants for the lively discussions - exactly what workshops are supposed to be for.

Ted Briscoe
Ann Copestake
Valeria de Paiva

List of Participants in the Workshop

Name	Affiliation
Robert Carpenter	Carnegie Mellon University, Pittsburgh, USA
Roger Evans	School of Cognitive and Computing Sciences-University of Sussex
Jo Calder	University of Edinburgh
Ewan Klein	University of Edinburgh
Claire Grover	University of Edinburgh
Marc Moens	University of Edinburgh
Suresh Manandhar	University of Edinburgh
Michael Morreau	IMS - Stuttgart, Germany
Antje Rossdeutscher	IMS - Stuttgart, Germany
Remi Zajac	IMS - Stuttgart, Germany
John Nerbonne	University of Saarlandes, Saarbrucken, Germany
Hans-Ulrich Krieger	University of Saarlandes, Saarbrucken, Germany
Graham Russell	ISSCO - Geneva, Switzerland
Christoph Lehner	CIS - Munich, Germany
Petra Maier	CIS - Munich, Germany
Iskander Serail	University of Amsterdam, The Netherlands
Willem Meijs	University of Amsterdam, The Netherlands
Piek Vossen	University of Amsterdam, The Netherlands
Nicoletta Calzolari	Instituto di Linguistica Computazionale, Pisa Italy
Eugenio Picchi	Instituto di Linguistica Computazionale, Pisa Italy
Simonetta Montemagni	Instituto di Linguistica Computazionale, Pisa Italy
Joham Hagman	Instituto di Linguistica Computazionale, Pisa Italy
Felisa Verdejo	UPC - Barcelona, Spain
Horacio Rodriguez	UPC - Barcelona, Spain
Arthur Cater	University College, Dublin, Ireland
Cheng-ming Guo	University College, Dublin, Ireland
Ann Copestake	Computer Laboratory, University of Cambridge
Ted Briscoe	Computer Laboratory, University of Cambridge
Antonio Sanfilippo	Computer Laboratory, University of Cambridge
Valeria de Paiva	Computer Laboratory, University of Cambridge
John Carroll	Computer Laboratory, University of Cambridge

Papers presented at the Workshop

R. Carpenter	An Overview of Inheritance
G. Russell	A Practical Approach to Multiple Default Inheritance for Unification-Based Lexicons
R. Evans	Prioritised Multiple Inheritance in DATR
R. Zajac	Issues in the Design of a Language for Representing Linguistic Information based on Inheritance and Feature Structures
J. Calder	A Note on Priority Union
C. Lehner and P. Maier	Morphology and Grammar Development with PrologIII
P. Vossen	Extracting Taxonomies from Dictionary Definitions
A. Copestake/V. de Paiva/A. Sanfilippo	Functionality of the LKB
J. Nerbonne and U. Krieger	A Feature-Based Lexicon
R. Carpenter	Interpreting Concept Description Languages for Linguistic Knowledge Representation
A. Rossdeutcher	Remarks on Lexical Structure and DRS Construction
M. Morreau	Norms or Inference tickets- a collision of intuitions

Contents

Skeptical and Credulous Default Unification with Applications to Templates and Inheritance

Bob Carpenter

Philosophy Department
Carnegie Mellon University
Pittsburgh, PA 15213

carp@caesar.lcl.cmu.edu

Abstract

We present a definition of skeptical and credulous variants of default unification, the purpose of which is to add default information from one feature structure to the strict information given in another. Under the credulous definition, the default feature structure contributes as much information to the result as is consistent with the information in the strict feature structure.[1] Credulous default unification turns out to be non-deterministic due to the fact that there may be distinct maximal subsets of the default information which may be consistently combined with the strict information. Skeptical default unification is obtained by restricting the default information to that which is contained in every credulous result. Both definitions are fully abstract in that they depend only on the information ordering of feature structures being combined and not on their internal structure, thus allowing them to be applied to just about any notion of feature structure and information ordering. We then consider the utility of default unification for constructing templates with default information and for defining how information is inherited in an inheritance-based grammar. In particular, we see how templates in the style of PATR-II can be defined, but conclude that such mechanisms are overly sensitive to order of presentation. Unfortunately, we only obtain limited success in applying default unification to simple systems of default inheritance. We follow the Common Lisp Object System-based approach of Russell *et al.* (1990, this volume), in which information is always inherited from more specific sources before being inherited from more general sources. But in the case of orthogonal (multiple) inheritance in which information is inherited from distinct sources of incomparable specificity, a depth-first ordering is imposed based on the linear specification of the inheritance hierarchy.

[1] Jo Calder (1991) independently proposed a definition of "priority union" which is equivalent to our definition of credulous default unification.

1 Introduction

In many approaches to knowledge representation, some flavor of attribute-value logic is used to characterize objects in the empirical domain. In particular, the most well-known non-transformational linguistic formalisms such as LFG, GPSG and HPSG are (or can be) expressed in terms of attribute-value logics with fairly straightforward semantics. From the point of view of natural language processing, the attribute-value logics employed in these formalisms can be effectively characterized in terms of feature structures, where logical conjunction can be efficiently computed by unification. Our presentation in this paper is based on feature structures, but it naturally generalizes to alternative treatments of attribute-value logics such as those proposed by Aït-Kaci (1986), Pereira and Shieber (1984), Johnson (1986) and Smolka (1988).

Most current linguistic theories such as LFG, HPSG, GPSG, CG, GB along with many others, partition grammatical information into syntactic and lexical components. The distinction between the two components usually comes down to the fact that syntax involves some notion of recursive structure, while the lexicon is where information about the linguistic categories assigned to surface expressions is used to ground out the structural recursion. These linguistic theories employ syntactic components which are highly streamlined, with the burden of the representational work being placed on the lexicon. It is not unusual to see grammars with lexical categories represented by feature structures with more than one hundred nodes. Thus there is a strong demand for expressive methods of lexical knowledge representation and in particular, methods for expressing generalizations. But most useful lexical generalizations have exceptions and it is laborious to constantly introduce new concepts into an inheritance hierarchy to cover exceptional cases. Our focus in this paper is the extension of lexical inheritance to defaults. Of course, when grammar rules are stated in the same language as the lexicon, as in HPSG (Pollard and Sag 1987), FUG (Kay 1984) and PATR-II (Shieber *et al.* 1983), then the techniques presented here can be applied to syntactic information organization as well.

In unification-based grammar processing systems such as PATR-II and its descendents, a collection of hierarchically dependent templates may be defined and then incorporated into the definition of lexical entries and grammar rules. Using an example from HPSG (Pollard and Sag 1987), one template can be used for information about subcategorization, while another can contain information about agreement. Such templates are orthogonal, while a template for a transitive verb with a sentential object is more specific than a template for a transitive verb which is in turn more specific than the template for an arbitrary lexical verb which is itself more specific than the template for an arbitrary verbal category. By making use of such templates, lexical entries and grammar rules can be expressed much more succinctly than if the lexical entry for each word had to independently specified.

It is widely believed that the organization of practical knowledge representation systems should allow for some notion of information that can be obtained by default. But as soon as the term "default" is introduced, a number of possible interpretations immediately present themselves. Thus we try to make clear up front which particular kind of default reasoning we try to capture. We follow the intuition that defaults provide a method for allowing information to be deduced about an object if it is consistent with what is already known about the object. We also follow a number of other motivations, which include the restriction that more specific information should overrule information gained from more general sources. Such intuitions are guided by the (im)famous Tweety Triangle in which Tweety is a penguin, penguins are birds, birds are fliers, but penguins are non-fliers.[2] Here the intuition is that Tweety is not a flier as the information that Tweety is a penguin is more specific than the information that Tweety is a bird. But other intuitions are tested when we consider the pacifism of Nixon in the Nixon Diamond example in which Nixon is both a quaker and a republican, while quakers are pacifists and republicans are non-pacifists. We consider ways to resolve information-conflicts related to that found in the Nixon Diamond. There are two sources in which such conflicts can arise. The first is when we are unifying one feature structure by default into another and there are two pieces of information in the default feature structure which are consistent with the strict information, but are inconsistent with the strict information when taken together. The second is when we are resolving conflicts across levels of an inheritance hierarchy as in the Nixon Diamon example itself.

We consider two distinct approaches to default unification. Both approaches are non-symmetric in that they are designed to take one feature structure representing strict information and add in the information in a second feature structure representing default information. The methods vary in terms of how much inconsistency they are willing to tolerate before concluding that a piece of default information should be discarded. Credulous unification gets its name from the fact that it tries to add as much of the default information as is possible to the strict information without creating an inconsistency. As there may in general be distinct maximal subsets of the default information which can be consistently added to the strict information, credulous default unification may return more than one result. For instance, in the Nixon Diamond case, a credulous reasoner would conclude non-deterministically that either Nixon is a pacificist or that Nixon is not a pacificist. On the other hand, skeptical unification is deterministic in that it keeps all and only the default information that is contained in every credulous result. Thus in a skeptical approach to reasoning, we would not be able to conclude that Nixon is either a pacificist or that he is

[2]While the Tweety Triangle usually arises with ISA and ISNOTA links, it can just as easily arise with feature inheritance. Consider a case where the bird concept provides a value yes to a feature such as FLIER with the penguin concept provides the contradictory value no for the FLIER feature.

not a pacifist.

Even with what we take to be a sensible definition of default unification, we see that our problems are far from over when it comes to designing a lexical template or inheritance system that is defined in terms of default unification. We present a standard definition of templatic inheritance with defaults which suffers from a sensitivity to order that overrides even the specificity ordering induced on the templates. Templates are also not very satisfying in that constraints ¿from a single template are incorporated piecemeal and interleaved with information from inherited templates rather than being taken as a unified whole. In particular, the standard approach to templates emphasizes user presentation order by employing a completely depth-first strategy to unfold template definitions. For instance, a default feature value which was specified to hold in a template definition before a strict value would cause a conflict in that the default is added if it is consistent with the information already inherited up to the point at which the default is found without considering what strict information might come later that might override it.

After considering templatic inheritance, we consider an approach to default inheritance which inherits information from more specific sources before information from more general sources. We allow information attached to a concept in an inheritance hierarchy to be marked as either strict or default. Strict information is all inherited and taken to override any default information. In particular, strict information on a superconcept overrides default information on a subconcept. After carrying out strict inheritance, default information is inherited by default unification. The order in which default structures are unified is based on the resolution strategy of the Common Lisp Object System, an idea first considered by Russell *et al.* (1990, this volume). But even with this approach, there is residual order-sensitivity when inconsistencies arise from a combination of information from sources neither of which is more specific than the other. In these cases, user presentation order is used to resolve conflicts with results that may be surprising.

2 Feature Structures

In this section we review the basics of feature structures, including subsumption (information entailment) and unification (information conjunction). We only consider the most straightforward PATR-style feature structures. The definition of defaults that we present later is sufficiently abstract that it can be applied to more sophisticated notions of feature structure such as those presented by Aït-Kaci (1986), Moshier (1988, Moshier and Rounds 1987) and Carpenter (1990).

Feature structures are a means of representing information about the value of an object's attributes or features. To get off the ground, we assume a finite set Feat of *basic features*

and a finite set Atom of *atomic values*. We display features such as NUM, VFORM, SUBCAT in small capitals and atoms such as **sing**, **present-participle** and **noun** in bold face. A feature structure is either basic, in which case it must be one of the atomic values, or it is complex, in which case it provides values for a number of features. These values in turn are themselves taken to be either atomic or complex feature structures.

2.1 Feature Structures

Following Kasper and Rounds (1986, 1990), we define feature structures as a kind of labeled finite-state automata. The standard graph of such an automata brings out the relationship between feature structures and other frame-based knowledge representation systems. The only complications in the definition stems from the standard treatment of atomic values which must be such that there are no features defined for an atomic feature structure. We thus allow atomic values to label some (but not necessarily all) of the nodes in the graph from which there are no outgoing arcs.

Definition 1 (Feature Structure) *A* feature structures *is a tuple* $F = \langle Q, q_0, \delta, \alpha \rangle$ *where:*

- Q*: a finite set of* nodes
- $q_0 \in Q$*: the* root node
- $\delta : \mathsf{Feat} \times Q \rightarrow Q$ *: the partial* feature value *function*
- $\alpha : Q \rightarrow \mathsf{Atom}$ *: the partial* atomic value *function*

subject to the following constraints:[3]

- (Connectedness)
  every node must be reachable from the root (see below)
- (Atomic Values)
  only nodes without features can be atomic values so that if $\alpha(q)$ *is defined then* $\delta(f, q)$ *is undefined for every* $f \in \mathsf{Feat}$
- (Acyclic)
  the resulting graph is acyclic in that there is no path π *and non-empty path* π' *such that* $\delta(\pi, q_0) = \delta(\pi \cdot \pi', q_0)$ *(see below).*

[3]Sometimes the atomic value function α is required to be one to one, as in Pereira and Shieber (1984) and Rounds and Kasper (1986, 1990). Everything we say is compatible with such an assumption as with the more general notions of extensionality discussed in Carpenter (1990).

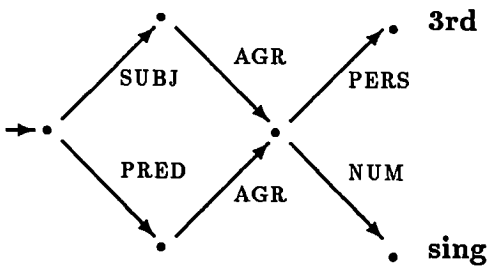


Figure 1: Feature Structure: Graph Notation

$$\begin{bmatrix} \text{SUBJ} : \begin{bmatrix} \text{AGR} : \boxed{4} \begin{bmatrix} \text{PERS} : \textbf{3rd} \\ \text{NUM} : \textbf{sing} \end{bmatrix} \end{bmatrix} \\ \text{PRED} : \begin{bmatrix} \text{AGR} : \boxed{4} \end{bmatrix} \end{bmatrix}$$

Figure 2: Feature Structure: AVM Notation

We provide two graphical depictions of the same feature structure in Figure 1 and Figure 2. The diagram in Figure 1 shows the graphical representation of feature structures in a standard finite-state transition diagram. This graphical notation soon become unwieldy, and so it is standardized in a record- and frame-like attribute-value matrix notation displayed in Figure 2. In the attribute-value matrix notation, each bracketed grouping corresponds to a node in the graphical representation. The tag $\boxed{4}$ in Figure 2 is taken to indicate structure sharing for subgraphs. Using this notation, substructures that are reachable by different sequences of features are displayed only once.

It is important to be able to represent sequences of features, which are called *paths*, so we take $\mathsf{Path} = \mathsf{Feat}^*$. We let ϵ be the empty path. We extend the definition of the transition function to paths by taking $\delta(\epsilon, q) = q$ and $\delta(f \cdot \pi, q) = \delta(\pi, \delta(f, q))$. We say that q' is *reachable* from q if there is a path π such that $q' = \delta(\pi, q)$. Thus it can be seen that the connectedness requirement amounts to requiring there to be a path of features from the root node to every other node in the feature structure.

It should also be noted that we have explicitly ruled out cyclic feature structures. Acyclicity is easily enforced by requiring that no path has a proper extension that leads to the same node as it does. Acyclic feature structures needlessly complicate results that are most easily presented in terms of acyclic feature structures. Our definitions below could be extended to allow cycles, but we do not do so here.

2.2 Subsumption

In this section we see how two feature structures can be compared to determine when one contains more information than the other. Our approach to subsumption is due to Moshier (1988, Moshier and Rounds 1987). In particular, for each feature structure, we associate a pair of sets which determine the path equivalences that hold and the atomic values assigned to paths by a feature structure. In particular, we let $\doteq_F$ be the equivalence relation induced between paths by the structure sharing in F and let $\mathcal{P}_F$ be the partial function induced by F which maps paths in F to atomic values.

Definition 2 (Abstract Feature Structure) *If $F = \langle Q, q_0, \delta, \alpha \rangle$ is a feature structure, we let $\equiv_F \subseteq \mathsf{Path} \times \mathsf{Path}$ and $\mathcal{P}_F : \mathsf{Path} \rightarrow \mathsf{Atom}$ be such that:*

- *(Path Equivalence)*
  $\pi \equiv_F \pi'$ if and only if $\delta(\pi, q_0) = \delta(\pi', q_0)$
- *(Path Value)*
  $\mathcal{P}_F(\pi) = \sigma$ if and only if $\alpha(\delta(\pi, q_0)) = \sigma$.

The pair $\langle \mathcal{P}, \equiv_F \rangle$ is called the abstract feature structure *corresponding to F.*

We see below that both the path equivalence relation and path value function are in fact finite for any feature structure, a useful consequence of eliminating cyclic feature structures which are always defined for infinitely many paths.

We say that a feature structure F subsumes another feature structure F' if and only if the information in F is contained in the information in F'; that is, if F' provides at least as much information about path values and structure sharing as F. Thus the abstract feature structure corresponding to F is sufficient to determine its information content and thus subsumption.

Definition 3 (Subsumption) *F subsumes F', written $F \sqsubseteq F'$, if and only if:*

- *$\pi \equiv_F \pi'$ implies $\pi \equiv_{F'} \pi'$*
- *$\mathcal{P}_F(\pi) = \sigma$ implies $\mathcal{P}_{F'}(\pi) = \sigma$*

Thus F subsumes F' if and only if every piece of information in F is contained in F'. Some examples of subsumption are as follows, where each left-hand-side properly subsumes the right-hand-side in that the converse subsumptions do not hold.

(1) $\begin{bmatrix} \mathrm{F} : \mathbf{a} \end{bmatrix} \sqsubseteq \begin{bmatrix} \mathrm{F} : \mathbf{a} \\ \mathrm{G} : \mathbf{b} \end{bmatrix}$

(2) $\left[\mathtt{H} : \left[\mathtt{F} : \mathtt{a}\right]\right] \sqsubseteq \begin{bmatrix} \mathtt{H} : \begin{bmatrix} \mathtt{F} : \mathtt{a} \\ \mathtt{G} : \mathtt{b} \end{bmatrix} \\ \mathtt{J} : \mathtt{c} \end{bmatrix}$

(3) $\begin{bmatrix} \mathtt{F} : \left[\mathtt{G} : \mathtt{a}\right] \\ \mathtt{G} : \left[\mathtt{G} : \mathtt{a}\right] \end{bmatrix} \sqsubseteq \begin{bmatrix} \mathtt{F} : \boxed{1}\left[\mathtt{G} : \mathtt{a}\right] \\ \mathtt{G} : \boxed{1} \end{bmatrix}$

We let $\perp$ be the single node feature structure with no atomic value assigned. Thus $\pi \equiv_\perp \pi'$ if and only if $\pi = \pi' = \epsilon$ and $\mathcal{P}(F)$ is undefined everywhere. Note that $\perp \sqsubseteq F$ for every feature structure F. We sometimes include $\perp$ in our attribute-value matrices to denote the lack of any known value.

We are not interested in the difference between feature structures which only vary in the identity of their nodes and if $F \sqsubseteq F'$ and $F' \sqsubseteq F$ then we say that F and F' are *alphabetic variants*. None of our definitions are sensitive to the difference between feature structures which are alphabetic variants.

2.3 Unification

Unification is an operation of information conjunction, and as such, can be naturally defined in terms of subsumption, a relation of information containment. The unification of two feature structures is defined to be the most general feature structure which contains all the information in both of the feature structures. In particular, we have the following standard definition.

Definition 4 (Unification) *The unification $F \sqcup F'$ of two feature structures F and F' is taken to be the least upper bound of F and F' in the collection of feature structures ordered by subsumption.*

Unpacking the definition, $F \sqcup F' = F''$ if and only if $F \sqsubseteq F''$, $F' \sqsubseteq F''$ and for every F''' such that $F \sqsubseteq F'''$ and $F' \sqsubseteq F'''$ we have $F'' \sqsubseteq F'''$. Moshier (1988) shows how to define unification directly in terms of abstract feature structures and also proves that the operation is well-defined up to alphabetic variance. Consistency is defined in terms of unification, where two feature structures F and F' are said to be *consistent* if their unification $F \sqcup F'$ is defined. Similarly, a finite set $\{F_1, \ldots, F_n\}$ is said to be consistent if $F_1 \sqcup \cdots \sqcup F_n$ is well-defined. Some examples of unification are given below.

(4) $\left[\mathtt{F} : \mathtt{a}\right] \sqcup \left[\mathtt{G} : \mathtt{b}\right] = \begin{bmatrix} \mathtt{F} : \mathtt{a} \\ \mathtt{G} : \mathtt{b} \end{bmatrix}$

(5) $\begin{bmatrix} \mathtt{F} : \boxed{1} \\ \mathtt{G} : \boxed{1} \end{bmatrix} \sqcup \begin{bmatrix} \mathtt{F} : \left[\mathtt{H} : \mathtt{a}\right] \\ \mathtt{G} : \left[\mathtt{J} : \mathtt{b}\right] \end{bmatrix} = \begin{bmatrix} \mathtt{F} : \boxed{1}\begin{bmatrix} \mathtt{H} : \mathtt{a} \\ \mathtt{J} : \mathtt{b} \end{bmatrix} \\ \mathtt{G} : \boxed{1} \end{bmatrix}$

(6) $\begin{bmatrix} \mathrm{F}: \begin{bmatrix} \mathrm{G}: \mathbf{a} \\ \mathrm{H}: \mathbf{b} \end{bmatrix} \\ \mathrm{J}: \begin{bmatrix} \mathrm{G}: \mathbf{a} \\ \mathrm{H}: \mathbf{c} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \mathrm{F}: \boxed{1} \\ \mathrm{J}: \boxed{1} \end{bmatrix}$ undefined

(7) $\begin{bmatrix} \mathrm{F}: \boxed{1}\begin{bmatrix} \mathrm{G}: \bot \end{bmatrix} \\ \mathrm{H}: \begin{bmatrix} \mathrm{J}: \boxed{1} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \mathrm{F}: \begin{bmatrix} \mathrm{G}: \boxed{2} \end{bmatrix} \\ \mathrm{H}: \boxed{2}\begin{bmatrix} \mathrm{J}: \bot \end{bmatrix} \end{bmatrix}$ undefined

(8) $\begin{bmatrix} \mathrm{F}: \mathbf{a} \end{bmatrix} \sqcup \begin{bmatrix} \mathrm{F}: \begin{bmatrix} \mathrm{G}: \mathbf{b} \end{bmatrix} \end{bmatrix}$ undefined

The notion of abstract feature structure yields a number of pleasant technical results. The most important of these results for our purposes is that every feature structure can be decomposed into a unification of atomic feature structures, where atomic feature structures are defined as follows:

Definition 5 (Atomic Feature Structure) *A feature structure is* atomic *if it is of one of the following two forms:*

- (Path Value)
  the feature structure contains a single path assigned to an atomic value.
- (Path Sharing)
  the feature structure contains only a pair of (possibly identical) paths which are shared.

Thus an atomic feature structure consists of either a single path being assigned an atomic value or the structure sharing between two (possibly identical) paths. The case where two identical paths are shared yields the information that the path is defined but its value is not known. Of course, these are just the components that make up our definition of $\equiv_F$ and $\mathcal{P}(F)$. We define a notation for extracting the atomic feature structures from a given feature structure:

(9) $At(F) = \{F' \sqsubseteq F \mid F' \text{ atomic}\}$

In general, we have the following results concerning the atomic decomposition of a feature structure.

Proposition 6

- $At(F)$ *is finite*
- $F = \bigsqcup At(F)$
- *if* $A \in At(F)$ *then if* $A = G \sqcup G'$ *then* $A = G$ *or* $A = G'$

- $F \sqsubseteq F'$ *if and only if* $At(F) \subseteq At(F')$

Proof: The fact that $At(F)$ is finite stems from the fact that there are only finitely many paths defined in any given feature structure.

The fact that $F = \bigsqcup At(F)$ arises from the fact that every piece of information in F is captured by some atomic feature structure in $At(F)$, so that $F \sqsubseteq \bigsqcup At(F)$. But since $G \sqsubseteq F$ for every $G \in At(F)$, we have $\bigsqcup At(F) \sqsubseteq F$ and hence $\bigsqcup At(F) = F$.

Neither information about a single path sharing or an atomic value can come from a feature structure which is not subsumed by an atomic one so that if $G \sqcup G' = F$ then either $F \sqsubseteq G$ or $F \sqsubseteq G'$, in which case the conditions that $G \neq F$ and $G' \neq F$ and $G \sqcup G' = F$ cannot be simultaneously satisfied.

The fact that subsumption reduces to inclusion of atomic constraints follows from the definition of subsumption. □

Of course this gives us a set-theoretic characterization of the information in feature structures as sets of atomic feature structures, a technique exploited originally by Pereira and Shieber (1984) and later by Moshier (1988). Taking this view of feature structures makes many of the following definitions easier to digest, particularly as we keep in mind the last of the above results which allows us to reduce subsumption to set inclusion between atomic feature structures.

2.4 Generalization

For our purposes, the order-theoretic dual to unification in which the greatest lower bound of two feature structures is computed is useful when it comes to define the skeptical form of default unification. The generalization of two feature structures is defined to be the most specific feature structure which contains only information found in both feature structures.

Definition 7 (Generalization) *The* generalization $F \sqcap F'$ *of two feature structures is defined to be their greatest lower bound in the subsumption ordering.*

Some examples of generalization are as follows:

(10) $\begin{bmatrix} \mathtt{F} : \mathtt{a} \\ \mathtt{G} : \mathtt{b} \end{bmatrix} \sqcap \begin{bmatrix} \mathtt{G} : \mathtt{b} \\ \mathtt{H} : \mathtt{c} \end{bmatrix} = \begin{bmatrix} \mathtt{G} : \mathtt{b} \end{bmatrix}$

(11) $\begin{bmatrix} \mathtt{F} : \boxed{1}\mathtt{a} \\ \mathtt{G} : \boxed{1} \end{bmatrix} \sqcap \begin{bmatrix} \mathtt{F} : \mathtt{c} \\ \mathtt{G} : \mathtt{b} \end{bmatrix} = \begin{bmatrix} \mathtt{F} : \bot \\ \mathtt{G} : \mathtt{b} \end{bmatrix}$

(12) $\begin{bmatrix} \mathtt{F} : \boxed{1} \begin{bmatrix} \mathtt{G} : \mathtt{a} \\ \mathtt{H} : \mathtt{b} \end{bmatrix} \\ \mathtt{J} : \boxed{1} \end{bmatrix} \sqcap \begin{bmatrix} \mathtt{F} : \boxed{2} \begin{bmatrix} \mathtt{G} : \mathtt{a} \\ \mathtt{H} : \mathtt{c} \end{bmatrix} \\ \mathtt{G} : \boxed{2} \end{bmatrix} = \begin{bmatrix} \mathtt{F} : \boxed{3} \begin{bmatrix} \mathtt{G} : \mathtt{a} \\ \mathtt{H} : \bot \end{bmatrix} \\ \mathtt{G} : \boxed{3} \end{bmatrix}$

It is important to note that while unification corresponds to conjunction, generalization does not correspond to disjunction (a detailed discussion of this point may be found in Pollard and Moshier 1990). In particular, the distributive law fails so that for instance:

$$(13)\quad (\left[\textsc{f} : \mathbf{a}\right] \sqcap \left[\textsc{f} : \mathbf{b}\right]) \sqcup \left[\textsc{f} : \mathbf{c}\right] = \left[\textsc{f} : \mathbf{c}\right] \neq (\left[\textsc{f} : \mathbf{a}\right] \sqcup \left[\textsc{f} : \mathbf{c}\right]) \sqcap (\left[\textsc{f} : \mathbf{b}\right] \sqcup \left[\textsc{f} : \mathbf{c}\right])$$

Thus it can be seen that generalization is more like information intersection than disjunction. In fact, Moshier (1988) showed that generalization could be defined by means of intersecting atomic values, so that we have the following.

Proposition 8 $At(F \sqcap G) = At(F) \cap At(G)$

In particular, the generalization of a finite set of feature structures is always well defined.

Other linguistic applications for generalization have been proposed by Karttunen (1984) and Pereira and Shieber (1984).

3 Default Unification

In this section, we present two alternative definitions of an operation of default unification, the purpose of which is to take a feature structure F, whose information is taken to be strict and combine it with a feature structure G, whose information is taken to be defeasible.

3.1 Credulous Default Unification

In the credulous approach to default reasoning, the idea is to maintain as much of the default information as is possible, as long as it does not conflict with the strict information. We base our definition of credulous default unification directly on this intuition. We should also note that an equivalent definition was independently proposed in slightly different terms by Calder (1991).

Definition 9 (Credulous Default Unification) *The result of credulously adding the default information in G to the strict information in F is given by:*

$$F \overset{<}{\sqcup}_c G = \{F \sqcup G' \mid G' \sqsubseteq G \textit{ is maximal such that } F \sqcup G' \textit{ is defined}\}$$

First off, it should be noted that the definition returns a set of feature structures rather than a unique value. This is because there may be more than one G' which is maximal such that it subsumes G and is consistent with F. For instance, consider the following example (taken out of context from Bouma 1990).

$$(14)\quad \left[\textsc{f} : \mathbf{a}\right] \overset{<}{\sqcup}_c \begin{bmatrix} \textsc{f} : \boxed{1}\mathbf{b} \\ \textsc{g} : \mathbf{b} \end{bmatrix} = \left\{ \begin{bmatrix} \textsc{f} : \mathbf{a} \\ \textsc{g} : \mathbf{b} \end{bmatrix}, \begin{bmatrix} \textsc{f} : \boxed{1}\mathbf{a} \\ \textsc{g} : \boxed{1} \end{bmatrix} \right\}$$

The non-determinism in (14) arises with the following choices of G' according to the definition of credulous default unification:

(15) $F = \begin{bmatrix} \mathrm{F} : \mathbf{a} \end{bmatrix}$ $\quad G = \begin{bmatrix} \mathrm{F} : \boxed{1}\mathbf{b} \\ \mathrm{G} : \boxed{1} \end{bmatrix}$

$G' = \begin{bmatrix} \mathrm{F} : \bot \\ \mathrm{G} : \mathbf{b} \end{bmatrix}$ $\quad G' = \begin{bmatrix} \mathrm{F} : \boxed{1} \\ \mathrm{F} : \boxed{1} \end{bmatrix}$

The credulous default unification operation is greedy in that it tries to maximize the amount of information it retains from the default structure. As a consequence, there may be more than one answer. This situation is common in other credulous default logics. Evans (1987) allowed such a credulous definition in his reconstruction of the feature specification default component of Generalized Phrase Structure Grammar (Gazdar *et al.* 1985). In general, there may be more than one result in $F \overset{<}{\sqcup}_c G$ if there are two pieces of information in G which are each compatible with F independently, but not when taken together. If the logic of feature structures were closed under disjunction, as in the logic presupposed by Calder (this volume), then such a non-deterministic result could be expressed as the disjunction of the set of values in $(F \overset{<}{\sqcup}_c G)$.

Now that we have a definition of default unification, albeit a credulous one, we can see that it satisfies a number of desiderata which have been previously put forward for notions of default unification. In particular, we have the following.

Proposition 10

- It is always well-defined.
  That is, $(F \overset{<}{\sqcup}_c G)$ *is always non-empty.*
- All strict information is preserved.
  If $H \in (F \overset{<}{\sqcup}_c G)$ *then* $F \sqsubseteq H$.
- It reduces to standard unification in case F and G are consistent.
  That is, $(F \overset{<}{\sqcup}_c G) = \{F \sqcup G\}$ *if* $F \sqcup G$ *is well-defined.*
- It is always finite.
  That is, $(F \overset{<}{\sqcup}_c G)$ *is a finite set.*

Proof: The fact that $(F \overset{<}{\sqcup}_c G)$ is always non-emptpy arises from the fact that $\bot \sqsubseteq G$ is such that $F \sqcup \bot = F$ is defined.

All strict information is preserved as every result is expressed as a unification of the strict information with some additional information contained in G.

If $F \sqcup G$ is well-defined then $G' = G$ is the unique maximal feature structure such that $G' \sqsubseteq G$ and $F \sqcup G'$ is well-defined.

Finiteness derives from the fact that we can break down any feature structure into the join of a finite set of atoms in the information ordering. In computing $F \overset{<}{\sqcup}_c G$, any maximal $G' \sqsubseteq G$ that we keep to unify in with F is composed of the join of a finite subset of the atoms which make up G. □

It is significant to note that our definition of credulous default unification is not in any way based on the internal structure of feature structures, but is derived entirely from the information containment ordering among them. Such a definition would be applicable in any partial order where least upper bounds are defined for pairs of bounded elements.

While it is not immediately obvious how to define nested applications, if we were to treat the set of results returned by credulous default unification as disjunctions in the standard way (that is, by taking the unification of a set of feature structures to distribute over the members of the set), then we could see that the operation is not associative. That is, we could find F, G and H such that $F \overset{<}{\sqcup}_c (G \overset{<}{\sqcup}_c H) \neq (F \overset{<}{\sqcup}_c G) \overset{<}{\sqcup}_c H$. We take up this lack of associativity when we consider the skeptical notion of default unification and consider how default unification can be integrated into a lexical knowledge representation system with inheritance.

It is rather difficult to compare our notion of credulous default unification to other proposals for default unification as our operation returns multiple answers, while previous definitions have assumed a unique result (with the notable exception of Evans (1987)).

3.2 Skeptical Default Unification

Now that we have a notion of credulous default unification which tries to maintain as much information from the default feature structure as possible, we turn our attention to a more skeptical definition which attempts to only maintain default information which is not in any way conflicted. To do this, it suffices to simply generalize the set of feature structures which results from credulous unification.

Definition 11 (Skeptical Default Unification) $F \overset{<}{\sqcup}_s G = \sqcap(F \overset{<}{\sqcup}_c G)$

In particular, the definition of skeptical default unification leads to a unique result. The only default information that remains is that which is found in every credulous result. Consider the following example of skeptical unification:

$$(16) \quad \begin{bmatrix} \mathrm{F} : \mathbf{a} \end{bmatrix} \overset{<}{\sqcup}_s \begin{bmatrix} \mathrm{F} : \boxed{1}\mathbf{b} \\ \mathrm{G} : \boxed{1} \\ \mathrm{H} : \mathbf{c} \end{bmatrix} = \sqcap \left(\begin{bmatrix} \mathrm{F} : \mathbf{a} \end{bmatrix} \overset{<}{\sqcup}_c \begin{bmatrix} \mathrm{F} : \boxed{1}\mathbf{b} \\ \mathrm{G} : \boxed{1} \\ \mathrm{H} : \mathbf{c} \end{bmatrix} \right)$$

$$= \sqcap \left\{ \begin{bmatrix} \mathrm{F} : \mathbf{a} \\ \mathrm{G} : \mathbf{b} \\ \mathrm{H} : \mathbf{c} \end{bmatrix}, \begin{bmatrix} \mathrm{F} : \boxed{1}\mathbf{a} \\ \mathrm{G} : \boxed{1} \\ \mathrm{H} : \mathbf{c} \end{bmatrix} \right\} = \begin{bmatrix} \mathrm{F} : \mathbf{a} \\ \mathrm{G} : \bot \\ \mathrm{H} : \mathbf{c} \end{bmatrix}$$

Thus we can see that all of the information that is contained in both of the credulous extensions is maintained in the skeptical result. In particular, since both credulous results are defined for the path G, the result is defined for the path G, but since they provide conflicting atomic values, no value is retained in the result. On the other hand, the fact that the H feature has value **c** is maintained in the result as it is found in every credulous extension.

This example shows how in general atomic information from the default feature structure is only maintained in the result of skeptical unification if it does not cause a conflict when combined with any other information drawn from either the strict or default feature structure. In fact, skeptical unification could be defined in these terms without going through the definition of credulous default unification.

Note that our notion of default unification is distinct from that proposed by Copestake *et al.* (Copestake, this volume), as their unification strategy gives preference to path sharing over path values during default unification. Thus the fact that F and G were shared would be kept in the result, while the information that their values were **b** would be discarded. Also note that our method of default unification cannot be compared to the PATR-II notion of overwriting (Shieber 1986) as overwriting only applies to single atomic descriptions and not to entire feature structures. We come back to a notion of overwriting when we consider templates below. Our notion of default unification reduces to Kaplan's (1987) sketch of an operation of priority union, under the strong assumption that both feature structures contain no structure sharing. Priority union was simply not defined in the case where either the default or strict feature structure contained structure sharing. Bouma's (1990) approach to structure sharing was the most difficult aspect of his rather intricate definition of default unification. We compare our notion of skeptical default unification with Bouma's operation below.

Not surprisingly, the skeptical notion of default unification maintains the desiderata satisfied by credulous unification.

Proposition 12

- $F \overset{<}{\sqcup}_s G$ is always well defined (and produces a unique result).
- Strict information is preserved.
  That is, $F \sqsubseteq (F \overset{<}{\sqcup}_s G)$
- It reduces to standard unification if F and G are consistent.
  That is, $(F \overset{<}{\sqcup}_s G) = (F \sqcup G)$ *if* $F \sqcup G$ *is well defined.*

Proof: That $F \overset{<}{\sqcup}_s G$ is always well-defined follows from the fact that $F \overset{<}{\sqcup}_c G$ is always non-empty and finite non-empty meets are always well-defined.

The fact that $F \sqsubseteq (F \overset{<}{\sqcup}_s G) = \sqcap(F \overset{<}{\sqcup}_c G)$ follows from the fact that $F \sqsubseteq H$ for every $H \in (F \overset{<}{\sqcup}_c G)$.

If F and G are consistent then $F \overset{<}{\sqcup}_c G = \{F \sqcup G\}$ and hence $F \overset{<}{\sqcup}_s G = \sqcap(F \overset{<}{\sqcup}_c G) = \sqcap\{F \sqcup G\} = F \sqcup G$. □

We now consider some additional examples of skeptical unification. In each case, we have shown the credulous feature structures that would be defined on the way to computing the result of skeptical default unification. First consider the case where there is a three-way sharing in the default structure which is incompatible with two values in the original structure.

$$(17)\quad \begin{bmatrix} \mathtt{F} : \mathbf{a} \\ \mathtt{H} : \mathbf{b} \end{bmatrix} \overset{<}{\sqcup}_s \begin{bmatrix} \mathtt{F} : \boxed{1} \\ \mathtt{G} : \boxed{1} \\ \mathtt{H} : \boxed{1} \end{bmatrix} = \sqcap \left\{ \begin{bmatrix} \mathtt{F} : \mathbf{a} \\ \mathtt{H} : \mathbf{b} \end{bmatrix} \sqcup \begin{bmatrix} \mathtt{F} : \boxed{1} \\ \mathtt{G} : \boxed{1} \\ \mathtt{H} : \bot \end{bmatrix}, \begin{bmatrix} \mathtt{F} : \mathbf{a} \\ \mathtt{H} : \mathbf{b} \end{bmatrix} \sqcup \begin{bmatrix} \mathtt{F} : \bot \\ \mathtt{G} : \boxed{1} \\ \mathtt{H} : \boxed{1} \end{bmatrix} \right\}$$

$$= \sqcap \left\{ \begin{bmatrix} \mathtt{F} : \boxed{1}\mathbf{a} \\ \mathtt{G} : \boxed{1} \\ \mathtt{H} : \mathbf{b} \end{bmatrix}, \begin{bmatrix} \mathtt{F} : \mathbf{a} \\ \mathtt{G} : \boxed{1}\mathbf{b} \\ \mathtt{H} : \boxed{1} \end{bmatrix} \right\} = \begin{bmatrix} \mathtt{F} : \mathbf{a} \\ \mathtt{G} : \bot \\ \mathtt{H} : \mathbf{b} \end{bmatrix}$$

Adding the fact that all of the path values are **b** in the default structure does not change the result.

$$(18)\quad \begin{bmatrix} \mathtt{F} : \mathbf{a} \\ \mathtt{H} : \mathbf{b} \end{bmatrix} \overset{<}{\sqcup}_s \begin{bmatrix} \mathtt{F} : \boxed{1}\mathbf{b} \\ \mathtt{G} : \boxed{1} \\ \mathtt{H} : \boxed{1} \end{bmatrix} = \begin{bmatrix} \mathtt{F} : \mathbf{a} \\ \mathtt{G} : \bot \\ \mathtt{H} : \mathbf{b} \end{bmatrix}$$

It is interesting to compare the previous examples to the following one.

$$(19)\quad \begin{bmatrix} \mathtt{F} : \mathbf{a} \\ \mathtt{H} : \mathbf{b} \end{bmatrix} \overset{<}{\sqcup}_s \begin{bmatrix} \mathtt{F} : \boxed{1} \\ \mathtt{G} : \boxed{1} \\ \mathtt{H} : \boxed{1} \\ \mathtt{J} : \boxed{1} \end{bmatrix} = \begin{bmatrix} \mathtt{F} : \mathbf{a} \\ \mathtt{G} : \bot \\ \mathtt{H} : \mathbf{b} \\ \mathtt{J} : \bot \end{bmatrix}$$

This last example is interesting in that it provides a result which is distinctly different from the result given by Bouma's (1990) definition of default unification in which the sharing between G and J would be maintained in the result after the paths F and H were removed from the sharing in the default feature structure due to the fact that they might cause conflicts. In this case, Bouma's default unification returns a more specific value than our skeptical notion (though of course, some of the credulous results are as specific as Bouma's result, as Bouma correctly never takes more than a maximally consistent subset of the default information). But consider the following case, for which our notion of skeptical default unification returns a result more specific than Bouma's notion:

$$(20)\quad \begin{bmatrix} \mathtt{F} : \boxed{1} \\ \mathtt{G} : \boxed{1} \end{bmatrix} \overset{<}{\sqcup}_s \begin{bmatrix} \mathtt{F} : \mathbf{a} \end{bmatrix} = \begin{bmatrix} \mathtt{F} : \boxed{1}\mathbf{a} \\ \mathtt{G} : \boxed{1} \end{bmatrix}$$

This example illustrates how Bouma's definition does not meet the desideratum that default unification reduce to standard unification if the default information is wholly consistent with the strict information. Bouma simply discards the information that F's value is **a**, as this value is a *potential* conflict according to Bouma's definition because the strict feature structure provides information about F's value, namely that it is shared with G's value. Now consider the following two examples, which demonstrate an interesting contrast.

$$(21)\quad \left[\mathrm{F}:\left[\mathrm{H}:\mathbf{c}\right]\right] \overleftarrow{\sqcup}_s \left[\begin{array}{l}\mathrm{F}:\boxed{1}\left[\begin{array}{l}\mathrm{H}:\mathbf{a}\\ \mathrm{J}:\mathbf{b}\end{array}\right]\\ \mathrm{G}:\boxed{1}\end{array}\right] = \sqcap\left\{\left[\mathrm{F}:\left[\mathrm{H}:\mathbf{c}\right]\right] \sqcup \left[\begin{array}{l}\mathrm{F}:\left[\begin{array}{l}\mathrm{H}:\bot\\ \mathrm{J}:\boxed{2}\mathbf{b}\end{array}\right]\\ \mathrm{G}:\left[\begin{array}{l}\mathrm{H}:\mathbf{a}\\ \mathrm{J}:\boxed{2}\end{array}\right]\end{array}\right]\right\}$$

$$= \left[\begin{array}{l}\mathrm{F}:\left[\begin{array}{l}\mathrm{H}:\mathbf{c}\\ \mathrm{J}:\boxed{2}\mathbf{b}\end{array}\right]\\ \mathrm{G}:\left[\begin{array}{l}\mathrm{H}:\mathbf{a}\\ \mathrm{J}:\boxed{2}\end{array}\right]\end{array}\right]$$

$$(22)\quad \left[\mathrm{F}:\left[\mathrm{H}:\mathbf{c}\right]\right] \overleftarrow{\sqcup}_s \left[\begin{array}{l}\mathrm{F}:\boxed{1}\left[\begin{array}{l}\mathrm{H}:\mathbf{a}\\ \mathrm{J}:\mathbf{b}\\ \mathrm{K}:\bot\end{array}\right]\\ \mathrm{G}:\boxed{1}\end{array}\right]$$

$$= \sqcap\left\{\left[\mathrm{F}:\left[\mathrm{H}:\mathbf{c}\right]\right] \sqcup \left[\begin{array}{l}\mathrm{F}:\left[\begin{array}{l}\mathrm{H}:\bot\\ \mathrm{J}:\boxed{2}\mathbf{b}\\ \mathrm{K}:\boxed{3}\bot\end{array}\right]\\ \mathrm{G}:\left[\begin{array}{l}\mathrm{H}:\mathbf{a}\\ \mathrm{J}:\boxed{2}\\ \mathrm{K}:\boxed{3}\end{array}\right]\end{array}\right]\right\} = \left[\begin{array}{l}\mathrm{F}:\left[\begin{array}{l}\mathrm{H}:\mathbf{c}\\ \mathrm{J}:\boxed{2}\mathbf{b}\\ \mathrm{K}:\boxed{3}\bot\end{array}\right]\\ \mathrm{G}:\left[\begin{array}{l}\mathrm{H}:\mathbf{a}\\ \mathrm{J}:\boxed{2}\\ \mathrm{K}:\boxed{3}\end{array}\right]\end{array}\right]$$

In these two examples we see that whether or not a feature is specified as being defined in the default information has a strong bearing on whether it comes out shared in the result. This is because as much of the sharing from the default structure as is consistent with the strict structure is kept in the result. In particular, the fact that the path $\mathrm{F}\cdot\mathrm{K}$ is defined in the second example above means that the sharing between it and $\mathrm{G}\cdot\mathrm{K}$ which is induced by the sharing between F and K is kept in the result. It is interesting to note that Bouma's definition of default unification first relaxes the constraints in the default feature structure in part by replacing every path sharing which is internal to a feature structure with a collection of structure sharing constraints which hold only between terminal nodes in the feature structure. This leads to structures much like those found in Prolog terms, where the only sharing allowed is between variables. To accomplish such a relaxation, Bouma

```
<lexicon> ::=  (<lex-entry>)* (<template-def>)*

<lex-entry> ::= <expr> lex <desc>

<template-def> ::= <template-name> template <desc>

<desc> ::= ( <template-name> | <at-desc> | (? <at-desc>) | (! <at-desc>) )*

<at-desc> ::= <path> : <atom>
            | <path> == <path>
```

Figure 3: Templatic Lexical Specification Language

recursively replaces every internal path sharing such as that between F and G above with the collection of sharings between $F \cdot L$ and $G \cdot L$ for every feature L (as long as the strict feature structure is not defined for any of the new paths). It is surprising and probably undesirable for the two examples above to provide different results. This unwanted behavior could be eliminated by enforcing a type discipline on feature structures so that they are closed like first-order terms in that they provide values for every feature for which they are appropriate (such systems have been studied by Calder (1987, Moens *et al.* 1989) and Carpenter (1990)). The default feature structure in the first example above is not closed in this sense and would thus never arise.

4 Templatic Inheritance

In this section we discuss a notion of templates which can be thought of as abbreviatory conventions for lexical entries. Our definitions closely follow those of PATR-II (Shieber *et al.* 1983). We begin by defining a specification language for templates and lexical entries. After this, we show how feature structures are associated with basic expressions according to such a specification.

We present the syntax of the specification language in BNF in Figure 3. The * in the definitions is taken to be of the Kleene variety and denotes arbitrarily many occurrences of the pattern it is attached to, while | is taken to indicate disjunction and parentheses are used for grouping. We take the types `<expr>` of basic expressions, `<path>` of paths (sequences of features), `<template-name>` of template names, and `<atom>` of atomic values to be given. Typologically, we use N for template names, a for atoms, Φ for descriptions, ϕ for atomic descriptions and π for paths.

There are a number significant points to note about this definition. First, the operators ? and ! are used to mark the fact that information is to be interpreted by overwriting and by default respectively. But note that these operators can only apply to atomic descriptions. Secondly, note that templates can be included in descriptions, so that in general, one template may be defined in terms of others. It is this facility that allows such a system to be useful. The templates can be arranged hierarchically according to which are defined in terms of the others. Note that a lexicon is itself simply a sequence of lexical entries where the category of an expression is described and of template definitions which associate template names with descriptions. For this kind of definition to get off the ground, the templates cannot be recursive in such a way that expanding a template T involves expanding T' and vice-versa (in other words, the induced hierarchy must be a partial ordering).

We can now use a highly simplified case of default unification to provide a definition of the feature structures that are associated with any given lexical entry. In computing this feature structure, we follow the definition given by the user depth-first in the linear order in which it is presented. The order that we include atomic constraints depends on the order in which they are encountered in a depth-first expansion of template definitions. In order to make our definition functional, we adopt the skeptical form of default unification. It would not be difficult to change it to a relational definition in which credulous unification was used. Similarly, disjunction could be incorporated into the description language in the same way. In the following definition, we take $MGSat(\phi)$ to be the minimal feature structure which satisfies the description ϕ, a result which was proved to exist (in a much more general setting) by Kasper and Rounds (1986, 1990).

Definition 13 (Templatic Inheritance) *The lexical entry F associated with an expression e where there is a lexical entry e* `lex` Φ *is given by the result of the function $F = Add(\bot, \Phi)$ where Add is defined according to the following clauses:*

- $Add(F, ()) = F$
- $Add(F, \phi \cdot \Phi) = Add(Add(F, \phi), \Phi)$
- $Add(F, !\phi) = MGSat(\phi) \overset{<}{\sqcup}_s F$
- $Add(F, ?\phi) = F \overset{<}{\sqcup}_s MGSat(\phi)$
- $Add(F, \phi) = F \sqcup MGSat(\phi)$
- $Add(F, N) = add(F, \Phi)$ *if* (N `template` Φ) *is a template definition*

The immediate thing to note about this definition is that everything is evaluated depth-first according to how it is specified in the lexicon description. When it comes to adding in a default $?\phi$, we simply take the most general satisfier of the atomic description ϕ and

```
A template (B,C)
B template (E)
C template (D)
D template (E)
E template ().
```

Figure 4: Path Length Sensitivity Example

skeptically unify it into what we have so far. Similarly, if we add an overwriting description $!\phi$ we unify what we have so far by default into the most general satisfier of ϕ, which we treat as providing strict information. Information that is neither overwriting or default information is simply unified into the result. Note that this is the only way that conflicts might arise from such a specification as default unification always succeeds.

It is important to note that since we only ever take the most general satisfiers of atomic descriptions when computing *Add*, the definition $F \hat{\sqcup}_s MGSat(\phi)$ for adding the default information ϕ reduces to simply adding the information in ϕ if it is consistent with F. That is, defaults come all or nothing and are evaluated one at a time in this system. Of course, for overwriting, even though the overwriting feature structure will be atomic, it might lead to interesting behavior in evaluating $MGSat(\phi) \hat{\sqcup}_s F$, as F is not restricted to being atomic. For instance, the kind of behavior we saw in (16) would result from evaluating a description such as the following:

(23) `(f:b, f==g, h:c, !(f:a))`

With default and strict information, the *Add* is increasing so that $F \sqsubseteq Add(F, ?\phi)$ and $F \sqsubseteq Add(F, \phi)$, but we do not in general have $F \sqsubseteq Add(F, !\phi)$ in the overwriting case, as information in ϕ can override information in F.

The most significant aspect of our definition of templatic inheritance is that descriptions are evaluated depth-first and left-to-right, including template expansions. Simply consider the template specifications in Figure 4 (it is not important what other information they contain). In the case of the specification in Figure 4, the information from template `E` is inherited by `A` before information from either `C` or `D` due to the implicit depth-first evaluation order. While it would be easy to change the definition to make it breadth-first rather than depth-first, we would still get odd results that are sensitive to path length rather than the specificity ordering. Consider evaluating the template specifications in Figure 4 in a breadth-first manner. In this case, we would evaluate the template `E` before or at the same time as the template `D` because we get to `E` after two steps from `A` through `B` but only after three steps when inherited through `C` and `D`. `D` itself is at a depth of two from `A`. Ideally,

we would want to get the information from D before the information in E as D is defined in terms of E. We see how to get around some of these problems in the next section on default inheritance which uses a mixed breadth-first and depth-first approach to ordering default information.

Other kinds of order-sensitivity in these definitions stem from the fact that a template may sandwich a call to another template in between some basic descriptions. In this case as well as in others like it, the template is evaluated in the position in which it is found, thus allowing it to override the default information which comes after it and be overridden by any information which might have been included before it. Of course, templates could always be placed before or after other information as a matter of style or by syntactic restriction on the description language. But one possible benefit to this kind of ordering system is that it is rather flexible in the opportunities it affords to the lexical designer and is at the same time straightforward to debug as the evaluation function is itself simple to follow.

Uses of this kind of templatic inheritance has been discussed in the PATR-II literature (see Shieber 1986) and in the case of defaults, by Bouma (1990b). Our presentation extends the PATR-II specification somewhat by allowing default unification, while PATR-II itself contains overwriting which is broadly similar to what we have defined here. Bouma (1990b) extends the PATR-II template system by generalizing the description logic so that it is similar to the conjunctive portion of the Rounds/Kasper logic and allowing templates to apply at arbitrary levels of description, a move similar to that made in Carpenter's (1989) implementation of a PATR-II system with Rounds/Kasper-style syntax (including embedded templates, but not overwriting). In particular, if we allowed arbitrary conjunctions of descriptions such as $\Phi \wedge \Psi$ then we would simply take:

(24) $Add(F, \Phi \wedge \Psi) = Add(Add(F, \Phi), \Psi)$.

Similarly, if we allowed descriptions of the form $\pi : \Phi$ where Φ is now an arbitrary description (possibly incorporating a template) then the most sensible choice for evaluating the defaults seems to be to take:

(25) $Add(F, \pi : \Phi) = F[\pi := Add(F@\pi, \Phi)]$

where $F@\pi$ is the value of F for the path π and where we take $F[\pi := G]$ to be the result of replacing F's π value with G. Such a brutal operation is necessary to account for the effects of overwriting. Without overwriting, the definition could simply add the information in $Add(F@\pi, \Phi)$ to F's π value by unification. In particular, this example recursively adds the constraint Φ from $\pi : \Phi$ to the value of F at π rather than computing the most general satisfier of Φ all on its own and unifying that value into F.

What is not apparent in trying to extend default templates to the full Rounds/Kasper-style logical language is how to apply the default or overwriting operators themselves to

entire descriptions. That is, what to do with something like $?(\Phi \wedge \Psi)$ is rather open. It could be taken to evaluate $\Phi \wedge \Psi$ and then treat the result as a default, as in:

(26) $Add(F, ?(\Phi \wedge \Psi)) = F \hat{\sqcup}_s Add(\bot, \Phi \wedge \Psi)$.

Of course this would provide a different result than assuming that default specifications distribute as in:

(27) $Add(F, ?(\Phi \wedge \Psi)) = Add(F, ?\Phi \wedge ?\Psi)$

Of course, the same sorts of differences would be found with allowing the overwriting operator to apply to complex descriptions. A description language which applies default and overwriting operators to complex descriptions but is evaluated left to right would be misleading in cases such as $?\Phi \wedge !\Psi$ which would give priority to information in Φ over information in Ψ. As things stand in our PATR-II-like system, only fully distributed defaults or overwritings are allowed simply because the default and overwriting operators can only apply to atomic descriptions.

If we were willing to allow disjunctions of the form $\Phi \vee \Psi$, then we could make *Add* non-deterministic by allowing $Add(F, \Phi \vee \Psi)$ to return either $Add(F, \Phi)$ or $Add(F, \Psi)$. Of course, if we make *Add* non-deterministic, then we could employ credulous unification in the obvious way.

5 Default Inheritance

In this section, we turn our attention to the specification of a lexical inheritance system based explicitly on a notion of inheritance hierarchy which uses default unification to add more specific default information before more general default information. Unfortunately, as we have already mentioned, we are not able to excise all of the remaining order-sensitivity employing this method. Following Russell *et al.* (1990, this volume), we rely on the Common Lisp Object System method of resolving conflicts between inconsistent information inherited from orthogonal sources by visiting the sources in a depth-first order (see Steele (1990) for details concerning the Common Lisp Object System).

As with the definition of templatic inheritance, the definition of default inheritance is based on a lexical specification. We present the language in which such specifications are expressed in BNF in Figure 5. Note the differences between the default inheritance specification language and the templatic inheritance specification language. Most significantly, the default inheritance system specifies a set of concept definitions rather than template definitions. A concept definition supplies three pieces of information: the superconcepts from which this concept inherits, the strict information attached to the concept and the default information about the concept. Both strict and default information is presented as

```
<lexicon> ::= (<lex_entry>)* (<conc-def>)*

<conc-def> ::=  (<conc> isa (<conc>*) strict <desc> default <desc>)

<lex-entry> ::= <expr> lex (<conc>)*

<desc> ::= (<atomic-desc>)*

<at-desc> ::= <path> : <atom>
            | <path> == <path>
```

Figure 5: Default Inheritance Specification Language

a simple description, thus making the distinction between default and strict information on a concept by concept rather than an atomic description by atomic description basis. One benefit of this approach, which we have not exploited here, is that a more general description logic could be provided; we use the simple PATR-II language, while a language such as that provided by Rounds and Kasper (1986, 1990) is slightly nicer to work with. Another thing to notice about the definition is that we allow both strict and default information to be attached to a given concept. According to our inheritance scheme, strict information is always inherited, while default information may be overridden.

We turn our attention now to specifying the categories associated with basic expressions by the lexicon. As we have said before, we are not able to totally eliminate the order dependence in inheritance and is thus sensitive to the way in which the inheritance between concepts is ordered in the description as well as the partial ordering it induces. But we do follow a specificity-based approach in taking default information in more specific concepts to override default information attached to more general concepts. But in cases where default information from orthogonal concepts is in conflict, we choose to maintain the information that comes from the first concept visited in a depth-first traversal of all of the concepts which are inherited by a lexical entry. To make matters more precise, we define for any sequence $(C_1, \ldots, C_n)$ of concepts, the depth-first ordering $(D_1, \ldots, D_m)$ of concepts from which they inherit information according to the inheritance hierarchy specification. This depth-first ordering is then used to induce the final inheritance ordering.

Definition 14 (Depth-first Ordering) *The depth first ordering $DF((C_1, \ldots, C_n))$ of the concepts inherited by an ordered sequence $(C_1, \ldots, C_n)$ of concepts is given by:*

- $DF(C) = C \cdot DF(C_1) \cdot \cdots \cdot DF(C_n)$

```
A isa (B, F)    E isa (D)
B isa (C, E)    F isa (E,G)
C isa (D)       G isa ()
D isa ()

df(A) = A df(B) df(F)
      = A B df(C,E) F df(E,G)
      = A B df(C) df(E) F df(E) df(G)
      = A B C df(D) E df(D) F E df(D) G
      = A B C D E D F E D G
```

Figure 6: Depth-first Ordering

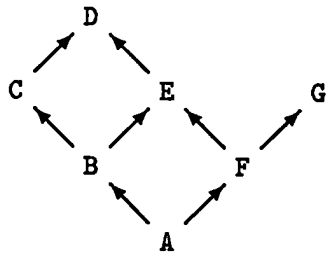


Figure 7: Inheritance Hierarchy Diagram

if C `isa` $(C_1, \ldots, C_n)$ *is part of the lexical specification*

- $DF((C_1, C_2, ..., C_n)) = DF(C_1) \cdot DF(C_2) \cdot \cdots \cdot DF(C_n)$

We provide a simple example in Figure 6, which illustrates many of the properties of the depth-first ordering. A graphical depiction of the hierarchy in Figure 6 is given in Figure 7.

As can be seen in Figure 6, in the case of multiple inheritance, the depth-first ordering contains more than one occurrence of some concepts. For instance, `D` appears three times in the final depth-first ordering in Figure 6 as there are three paths from `A` to `D` specified in the inheritance hierarchy. Multiple occurrences in the depth-first ordering, while easily eliminable with a more sophisticated definition, do not pose any problems as our final ordering is only sensitive to the first occurrence of a concept in the depth-first ordering.

Note that we can use the depth-first ordering to define the specificity ordering.

```
  TS(A B C D E D F E D G)
= A TS(B C D E D F E D G)
= A B TS(C D E D F E D G)
= A B C TS(D E D F E D G)
= A B C F TS(D E D E D G)
= A B C F E TS(D D D G)
= A B C F E D TS(G)
= A B C F E D G
```

Figure 8: Topological Ordering of Concepts

Definition 15 (Specificity) *A concept C is equal or more* specific *than C' if and only if C' is an element of the sequence $DF(C)$.*

The reason this definition makes sense is that $DF(C)$ is a list which is closed under superconcepts given in the lexical specification; that is, if a concept C appears in $DF((C_1, \ldots, C_n))$ and C' is specified as a superconcept of C, then C' appears in $DF((C_1, \ldots, C_n))$.

We use the depth-first sequence of subconcepts to perform a topological sort of the concepts inherited by any given sequence of concepts. More specifically, we form the final inheritance ordering by ordering elements from the front of the depth-first ordering as soon as all of their subconcepts are included.

Definition 16 (Topological Ordering) $TS(C_1, \ldots, C_n) = TS_2(DF(C_1, \ldots, C_n))$ *where:*

- $TS_2(()) = ()$
- $TS_2(D_1, \ldots, D_m) = D_k \cdot TS_2((D_1, \ldots, D_m) - D_k)$ *if k is minimal such that there is no $D_j \neq D_k$ such that $1 \leq j \leq m$ and D_j is more specific than D_k.*

This definition is actually simpler than it appears as we are really doing nothing more than taking the depth-first ordering of concepts inherited by $(C_1, \ldots, C_n)$ and then successively choosing the first element in the depth-first order such that all of its subconcepts have already been chosen. We continue the example in Figure 6 in Figure 8. It can be seen in Figure 8 that the topological ordering respects the specificity ordering in that every concept occurs on the list before any of its superconcepts. On the other hand, it can also be seen that the depth-first ordering is used to order concepts neither of which is more specific than the other. For instance, the fact that `F` is more general than `D` means that `D` is ordered before `F`. But considering `B` and `F`, neither of which is more specific than the other, we take `B` before `F` as `B` shows up before `F` in the depth-first ordering. `C` shows up before `G` for the same reason.

Note that we only consider subconcepts induced along the path from the concepts being inherited from and do not perform a global depth-first ordering of the entire graph, which provides different results in some cases. In particular, it may turn out that a concept C is inherited before an orthogonal concept C' in some cases and after C' in others if the specification is not uniform as to the order in which subconcepts are introduced. For instance, a specification that puts `A isa (C,D)` and `B isa (D,C)` would have the concept `C` inherited before `D` for `A`, but after `D` for `B`.

Now that we have defined the order in which concepts are visited for inheritance, it is a straightforward matter to characterize the linguistic categories (represented by feature structures) which are assigned by the lexicon. We first inherit all of the strict information and then inherit default information one concept at a time following the topological ordering of concepts.

Definition 17 (Default Inheritance) *A lexical entry* (e `lex` $C_1,\ldots,C_n$) *assigns the category*

$$(\cdots(((F_1 \sqcup \cdots \sqcup F_m) \overset{<}{\sqcup}_s G_1) \overset{<}{\sqcup}_s G_2) \overset{<}{\sqcup}_s \cdots \overset{<}{\sqcup}_s G_{m-1}) \overset{<}{\sqcup}_s G_m$$

to e, where $TS(C_1,\ldots,C_n) = D_1,\ldots,D_m$ and where F_i is the feature structure containing strict information and G_i is the feature structure holding default information attached to the concept D_i.

For instance, in the hierarchy in Figure 6, we first combine all of the strict information and then successively skeptically unify in the default information from the inherited concepts in the order that they appear in the topological ordering. In particular, strict information from `D` overrides default information, even if it appears directly on `A`. Since we take the topological ordering, default information on `C` would override default information either on its superconcept `D` or on orthogonal concepts visited later in the topological ordering such as `F` or `G`. Obviously under this definition, information from more specific sources overrides information from more general sources.

We finally take up a point we promised to come back to concerning the (lack of) associativity of default unification. In particular, we have the following result:

$$(28)\quad \left(\left[\mathtt{F}:\mathbf{a}\right] \overset{<}{\sqcup}_s \begin{bmatrix}\mathtt{F}:\boxed{1}\\ \mathtt{G}:\boxed{1}\end{bmatrix}\right) \overset{<}{\sqcup}_s \left[\mathtt{G}:\mathbf{b}\right] = \begin{bmatrix}\mathtt{F}:\boxed{1}\mathbf{a}\\ \mathtt{G}:\boxed{1}\end{bmatrix} \overset{<}{\sqcup}_s \left[\mathtt{G}:\mathbf{b}\right] = \begin{bmatrix}\mathtt{F}:\boxed{1}\mathbf{a}\\ \mathtt{G}:\boxed{1}\end{bmatrix}$$

$$\neq \left[\mathtt{F}:\mathbf{a}\right] \overset{<}{\sqcup}_s \left(\begin{bmatrix}\mathtt{F}:\boxed{1}\\ \mathtt{G}:\boxed{1}\end{bmatrix} \overset{<}{\sqcup}_s \left[\mathtt{G}:\mathbf{b}\right]\right) = \left[\mathtt{F}:\mathbf{a}\right] \overset{<}{\sqcup}_s \begin{bmatrix}\mathtt{F}:\boxed{1}\mathbf{b}\\ \mathtt{G}:\boxed{1}\end{bmatrix} = \begin{bmatrix}\mathtt{F}:\mathbf{a}\\ \mathtt{B}:\bot\end{bmatrix}$$

This kind of contrast is well-known in the inheritance literature and corresponds to the distinction between performing inheritance top-down and bottom-up. Such contrasts arise

because default inheritance is rarely transitive. In particular, consider the following specification:

(29) `A isa (B) default f:a`
`B isa (C) default f == g`
`C isa () default g:b`

The feature structure inherited by `A` according to our definitions maintains the sharing because our definition proceeds bottom-up. If we had been proceeding top-down instead, we would first inherit all of the information relevant to `B` before adding the information from `B` to the information attached to `A`. In the top-down case, no sharing would be inherited by `A`. It seems intuitively at least, that the bottom-up definition is preferable as the fact that `f` is shared with `g` is presented at a more specific node than the fact that `g`'s value is `b`. In fact, it is generally much more straightforward to define a sensible notion of default inheritance bottom-up than it is to define inheritance in a top-down fashion.

6 Future Directions

While we have gone some distance in achieving a lexical organization that employs default information, we are still left with a distastefully degree of order sensitivity in the result. It remains to be seen whether a reasonable definition can be made that achieves complete order-neutrality and finds a sensible way to resolve orthogonal conflicts. Ideally, we would remain skeptical in the light of such conflicts, not choosing to favor information from one orthogonal source over another. On the other hand, working out such a definition is far from trivial. Of course, if no default information from orthogonal concepts ever arises, this is not a problem. Many systems such as Copestake *et al.* (this volume) and in the DATR system of Evans and Gazdar (1989, 1989b) require orthogonal information to always be compatible.

One subject we have not even broached, which also presents a significant challenge for future research is the characterization of recursive default constraints. In particular, we might want to state that the value of a path satisfies some concept definition. While such a system can be worked out even in the face of recursion in the case of purely strict information, the result being an interpreter for an HPSG-like grammar (see Franz 1990, Carpenter, Pollard and Franz 1991), no one has put forward a sensible definition for the default case. Immediate problems arise when considering such a system, even without recursion, such as whether to inherit a specification on a path before inheriting specifications on superconcepts, which may affect the same path's value.

Acknowledgements

I would like to thank Ted Briscoe, Jo Calder, Ann Copestake, Roger Evans and Graham Russell for getting me interested in default unification in the first place and discussing the basics of their own approaches with me. I would also like to thank Gosse Bouma for providing specific comments on comparisons between his definition and the one found here. I should also say that almost everything I know about defaults that I didn't learn from those mentioned above was gained in conversations with Rich Thomason.

References

Aït-Kaci, H. (1986). *Solving type equations by graph rewriting*, volume 202 of *Lecture Notes in Computer Science*, pages 158–179. Springer-Verlag, West Berlin, FRG.

Bouma, G. (1990a). Defaults in unification grammar. In *Proceedings of the 28th Annual Conference of the Association for Computational Linguistics*, pages 165–172, Pittsburgh.

Bouma, G. (1990b). Non-monotonic inheritance and unification. In *Proceedings of the First International Workshop on Inheritance in Natural Language Processing*, pages 1–8, Tilburg, The Netherlands.

Calder, J. (1987). Typed unification for natural language processing. In Klein, E. and van Benthem, J., editors, *Categories, Polymorphism and Unification*, pages 65–72. Centre for Cognitive Science, University of Edinburgh, Edinburgh.

Calder, J. (1991). A Note on Priority Union. Paper presented at the ACQUILEX Workshop on Default Inheritance in the Lexicon, Cambridge, April 1991.

Carpenter, B. (1989). PROP: A Prolog implementation of PATR-II. Technical Report LCL-89-3, Laboratory for Computational Linguistics, Carnegie Mellon University, Pittsburgh.

Carpenter, B. (1990). Typed feature structures: Inheritance, (in)equations and extensionality. In *Proceedings of the First International Workshop on Inheritance in Natural Language Processing*, pages 9–13, Tilburg, The Netherlands.

Carpenter, B., Pollard, C., and Franz, A. (1991). The specification and implementation of constraint-based unification grammars. In *Proceedings of the Second International Workshop on Parsing Technology*, Cancun, Mexico.

Evans, R. (1987). Towards a formal specification of defaults in gpsg. In Klein, E. and van Benthem, J., editors, *Categories, Polymorphism and Unification*, pages 73–93. Centre for Cognitive Science, University of Edinburgh, Edinburgh.

Evans, R. and Gazdar, G. (1989a). Inference in DATR. In *Proceedings of the Fourth EACL*, pages 66–71, Manchester, England.

Evans, R. and Gazdar, G. (1989b). The semantics of DATR. In Cohn, A. G., editor, *Proceedings of teh Seventh Conference of the Society for the Study of Artificial Intelligence and Simulation of Behavior*, pages 79–87, London. Pitman/Morgan Kaufmann.

Evans, R. and Gazdar, G. (1990). The DATR papers. Cognitive Science Research Reports CSRP 139, University of Sussex, Sussex.

Flickinger, D. (1987). *Lexical Rules in the Hierarchical Lexicon.* PhD thesis, Stanford University, Stanford, California.

Flickinger, D., Pollard, C. J., and Wasow, T. (1985). Structure-sharing in lexical representation. In *Proceedings of the 23rd Annual Conference of the Association for Computational Linguistics.*

Franz, A. (1990). A parser for HPSG. Technical Report LCL-90-3, Laboratory for Computational Linguistics, Carnegie Mellon University, Pittsburgh.

Gazdar, G. (1987). Linguistic applications of default inheritance mechanisms. In Whitelock, P., Somers, H., Bennett, P., Johnson, R., and Mcgee Wood, M., editors, *Linguistic Theory and Computer Applications*, pages 37–68. Academic Press, London.

Gazdar, G., Klein, E., Pullum, G., and Sag, I. (1985). *Generalized Phrase Structure Grammar.* Basil Blackwell, Oxford.

Kaplan, R. (1987). Three seductions of computational psycholinguistics. In Whitelock, P., Somers, H., Bennett, P., Johnson, R., and Mcgee Wood, M., editors, *Linguistic Theory and Computer Applications*, pages 149–188. Academic Press, London.

Karttunen, L. (1984). Features and values. In *Proceedings of the 10th International Conference on Computational Linguistics.*

Kasper, R. T. and Rounds, W. C. (1986). A logical semantics for feature structures. In *Proceedings of the 24th Annual Conference of the Association for Computational Linguistics*, pages 235–242.

Kasper, R. T. and Rounds, W. C. (1990). The logic of unification in grammar. *Linguistics and Philosophy*, 13(1):35–58.

Kay, M. (1984). Functional unification grammar: a formalism for machine translation. In *Proceedings of the 10th International Conference on Computational Linguistics*, pages 75–78.

Moens, M., Calder, J., Klein, E., Reape, M., and Zeevat, H. (1989). Expressing generalizations in unification-based grammar formalisms. In *Proceedings of the Fourth EACL*, pages 66–71, Manchester, England.

Moshier, D. (1988). *Extensions to Unification Grammar for the Description of Programming Languages.* PhD thesis, University of Michigan, Ann Arbor.

Moshier, D. and Rounds, W. (1987). A logic for partially specified data structures. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages.*

Pereira, F. C. N. and Shieber, S. M. (1984). The semantics of grammar formalisms seen as computer languages. In *Proceedings of the 10th International Conference on Computational Linguistics*, pages 123–129.

Pollard, C. J. and Moshier, M. D. (in press). Unifying partial descriptions of sets. In Hanson, P., editor, *Information, Language and Cognition*, volume 1 of *Vancouver Studies in Cognitive Science.* University of British Columbia Press, Vancouver.

Pollard, C. J. and Sag, I. A. (1987). *Information-Based Syntax and Semantics: Volume I – Fundamentals*, volume 13 of *CSLI Lecture Notes.* Chicago University Press, Chicago.

Rounds, W. C. and Kasper, R. T. (1986). A complete logical calculus for record structures representing linguistic information. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science*, Cambridge, Massachusetts.

Russell, G., Carroll, J., and Warwick, S. (1990). Multiple inheritance in a unification-based lexicon. In *Proceedings of the First International Workshop on Inheritance in Natural Language Processing*, pages 93–103, Tilburg, The Netherlands.

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

Shieber, S. M., Uszkoreit, H., Pereira, F. C. N., Robinson, J., and Tyson, M. (1983). The formalism and implementation of PATR-II. In *Research on Interactive Acquisition and Use of Knowledge*, volume 1894 of *SRI Final Report.* SRI International, Menlo Park, California.

Smolka, G. (1988). A feature logic with subsorts. LILOG-REPORT 33, IBM – Deutschland GmbH, Stuttgart, FRG.

Smolka, G. (1989). Feature constraint logics for unification grammars. IWBS Report 93, IBM – Deutschland GmbH, Stuttgart, FRG. To appear in *Journal of Logic Programming.*

Steele Jr., G. L. (1990). *Common Lisp.* Digital Press, Bedford, Massachussetts, second edition.

Prioritised Multiple Inheritance In DATR*

Roger Evans, Gerald Gazdar and Lionel Moser

Cognitive and Computing Sciences
University of Sussex

August 1991

Abstract

We characterise a notion of prioritised multiple inheritance (PMI) and contrast it with the more familiar orthogonal multiple inheritance (OMI). DATR is a knowledge representation language that was designed to facilitate OMI analyses of natural language lexicons: it contains no special purpose facility for PMI and this has led some researchers to conclude that PMI analyses are beyond the expressive capacity of DATR. Here, we present three different techniques for implementing PMI entirely within DATR's existing syntactic and semantic resources. In presenting them, we draw attention to their respective advantages and disadvantages.

1 Introduction

"Multiple inheritance", in inheritance network terminology, describes any situation where a node in an inheritance network inherits information from more than one other node in the network. Wherever this phenomenon occurs there is the potential for conflicting inheritance, i.e., when the information inherited from one node is inconsistent with that inherited from another. Because of this, the handling of multiple inheritance is an issue which is central to the design of any formalism for representing inheritance networks. For the formalism to be sound, it must provide a way of avoiding or resolving any conflict which might arise. This might be by banning multiple inheritance altogether, restricting it so that conflicts are avoided, providing some mechanism for conflict resolution as part of the formalism itself, or providing the user of the formalism with the means to specify how the conflict should be resolved.

Touretzky (1986, p70ff) provides a formal description of a number of properties that an inheritance network may have, and discusses their significance with respect to the problem of multiple inheritance. Tree-structured networks, as their name suggests, allow

*We are grateful to Dan Flickinger and Susan Warwick for relevant conversations on the topic of this paper.

any node to inherit from at most one other node, so multiple inheritance conflict cannot arise. Orthogonal networks allow a node to inherit from more than one other node, but the properties it inherits from each must be disjoint, so that again, no conflict can possibly arise. In Touretzky's framework all inheritance links are essentially simple specifications of truth values (`true`, `false` or `unknown`) for predicates, and in this situation the orthogonality constraint is the only sure way to guarantee network consistency. However, more general inheritance patterns can be achieved by using more powerful inheritance mechanisms. One approach is to forego orthogonality and provide mechanisms for resolving among conflicting values inherited at a node (Touretzky's 'inferential distance' is an example of this). The examples in this paper indicate an alternative approach is also possible.

DATR (Evans and Gazdar 1989a,b, 1990) is an inheritance network formalism designed for the representation of natural language lexical information. The basic descriptive features of DATR allow the specification of simple orthogonal networks similar to Touretzky's. For example, if we write:

```
A:  <a> == true.

B:  <b> == false.

C:  <a> == A
    <b> == B.
```

then we are specifying a network of three nodes (`A` `B`, and `C`), and two 'predicates' (DATR paths `<a>` and `<b>`), with `C` inheriting a value for `<a>` from `A`, and for `<b>` from `B`. The network is orthogonal, since `<a>` and `<b>` represent disjoint (sets of) predicates.

This case provides an example of a *functional* DATR theory: a DATR theory is functional if and only if the set of equations defines a (partial) function from node/path pairs to value or inheritance descriptors. Since functionality is defined extensionally, it is trivial (for user or implementation) to check that a DATR theory is functional. Every functional DATR theory is orthogonal. Of course, there is more to DATR than is illustrated by the tiny example above, but the longest-defined-subpath-wins principle ensures that basic inheritance is always orthogonal in functional DATR theories.

Orthogonal multiple inheritance (OMI) is a very desirable property of lexical representation systems. Consider an analysis in which we put the common properties of verbs at a `V` node and the (completely disjoint) common properties of words that take noun phrase complements at an `NPC` node. A transitive verb (`VTR`) is both a verb and a word that takes an NP complement, thus it should inherit from both `V` and `NPC` in this analysis. In DATR, this might be expressed as follows:

```
V:  <cat> == verb.

NPC:<comp cat> == np
    <comp case> == acc.
```

```
VTR:<cat> == V
    <comp> == NPC.
```

However, a number of recent lexical theories have invoked a form of inheritance in which multiple parents with *overlapping* domains are specified, and a priority ordering imposed to resolve potential inheritance conflicts (e.g., Flickinger 1987, Russell et al. 1991). In this prioritised multiple inheritance (PMI), precedence is given to nodes that come earlier in the ordering, so that the inherited value for a property comes from the first parent node in the ordering that defines that property, regardless of whether other later nodes also define it (possibly differently).

Here is an abstract example of PMI written in a DATR-like pseudoformalism:

```
A:  <a> == one.

B:  <a> == two
    <b> == two.

C:  <a> == three
    <b> == three
    <c> == three.

ABC:<> == A, B, C.
```

Under the intended interpretation of the example, ABC inherits ¿from nodes A, B and C prioritised in that order, so that the following set of inferences could be drawn:

```
ABC:<a> = one
    <b> = two
    <c> = three.
```

while the following could not be drawn:

```
ABC:<a> = two
    <b> = three.
```

But, as the Mad Hatter once explained to Alice, pseudoformalisms have the great advantage of meaning whatever it is that one wants them to mean.

The question we address in this paper is whether DATR's style of OMI can be used to reconstruct PMI without making syntactic and semantic additions to the language. In fact, we shall describe no fewer than three different techniques for characterising PMI in DATR. For each technique, we consider the following simple scenario. We have nodes A and B defining values for paths <x> and <y> as follows:

```
A:  <x> == yes.

B:  <x> == no
    <y> == yes.
```

Our goal is to define a node C that will inherit values from node A where they are defined at A, but otherwise inherit them ¿from node B. Thus we will want to be able to derive the following as theorems:

```
C:  <x> = yes
    <y> = yes.
```

A fundamental property of all three approaches is that node A cannot actually leave *any* value undefined, since failure at any point during query evaluation would cause failure of the entire query. Instead, at the point where querying A 'ought' to fail, some mechanism for transferring attention to B must be provided.

2 Prioritising using explicit failure paths

In the first approach, we use a global path inheritance specification as the 'failure-value' for A:

```
A:  <> == "<not from_A>"
    <x> == yes.
```

Here, the value for <x> is defined in the normal way, but the empty path case uses a global descriptor to allow inheritance at that point to be conditioned by the queried node. In effect, we allow the queried node to specify the inheritance default for A. In our example, the querying node is C, and the inheritance required is inheritance from B:

```
C:  <> == A:<>
    <not from_A> == B:<>.
```

B itself need have no special properties:

```
B:  <x> == no
    <y> == yes.
```

To see how this works out in practice, consider our two queries C:<x> and C:<y>. The derivation of C:<x> is a straightforward inheritance from A[1]:

[1]By derivation here we mean the sequence of inheritance evaluations leading to the definition of a value. The DATR sentence justifying the inheritance is shown in parentheses on the right where applicable.

```
C:<x>
A:<x>                    (using C:<> == A:<>)
yes                      (using A:<x> == yes)
```

The derivation of `C:<y>` looks like this:

```
C:<y>
A:<y>                    (using C:<> == A:<>)
C:<not from_A y>         (using A:<> == "<not from_A>")
B:<y>                    (using C:<not from_A> == B:<>)
yes                      (using B:<y> == yes)
```

In the first case, A supplies the value, and the conflicting value ¿from B is not considered at all, while in the second, A fails to supply the value, and so the value at B is inherited by C.

It is easy to see that this technique can be extended to an arbitrary sequence of nodes to try in turn — by adding `B:<> == "<not from_B>"`, C can specify a further inheritance for values not defined at either A or B. It is also possible to extend this approach to more complex inheritance patterns, including cases where A or B itself inherits from a further node.

The explicit failure path approach is minimalist in the DATR resources it employs, and offers a clean, understandable, low level solution to the problem. However, (adopting a procedural metaphor) this approach to PMI in DATR effectively chains evaluation of the query at B after evaluation at A. This means that if the attempted evaluation at A changes the evaluation context, the mechanism will not function correctly: the inheritance specification `"<not from_A>"` exploits both the global node (through which it inherits) and the local path (to which it prefixes). If either of these have been changed during evaluation at A, the inheritance may not be as expected.

3 Prioritising using evaluable paths

One way to overcome the problem just described is to evaluate the inheritance possibilities separately using evaluable paths. Consider the following DATR sentence:

```
C:  <> == <A B>.
```

This specifies evaluation via two nested inheritance specifications (A and B) operating completely independently of each other. The results of these evaluations are spliced together to produce the body of a path inheritance specification, which is itself evaluated. If any of A, B or the final path inheritance fails, the entire inheritance fails[2].

[2]We are adopting here Gibbon's (1989) approach to evaluable paths in DATR: in earlier, unpublished work, we only permitted evaluable paths to contain descriptors that evaluated to atoms. To convert

We can produce a more elegant and robust implementation of PMI by using this technique and by exploiting the fact that DATR effectively ignores arbitrary extensions to paths not referenced in the definitions. We can ensure that neither A nor B will fail by providing a 'failure-value', and in this case we make it the empty value sequence:

```
A:  <> == ()
    <x> == yes.

B:  <> == ()
    <x> == no
    <y> == yes.
```

Now, consider what happens when we use the evaluable path specification introduced above: the path constructed contains the result from A followed by the result from B. If A defines a value, then that value is a leading subsequence of the elements of the path (followed by the value of B, if any). If A does not define a value, it returns the empty sequence, which disappears when absorbed into the path, so B's value alone is in the path. Either way, the answer we require is a leading subsequence of the path elements. To return the actual result required, we add statements to C, mapping paths with such leading subsequences into their associated values, but ignoring any trailing path elements, produced by inheritance lower down the prioritisation ordering.

```
C:  <> == <A B>
    <yes> == yes
    <no> == no.
```

The derivation of C:<x> now looks like this:

```
C:<x>
C:<A:<x> B:<x> x>      (using C:<> == <A B>)
C:<yes no x>           (using evaluable path instantiation)
yes                    (using C:<yes> == yes)
```

Notice how the result of B:<x> (no) is ignored when mapping to the final result. C:<y> comes out as:

```
C:<y>
C:<A:<y> B:<y> y>      (using C:<> == <A B>)
C:<() yes y>           (using evaluable path instantiation)
C:<yes y>
yes                    (using C:<yes> == yes)
```

value sequences to paths, it was necessary to invoke an explicit sequence-to-path coercion operator. It now seems to us that Gibbon's approach (which dispenses with the operator and allows descriptors to evaluate to value sequences in paths) is syntactically more elegant. The two approaches are equivalent semantically, so no expressive power is lost (or gained) by adopting one rather than another.

This PMI technique will work even when A and B return more complex results. However, it depends on a mapping from these results onto the results actually returned by C. In our example, this is an identity mapping, but nothing forces this: the results of A and B can be viewed more generally as indexes into a table of arbitrary inheritance specifications.

Although elegant, the evaluable path technique has one major drawback: a (finite) table has to be provided, mapping from every possible value returned by A or B onto an appropriate result. This is acceptable for applications that are restricted to a small set of possible results, but may well be impractical for larger finite applications, and is expressively inadequate for applications where the set of possible results is unbounded (as can easily arise when value sequences are returned).

4 Prioritising using negative path extension

Our third technique solves the finite table problem by encoding the successful inheritance source (rather than the result) and then using it to direct the inheritance. Once again we need an explicit value to indicate failure of A or B, but this time it is not the empty value sequence, but instead an arbitrary but unique result value, undef.

```
A:  <> == undef
    <x> == yes.

B:  <> == undef
    <x> == no
    <y> == yes.
```

Node C is more complicated[3]:

```
C:  <> == <<one_of A B>>
    <one_of> == from_A
    <one_of undef> == from_B
    <one_of undef undef> == nowhere
    <nowhere> == undef
    <from_A> == A:<>
    <from_B> == B:<>.
```

Much as before, the prioritised inheritance operates by evaluating A and B inside an evaluable path and collecting up the results returned, now including undefs for failed queries. The resulting path, prefixed with one_of, is evaluated against a set of definitions which use the presence or absence of undef's to select among tokens from_A, from_B or

[3]In a real application, the bulk of this complexity could be stated once at a higher node rather than associated with every lower node that needs to make use of it. But we have located all the machinery at node C here, in order to keep the examples for our three techniques as comparable as possible.

nowhere. These in turn are evaluated as path elements (note the *double* path brackets on the empty path definition in C), directing inheritance through either A or B, or returning undef as a result. Our two derivations are now as follows:

```
C:<x>
C:<C:<one_of A:<x> B:<x> x> x>
                          (using C:<> == <<one_of A B>>)
C:<C:<one_of yes no x> x>
                          (using evaluable path instantiation)
C:<from_A x>              (using C:<one_of> == from_A)
A:<x>                     (using C:<from_A> == A:<>)
yes                       (using A:<x> == yes)

C:<y>
C:<C:<one_of A:<y> B:<y> y> y>
                          (using C:<> == <<one_of A B>>)
C:<C:<one_of undef yes y> y>
                          (using evaluable path instantiation)
C:<from_B y>              (using C:<one_of undef> == from_B)
B:<y>                     (using C:<from_B> == B:<>)
yes                       (using B:<y> == yes)
```

Thus the path resulting from evaluation at the parents A and B is matched against the 'selector' paths, <one_of>, <one_of undef>, and <one_of undef undef>. When node A returns a value other than undef, the instantiated path extends neither <one_of undef> nor <one_of undef undef>, and so inheritance through <one_of> occurs. If on the other hand A 'fails' and returns undef, <one_of> is ignored and the same situation applies for the value returned by B matching against <one_of undef>.

This technique might appropriately be called *negative path extension.* DATR's default mechanism operates by selecting the statement containing the longest leading subpath of the query path. Typically, this is used by specifying a path containing the particular domain attributes which correspond to the given definition. Here, however, we have introduced a distinguished value (undef), not properly part of the descriptive domain of the other path attributes, and used it in a negative fashion: the 'interesting' paths (at this point in the derivation) are those which do *not* extend undef, and which are therefore forced to inherit through the more general (shorter) path.

Although this technique is not the simplest, it has the advantage of working in completely arbitrary (acyclic) network configurations. There can be an arbitrary number of parent nodes, arbitrarily related to each other. Furthermore, regardless of the inheritance specifications in the parents, no context is ever altered by the prioritisation step: the final inheritance from a parent occurs in the same local and global state as a direct inheritance would.

5 Conclusion

The three examples that we have presented above demonstrate how PMI can be implemented in DATR in a variety of ways, using mechanisms already provided in the language for quite different purposes. The three techniques given are illustrative and there may well be other ways of implementing PMI within DATR, quite possibly better ways. Furthermore, we are not seeking to recommend any one of the techniques over the other two, but simply to clarify the differences between them. As we have seen, they differ in flexibility and clarity, and indeed in efficiency, especially when used in implementations of DATR that make use of non-cacheing query algorithms.

We mentioned in the introduction that the approaches we describe differ in kind from techniques such as Touretzky's inferential distance. In effect, the DATR approach is to control conflict by not allowing it to arise: instead of resolving conflicting inherited values, we control the inheritance specifications themselves, directing inheritance deterministically to the appropriate parent. DATR's descriptive mechanisms thus allow an orthogonal, declarative treatment of prioritised multiple inheritance.

But it is also interesting to note that the first example suggests that at least a non-trivial subclass of PMI is almost within the reach of the simplest imaginable OMI systems: the only significant feature of DATR used there is the renaming of the property inherited (the prefixation of `<not from_A>` to the path) to indicate evaluation failure in `A`. The latter two analyses, however, exploit the more esoteric facilities of DATR, namely nested evaluation and value to property coercion.

DATR was designed to facilitate OMI analyses of natural language lexicons. The designers of DATR were not persuaded then (or now) that PMI treatments of the lexicon offer significant analytical or descriptive advantages. However, despite their theoretical intentions and analytical prejudices, it seems that DATR is surprisingly well suited to PMI-style lexicons.

References

[Evans & Gazdar 1989a] Roger Evans & Gerald Gazdar: Inference in DATR. *Fourth Conference of the European Chapter of the Association for Computational Linguistics* (1989) 66-71.

[Evans & Gazdar 1989b] Roger Evans & Gerald Gazdar: The semantics of DATR. In Anthony G. Cohn, ed. *Proceedings of the Seventh Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour.* London/Los Altos: Pitman/Morgan Kaufmann (1989) 79-87.

[Evans & Gazdar 1990] Roger Evans & Gerald Gazdar, eds.: *The* DATR *Papers.* Brighton: University of Sussex Cognitive Science Research Paper CSRP 139 (1990).

[Flickinger 1987] Daniel P. Flickinger: *Lexical Rules in the Hierarchical Lexicon*, doctoral dissertation, Stanford University (1987).

[Gibbon 1989] Dafydd Gibbon: **PCS-DATR:** A DATR implementation in **PC Scheme.** *English/Linguistics Interim Report* No. 3, University of Bielefeld (1989).

[Russell et. al. 1991] Graham Russell, John Carroll & Susan Warwick: Multiple Default Inheritance in a Unification-Based Lexicon. *Proceedings of the 29th Annual Meeting of the ACL,* 215-221, (1991).

[Touretzky 1986] David S. Touretzky: *The Mathematics of Inheritance Systems.* London/Los Altos: Pitman/Morgan Kaufmann (1986)

Norms or Inference Tickets?
a frontal collision between intuitions

M. Morreau

IMS, Universitaet Stuttgart, Keplerstr 17 D-7000 Stuttgart 1

mimo@adler.philosophie.uni-stuttgart.de

abstract

Theories of nonmonotonic reasoning are, on the face of it, of at least two sorts. In *Circumscription*, generic facts like *birds fly* are taken to be essentially normative, and nonmonotonicity arises when individuals are assumed to be as normal as is consistent with available information about them. In theories like *Default Logic* and *Update Semantics* such facts are taken to be mere rules of inference, and nonmonotonicity arises when available information is augmented by adding as many as possible of the inferences sanctioned by such rules. According to which of the two views taken, different patterns of nonmonotonic reasoning seem appropriate. Here it is shown that these different patterns of reasoning cannot be combined in a single theory of nonmonotonic reasoning.

introduction

Nonmonotonic reasoning is that which lacks a monotonicity property which has been taken to be a characteristic of logical reasoning. In a theory of nonmonotonic reasoning, the consequences of a set of premises do not always accumulate as the set of premises is expanded. Nonmonotonicity has been researched in artificial intelligence because people reason nonmonotonically, and this in ways which seem directly related to intelligence. The last decade or so of artificial intelligence research has turned up several *prima facia* different sources of nonmonotonicity in reasoning. Thus autoepistemic reasoning, or reflection on ones information or lack thereof, seems a different thing from default reasoning.

Nonmonotonicity in autoepistemic reasoning is a direct consequence of introspection: someone who knows nothing of stamps and who is capable of introspection may, by doing autoepistemic reasoning, come to the conclusion that he knows nothing about stamps. This conclusion will then be retracted on learning something about stamps, because then it is no longer true. Beforehand it was not true that he knew something about stamps, afterwards it was true that he knew something about stamps, so autoepistemic reasoning is tied up with changes in the truth values of sentences involving certain "introspective" epistemic operators.

Default reasoning, or augmenting ones information by means of default rules like *if something is a bird, and you have no reason to assume that it cannot fly, then assume that it can fly*, allows one to jump to conclusions on the basis of gaps in ones information. Nonmonotonicity comes about because one sometimes must retract such premature conclusions once the gaps in ones information have been filled in. On the basis of the above rule, for example, one might jump to the conclusion that a particular bird can fly, lacking information to the contrary. On obtaining information to the contrary, however, the conclusion that the bird can fly must be withdrawn. Then there is reason to assume that the bird cannot fly, so the above rule no longer allows the premature conclusion to be drawn.

A third source of nonmonotonicity is what might be called *norms.* Philosophers of science have reduced dispositional propositions such as *table salt is soluable in water* to nondispositional terms by invoking the notion of normal circumstances. This particular sentence then comes out as something like *pieces of table salt dissolve when placed in water under normal circumstances.* Similarly, generic sentences like *birds fly* have been thought to be true or false depending on the properties of prototypical or normal birds. It is this sort of statement about what normal or ideal individuals are like, or what happens under normal or ideal circumstances, that I mean by norms.

To see how norms give rise to nonmonotonic reasoning, assume the above analysis of the fact that table salt is soluable in water, and suppose that a particular piece of salt has been added to water. Assuming circumstances to be normal then allows the conclusion to be drawn that the salt will dissolve. The reasoning is defeasible because this conclusion will be retracted if the additional information is made available that the circumstances are somehow abnormal, say because the water is frozen. For circumstances can no longer be assumed to be normal once they are known to be abnormal (one can of course under circumstances which are known to be abnormal reason counterfactually about how things would have been if circumstances *had been* normal - but that is different from assuming that they *are* normal). In the Artificial Intelligence literature, the best known theory of nonmonotonic reasoning from norms is McCarthy's theory of *circumscription.* In this theory, the notion of normality is modelled formally by introducing into the object language a first order predicate *Ab* of situations, individuals and so on, *Abt* being true in a model, intuitively speaking, if t is abnormal there. Assuming maximal normality can then be modelled as taking into account only models in which this predicate has a set-theoretically minimal extension.

There are of course some obvious differences between norms and default rules. Whereas norms are the sort of thing which can be true or false, for example, default rules are not. A default rule, like any other injunction to behave in a particular way, can only be obeyed or disregarded. It makes no more sense to say that a default rule is true than it makes to assign a truth value to an instruction like *after you pass the petrol station on your right, take the next turn to the left.* But when it comes to the nonmonotonic inferences which they sanction, there are some striking similarities between norms and default rules. Both the norm *pieces of table salt are such that if added to water under normal circumstances, they will dissolve* and a default rule like *if a piece of table salt is added to water and there is no reason to believe that it will not dissolve, then assume that it will dissolve* allow the conclusion to be drawn that a given bit of salt which has been added to a quantity of water will dissolve. A glance through the artificial intelligence literature provides more circumstantial evidence that reasoning with normative facts and default reasoning are, if not one and the same, then at least very similar things. The same generic examples involving birds and whether or not they can fly are used to motivate both, and the same general patterns of nonmonotonic reasoning, such as the so called *Nixon Diamond* and the *Tweety Triangle,* are judged appropriate to both.

Another indication that norms and default rules are alike is that they differ in the same way from autoepistemic reasoning. Nonmonotonicity from norms, like that from defaults, has nothing to do with changing truth values. When the conclusion is withdrawn that a particular piece of salt which has been addded to water has dissolved, once it becomes known that the water was frozen when the salt was added, it is not that this extra information has somehow affected the truthvalue of this conclusion. The truth or otherwise of the proposition that the salt has dissolved depends only on the distribution of sodium and chloride ions among water molecules, and not on the temperature of the water, let alone on whether or not one has information about the temperature. Nor does providing such information cause salt to precipitate out of solution. Rather, the effect of the new information is to remove the *plausibility* of a conclusion whose truthvalue remains unchanged.

It is the purpose of this paper to argue that the above appearances to the contrary, nonmonotonic reasoning deriving from assuming maximal normality and that deriving from inference tickets are very different things. I argue that each of these kinds of nonmonotonic reasoning has characteristic properties which make it incompatible with the other. In the next section, I take a closer look at nonmonotonic reasoning on the basis of norms, and make explicit some properties which such reasoning may naturally be assumed to have. Then I do the same for nonmonotonic reasoning based on defaults. After considering some general properties which are taken to be minimal requirements on any form of reasoning, monotonic or otherwise, in a penultimate section it is shown how the characteristic properties of nonmonotonic reasoning based on norms conflict with the characteristic properties of that based on defaults. A final section summarises what is to be learned from this incompatibility.

norms

Nonmonotonic reasoning which derives from norms has two characteristic features. First, it is based on sentences expressing the properties of the normal or prototypical individuals of some kind. These sentences are taken to be the sort of thing which can be true or false, which is to say they can have a truth conditional semantics. This semantics then gives rise to a *monotonic* logic of such expressions in the standard way. In the case of circumscription, for example, the normative sentences which give rise to the nonmonotonic reasoning are representations in first order logic of generic sentences like *birds fly*. This particular example is represented as follows:

$$\forall x(bird(x) \wedge \neg Ab(x) \rightarrow fly(x))$$

Here the monadic predicate Ab is supposed to express abnormality. The truth conditional semantics, in the case of circumscription, is simply that of classical first order logic, as is the monotonic logic.

The second characteristic of nonmonotonic reasoning which derives from norms is that the monotonic logic of the expressions involved is strengthened by doing something the effect of which is to assume maximal normality. In the case of circumscription, this is done by restricting attention to a subclass of the models of a set of premises in whose nonmonotonic consequences one is interested. The preferred subclass contains those models in which the special predicate Ab has a set theoretically minimal extension. The resulting nonmonotonic extension of the classical underlying logic is called "minimal entailment."

Let us introduce some notation: $\varphi > \psi$ will be used as a kind of meta-notation, by means of which we can refer to whatever formula or rule is used in a formalism to express that *φ's are ψ's*. Later, when we come back to inference tickets, $\varphi > \psi$ will be used to refer to these. Thus while we are talking about circumscription, *bird > fly* is shorthand for the first order sentence above; while we are talking about default logic, it is shorthand for the following default rule.

$$\frac{bird(x) : fly(x)}{fly(x)}$$

The turnstile $\models$ will denote any monotonic entailment notion that is present, and $\models\!\!\approx$ will refer to the nonmonotonic entailment notion. So in the case of circumscription, $\models$ is classical entailment, and $\models\!\!\approx$ is minimal entailment. Using this notation we can state some general properties of nonmonotonic formalisms. Among the general properties which a theory of nonmonotonic reasoning based on norms just about cannot avoid having is the following:

If $\psi_1(x),..., \psi_n(x) \models \zeta(x)$ then $\varphi > \psi_1,..., \varphi > \psi_n \models \varphi > \zeta$.

The justification for this principle on the view that generic sentences express the properties of prototypical or normal individuals is simple: if individuals which have the properties $\psi_1,..., \psi_n$ necessarily have the property ζ, and if furthermore prototypical φ's have these properties $\psi_1,..., \psi_n$, then prototypical φ's have the property ζ. Since on this view $\mid\!\approx$ strengthens $\models$, we can continue below with the following slightly weaker form of this principle of

LOGICAL CLOSURE IN THE CONSEQUENT[1]:
If $\psi_1,..., \psi_n \models \zeta$ then $\varphi > \psi_1,..., \varphi > \psi_n \mid\!\approx \varphi > \zeta$.

Another principle is messier to state, but has just as clear a justification. I would call it the

SEPARATION OF THE IDEAL AND THE ACTUAL:
Let Γ be a set of > sentences, and let ψ be another > sentence. Then if $\Gamma \mid\!\approx \psi$, then for all non > sentences φ: $\Gamma, \varphi \mid\!\approx \psi$.

That is, if some collection Γ of statements about normality yields (under the assumption of maximal normality) some other statement ψ about normality, then provided φ is not a statement about normality (really, I should require that it not be modal in any sense) they will do so even if φ is true.

An equally plausible counterpart to this requirement could be stated, for the case where Γ contains no > sentences or modal sentences, where ψ is likewise neither a > sentence nor a modal sentence, and where φ is a > sentence. But this counterpart is not needed below.

I've stated this principle generally, since I think that this way it sounds most plausible. It aims to capture something like the following: what is *normally* the case, like that which *should* be the case, has nothing to do with what is *actually* the case. Prototypes keep each other's company in a prototypical world, and don't care about what goes on in the actual world. Similarly, the sentences which describe what goes on in the realm of prototypes enter into logical relationships among themselves, oblivious to sentences - non > and non modal sentences - which describe the actual world. This notion is closely related to an idea which in philosophy is associated with Hume. According to Hume, that which should be cannot be derived from that which is.

inference tickets

According to what I am calling the "inference ticket" view of nonmonotonic reasoning, nonmonotonicity arises from default rules. On this view, a natural language generic sentence like *birds fly* expresses something like the following rule: if you think something is a bird, and you have no reason to assume that it cannot fly, then assume that it can fly. Theories of nonmonotonic reasoning based on inference tickets are characteristically unconcerned about truth-conditional semantics; reasoning is accounted for in a syntactic and procedural manner. Among the inference ticket theories of genericity I would count, besides Reiter's [1979] *Default Logic*, the theory of *Nonmonotonic Inheritance Networks* due to Horty, Thomason and Touretzky [1987], and Veltman's [1991] theory of *Defaults in Update Semantics.*

---

[1]Obviously, as stated here this is not a constraint on $\mid\!\approx$ but a family of such constraints, one for each notion $\models$ of monotonic entailment. The relatively strong version in which $\models$ is classical entailment might well be found persuasive enough. But for the purposes of the result to be proved later, closure of the consequent is only required under some relatively weak $\models$; specifically, for monadic predicates P, Q and R, I assume the following arguments to go through in $\models$: $Q, R \models Q \leftrightarrow R$ and $Q, Q \leftrightarrow R \models R$

What are some of the characteristic principles of reasoning by inference tickets? Compliance with a collection of rules involves compliance with each of the rules in the collection. Suppose you entertain the default *a φ is to be assumed to be a ψ_1 unless there is reason to assume otherwise,* and also the default *a φ is to be assumed to be a ψ_2 unless there is reason to assume otherwise.* Suppose in addition that according to your information, *δ is a φ, but not a ψ_1*. Then compliance with the first of these rules does not require you to assume that *δ is a ψ_1*, since you have reason to assume that *δ is not a ψ_1*. But provided the information that *δ is a φ, but not a ψ_1* does not constitute a ground for assuming that *δ is not a ψ_2*, compliance with the second rule does involve assuming δ to be a ψ_2.

Let us now write $\varphi > \psi$ to represent the inference ticket: *a φ is to be assumed to be a ψ unless there is reason to assume otherwise.* Then, as argued in the previous paragraph, something like the following principle of *independence* seems inescapable on an inference ticket view of nonmonotonic reasoning:

INDEPENDENCE: $\varphi > \psi_1, \varphi > \psi_2, \varphi\delta, \neg\psi_1\delta \mid\!\approx \psi_2\delta$

The above argumentation for this principle covers only the case where δ, φ, ψ_1 and ψ_2 are such that the information that δ is a φ but not a ψ_1 does not provide a ground for assuming that δ is not a ψ_2. So clearly this principle must be restricted to formulas φ, ψ_1 and ψ_2 which are, in some appropriate sense of the word, independent of each other. Here I won't attempt to characterise completely a suitable notion of independence. On the basis of the story told above, and in a simple setting where causal and other nonlogical evidential relations are excluded, it is plausible that *logical independence* in the following sense is a sufficient criterion for independence: monadic predicates φ, ψ_1, and ψ_2 are *logically independent* provided that for each constant δ, every boolean combination (involving at most one occurrance each) of the formulas $\varphi\delta$, $\psi_1\delta$, and $\psi_2\delta$ is satisfiable.

Satisfiable in what? It is in the nature of inference ticket theories that they don't provide truth conditions for the rules $\varphi > \psi$; some such theories, like Veltman's, adopt a dynamic view of meaning and so don't even suggest a truth conditional semantics for the formulas φ and ψ which make up these rules. In a theory like Veltman's there is no obvious candidate for a notion of satisfiability here. In such theories there may be other notions like coherence which can play the role I want satisfiability to play; for the purposes of the argument of this paper however, all that is required is that there are some three monadic predicates P, Q, and R which are independent in the appropriate sense, and that for these three P, Q and R, the predicates P, Q, and Q$\leftrightarrow$R are independent too. Note that if logical independence is accepted as a sufficient condition for independence, and if furthermore it is classical satisfiability (or any more permissive notion) by means of which logical independence is judged, then any three different monadic predicate letters P, Q and R will do.

Independence is a compelling principle of reasoning in inference ticket formalisms. It holds in the non-monotonic inheritance networks of Horty, Thomason and Touretsky, though in a fairly restricted sense, since in effect all of the predicates φ, ψ_1, and ψ_2 with which this theory deals are atomic. The above argument for independence is restricted in no such way, and one might expect there to be inference ticket formalisms where independence holds also for logically complex φ, ψ_1, and ψ_2. Veltman's [1991] theory of defaults is such a theory (though Veltman does restrict himself to φ, ψ_1 and ψ_2 which do not involve >-expressions). In fact, Veltman apparently takes the principle of independence[2] to be so central to the theory of reasoning about defaults, which he does not distinguish from what I have called norms, that he builds up his entire formal

---

[2]he calls it *graded normality.*

apparatus around it: "This principle embodies an essential feature of commonsense reasoning. So, I cannot but conclude that selection functions are not the right kind of entities to model an agent's knowledge of the rules. They are too simple." (p. 45). That the principle should be so essential, not only to reasoning with default rules but also to reasoning with norms and other forms of commonsense reasoning, Veltman takes to be self-evident. No special reasons are given for thinking it to be such.

some general properties of nonmonotonic formalisms

In the 'thirties, Tarski stated quite generally some minimal requirements which a relation $\vdash$ must fulfill if it is to be called a notion of logical consequence. Tarski's three requirements of reflexivity, idempotence and monotonicity are, in combination, equivalent to the combination of the following three requirements. Here Γ and Γ' range over sets of formulas, and φ over isolated formulas:

REFLEXIVITY: If $\varphi \in \Gamma$, then $\Gamma \vdash \varphi$.

CUT: If $\Gamma \vdash \Gamma'$ and $\Gamma \cup \Gamma' \vdash \varphi$, then $\Gamma \vdash \varphi$.

MONOTONICITY: If $\Gamma \vdash \varphi$, then $\Gamma \cup \Gamma' \vdash \varphi$.

Gabbay[1984] and Makinson[1989] have stated and investigated some minimal requirements which a relation $\mid\!\approx$ should satisfy if it is to be a notion of nonmonotonic logical consequence. Clearly the third of the above requirements must be given up. But, as Gabbay and Makinson suggest, a nonmonotonic logic may reasonably be required to retain at least the little monotonicity which the following notion of *cautious monotonicity* would salvage:

CAUTIOUS MONOTONICITY: If $\Gamma \mid\!\approx \varphi$ and $\Gamma \mid\!\approx \Gamma'$, then $\Gamma \cup \Gamma' \mid\!\approx \varphi$.

There is no obvious reason why a notion of nonmonotonic logical consequence should not be required to satisfy the first two of the above requirements, reflexivity and cut. And here is a good technical reason for wanting them satisfied: cautious monotonicity and cut (a combination which Makinson refers to as *cumulativity*) together make consequence notions behave themselves. For provided $\mid\!\approx$ satisfies these two requirements, it can easily be shown that if $\Gamma \subseteq C(\Gamma')$ and $\Gamma' \subseteq C(\Gamma)$, then $C(\Gamma') = C(\Gamma)$ (here and below, $C(\Gamma)$ is short for $\{\varphi: \Gamma \mid\!\approx \varphi\}$).

For one half of the trivial proof, let (i) $\Gamma \subseteq C(\Gamma')$ and (ii) $\Gamma' \subseteq C(\Gamma)$, and suppose (iii) $\varphi \in C(\Gamma')$. To be shown is that $\varphi \in C(\Gamma)$. By cautious monotonicity we have from (i) and (iii) $\varphi \in C(\Gamma \cup \Gamma')$. But then with cut and (ii): $\varphi \in C(\Gamma)$.

q.e.d.

Logicians on both sides of the normative/inference ticket divide take it to be a bad sign if a system lacks these properties. Thus for example Horty, Thomason and Touretzky note with displeasure that their nonmonotonic inheritance algorithm is not cumulative.

norms are not inference tickets

In this section it will be shown that, against the background of these three general constraints on notions of nonmonotonic consequence, the principles which were deemed appropriate in the special case of reasoning about norms are in conflict with the principles deemed appropriate to reasoning with default rules. That is, I show that given *cut*,

cautious monotonicity and *reflexivity*, the principles of *separation of the ideal from the actual* and *closure of the consequent* are at odds with the principle of *independence*.

More specifically, I will show that $\mid\!\approx$ satisfies all of these constraints and principles only if the intuitively unproblematic theory Γ defined as $P>Q$, $P>R$, $P\delta$, $\neg Q\delta$ (say, *birds fly, birds lay eggs, tweety is a bird, tweety does not fly*) is $\mid\!\approx$ inconsistent.[3] Here P, Q and R are assumed to be independent monadic predicates.

Let Γ be this theory, and let Γ' be the theory $P>Q$, $P>(Q\leftrightarrow R)$, $P\delta$, $\neg Q\delta$. We now want to show that Γ is $\mid\!\approx$ inconsistent.

By independence,

(i) $\Gamma \mid\!\approx R\delta$, and $\Gamma' \mid\!\approx Q\delta\leftrightarrow R\delta$, while by reflexivity
(ii) $\Gamma' \mid\!\approx \neg Q\delta$.

By the principle of *closure of the consequent under logic* together with the *separation of the ideal and the actual*, however,

$\Gamma \subseteq Cn(\Gamma')$ and $\Gamma' \subseteq Cn(\Gamma)$, so with cut and cautious monotonicity we have:
$Cn(\Gamma') = Cn(\Gamma)$.

Thus with (i) and (ii) we have $R\delta$, $Q\delta\leftrightarrow R\delta$ $\neg Q\delta \in Cn(\Gamma)$, which makes Γ inconsistent.
q.e.d.

conclusion

Nonmonotonic reasoning which arises from normative statements like *Birds normally fly* is to be distinguished from that which arises from default rules or "inference tickets" like *If something is a bird, then assume that it can fly unless there is reason to assume the contrary.* These two kinds of nonmonotonic reasoning, exemplified respectively by circumscription and default logic, give rise to distinctive patterns of reasoning which would conflict were they to be combined in one theory.

All inference ticket theories of nonmonotonic reasoning that I know of, including default logic, the theory of nonmonotonic inheritance, and the theory of defaults in update semantics, are largely motivated with reference to norms like the example above. The result proved here suggests that such theories cannot (and indeed should not) do justice to this motivation. They cannot do justice to patterns of reasoning which are appropriate to such normative statements because they are in a different business.

Similarly, in developing a theory of nonmonotonic reasoning based on norms, one should be wary of principles of reasoning such as the principle of *independence* discussed above, for which I can think of no justification other than that which appeals to inference tickets.

Lastly, theorists on both sides of the norms-inference ticket divide have taken themselves to be explicating the meaning of natural language generic statements like *Lions eat meat* and *Boys don't cry*. Apparantly it can't be that both sides are right.

---

[3] In the usual sense that $C(\Gamma)$ contains some formula together with its negation. I take this opportunity to sneak in an extra assumption, namely that $C(\Gamma)$ is closed under "modus ponens" across the material biconditional $\leftrightarrow$.

literature

Gabbay [1984]. "Theoretical Foundations for Nonmonotonic Reasoning in Expert Systems," in K. Apt, (Ed.) *Logics and Models of Concurrent Systems*, Springer Verlag, Berlin, 1985.

Horty, J., Thomason, R., and Touretzky, D. [1987]. "A Skeptical Theory of Inheritance in Nonmonotonic Nets" Technical Report CMU.CS.87.175.

Hume, D. [1888] *A Treatise of Human Nature*, Selby-Bigge edition, Clarendon Press, Oxford.

Makinson, D. [1989]. "A General Theory of Cumulative Inference" in Reinfrank, de Kleer, Ginsberg and Sandewall (Eds.) *Nonmonotonic Reasoning*, Springer Lecture Notes in Artificial Intelligence vol. 346.

McCarthy, J. [1980]. "Circumscription - a form of nonmonotonic reasoning" *Artificial Intelligence*, vol.13.

Reiter, R. [1980]. "Default Logic, *Artificial Intelligence* vol.13.

Tarski, A. [1935] "On the Concept of Logical Consequence," reprinted as Chapter XVI of Tarski, *Logic, Semantics, Metamathematics*, translated by J.H. Woodger, Clarendon Press, Oxford, 1956.

Veltman, F. [1991]. "Defaults in Update Semantics" ITLI prepublication series LP-91-02, University of Amsterdam, Netherlands.

Issues in the design of a language for representing linguistic information based on inheritance and feature structures

Rémi Zajac
Project POLYGLOSS*
IMS-CL/IfI-AIS, University of Stuttgart
Keplerstraße 17, D-7000 Stuttgart 1
zajac@informatik.uni-stuttgart.de

April 15, 1991

DRAFT submitted for the *Acquilex Workshop on Default Inheritance*, 18-20th April 1991, Computer Laboratory, Cambridge University, UK.

Abstract

In this paper, we adress some issues in the design of declarative languages based on the notion of inheritance. First, we outline the connections and similarities between the notions of object, frame, conceptual graph and feature structures and we present a synthetic view of these notions. We then present the Typed Feature Structure (TFS) language developed at the University of Stuttgart which reconcilies the object-oriented approach with logic programming. We finally discuss some language design issues.

Contents

*Research reported in this paper is partly supported by the German Ministry of Research and Technology (BMFT, Bundesminister für Forschung und Technologie), under grant No. 08 B3116 3. The views and conclusions contained herein are those of the author and should not be interpreted as representing official policies.

1 Convergences

Developing large NLP software is a very complex and time consuming task. The complexity of NLP can be characterized by the following two main factors:

1. NLP is data-intensive. Any NLP application needs large amounts of complex linguistic information. For example, a realistic application has typically dictionaries with ten thousands of lexical entries.
2. Sophisticated NLP applications such as database interfaces or machine translation build very complex and intricate data structures for representing linguistic objects associated to strings of words. Part of the complexity also lies in the processing of such objects.

1.1 Object-oriented approaches

An object-oriented approach to linguistic description addresses these two sources of complexity by providing:

1. facilities to manage the design process: data abstraction and inheritance.
2. facilities for capturing directly the interconnections and constraints in the data: properties, relations and complex objects.

These features are common to object-oriented languages (OOL), object-oriented database management systems (OODBMS) or knowledge representation languages (KRL). There are more than superficial similarities between the notions of classes, types and concepts: they are all organized in inheritance hierarchies with some broad consensus on the abstract interpretation of such hierarchies (usually logical implication or set-inclusion), but with as many finer distinctions as there are specific languages. Likewise, there are strong relationships between the different data models for objects: frames, complex (recursive) records, nested tuples, or conceptual graphs.

Typically, these languages offer:

- complex objects can be represented as graphs where edges are labeled by roles and nodes by concept names.
- multiple inheritance
- role-value restrictions
- role-value equality
- classification

1.2 Logic programming

Computation is usually done either by providing a fully integrated procedural programming language (e.g. Smalltalk) or by providing an adequate interface to some programming language (e.g. LISP or C for O_2 [Lécluse/Richard/Velez 88]). A very few systems provide a declarative component for computation: e.g. FOOPlog [Goguen/Meseguer 87] or CLASSIC [Borgida et al. 89]. In these systems, procedural method/message passing is replaced with a logical rule-based component (in CLASSIC) or generic modules (in FOOPlog), which offer a much cleaner integration of the data component and the computational component. The evolution towards declarativity is motivated by [Goguen/Meseguer 87, p. 7] as follows:

> We believe that the many advantages claimed, including simplicity, clarity, understandability, reusability and maintainability, are all compromised to the degree that a programming language fails to correspond to a pure logic.

All these advantages are crucial in the development of NLP systems. Ideally, a linguistic formalism should be an object-oriented logic formalism. Feature structures, as used in unification-based grammar formalisms, provide the link between the object-oriented world and the logic programming world. Objects bear strong similarities with feature structures. For example, the O_2 data model has a notion of identity between parts of objects [Lécluse/Richard/Velez 88], which is equivalent to the notion of "sharing" or "reentrancy" in feature structures. In his PhD dissertation, Aït-Kaci [Aït-Kaci 84, Aït-Kaci 86] bridged the gap between these models. He synthesized the notions of inheritance, types and feature structure in a unique data structure, typed feature structures (called "ψ-terms"), and gave them a formal declarative semantics.

1.3 Typed feature structures

Assume the existence of an (abstract) informational domain, for example the set of linguistic objects. *Feature terms* describe objects of this universe by specifying values for attributes of objects. More precisely, as feature terms can provide only partial information about the objects they describe, a feature term denotes a *set* of objects in this universe. Feature terms are ordered by a subsumption relation: a feature term f_1 subsumes another feature term f_2 iff f_1 provides *less information* than f_2: $f_1 \geq f_2$. In our universe, this means that the set described by f_1 is *larger* than the set described by f_2. Note that there can be feature terms that describe objects which are not comparable (with respect to the subsumption relation), like for example a feature term describing verb phrases and a feature term describing noun phrases: the intersection of the sets they denote is empty.

As different sets of attribute-value pairs make sense for different kinds of objects, we also divide our feature terms into different types, which we call *feature types*. These types are ordered by a subtype relation: a type t_1 is a *subtype* of another type t_2 if t_1 provides *more information* than t_2. For example, if one assumes that a verb phrase is a phrase, then the set of verb phrases is included in the set of phrases. Using types to model this taxonomic hierarchy, the type symbol VP denotes the set of verb phrases, the symbol PH denotes the set of phrases, and we define VP as a subtype of PH.

This description implies of course that, if we know that a linguistic object is a verb phrase, we can deduce that it is a phrase. This deduction mechanism is expressed in our type system as *type inheritance*. Furthermore, with each type we associate *constraints* expressed as feature terms, thereby defining an inheritance hierarchy of typed feature terms: if a feature term is of type t_1 and there exist

supertypes of t_1, then t_1 inherits all the attribute-value pairs of the feature terms associated with the supertypes. A feature term of type VP describing a verb phrase can have an embedded verb phrase: the definition of the type VP is recursive, and this recursivity will give us the necessary expressive power to describe any complex linguistic object.

Given a typed feature term inheritance hierarchy, we can query the system and ask if some feature term belongs to the hierarchy. To produce the answer, the system will check the necessary and sufficient conditions such that all subterms of the query belong to the hierarchy. An answer will be the set of substitutions used to check these conditions: the answer will be printed as a set of feature terms subsumed by the query where all necessary and sufficient conditions hold. In that sense, the answer will give the best approximation of the set of objects denoted by the query, giving a formal basis of the traditional interpretation of feature structures as representing partial information.

A language based typed feature structures extends object-oriented programming with relational logic programming features:

- logical variables,
- non-determinism with backtraking,
- existential queries.

and conversely, extend logic programming with OO features:

- typed complex objects with role-value restrictions,
- multiple inheritance,
- classification.

Such a language is an attempt to combine the best of two worlds: declarativity and referential transparency from logic programming, modularity through inheritance and complex objects from object-oriented programming. Furthermore, based on feature structures, it is a good candidate as a computational linguistics formalism.

The basic approach described in this paper is based on original work by [Aït-Kaci 84, Aït-Kaci 86] on the KBL language and has been influenced by the work on HPSG by [Pollard/Sag 87, Pollard/Moshier 90]. The presentation in this paper is rather informal, and a more technical account can be found in [Emele/Zajac 90a, Zajac 90b]. Among the growing literature on the semantics of feature structures, many relevant results and techniques have been published by [Smolka 88, Smolka/Aït-Kaci 88, Smolka 89, Aït-Kaci/Podelski 90]. Based on [Pollard/Sag 87, Pollard 90, Pollard/Moshier 90], a computational formalism, very close to the TFS formalism, is currently under design at CMU for implementing HPSG [Carpenter 90, Franz 90]. The TFS formalism presented in the following section has been deisgned and implemented at IMS by Martin Emele and the author.

2 The TFS language

2.1 Types

The universe of feature terms is structured in an inheritance hierarchy which defines a partial ordering on kinds of available information. The backbone of the hierarchy is defined by a finite set of type

symbols $\mathcal{T}$ together with a partial ordering $\leq$ on $\mathcal{T}$: the partially ordered set (poset) $\langle \mathcal{T}, \leq \rangle$. The ordering $\leq$ defines the subtype relation: for $A, B \in \mathcal{T}$ we read $A \leq B$ as "A is a subtype of B".

In order to have a well-behaved type hierarchy, we require that $\langle \mathcal{T}, \leq \rangle$ be such that:

- $\mathcal{T}$ contains the symbols $\top$ and $\bot$, where $\top$ is the greatest element and $\bot$ is the least element of $\mathcal{T}$.
- any two type symbols A and B of $\mathcal{T}$ have a greatest common lower bound called the infimum of A, B and written $inf\{A, B\}$. A poset where this property holds is a meet semi-lattice: we introduce a new operation $A \wedge B = inf\{A, B\}$, where $A \wedge B$ is called the *meet* of A and B.

The tuple $\langle \mathcal{T}, \leq, \wedge \rangle$ is formally a meet semi-lattice. A technicality arises when two types A and B have more than one infimum: in that case, the set of infimums is interpreted disjunctively. We call the smallest types of $\mathcal{T}$, the minimal types.

A type hierarchy is interpreted set-theoretically, subtyping corresponding to set inclusion.

- $\top$ is interpreted as the whole universe.
- $\bot$ is interpreted as the empty set
- Any type of the hierarchy is interpreted as a subset of the universe.
- $\leq$ is interpreted as set inclusion: a subtype inherits all information of all its supertypes.
- $\wedge$ is interpreted as set intersection.

2.2 Feature terms

As different combinations of attribute-value pairs make sense for different kinds of objects, we divide our feature terms into different types. Types are closed in the sense that each type defines a specific collection of features (and restrictions on their possible values) which are appropriate for it, expressed as a feature structure (the definition of the type). Since types are organized in an inheritance hierarchy, a type inherits all the features and value restrictions from all its super-types. This type-discipline for feature structures enforces the following two constraints[1]: a type cannot have a feature which is not appropriate for it and conversely, a pair of feature and value should always be defined for some type. Thus a feature term is always typed.

As a notational convention, we use the attribute-value matrix (AVM) notation for feature terms and we write the type symbol for each feature term in front of the opening square bracket of the AVM. A type symbol which does not have any feature defined for it is called atomic. All other types are complex. In the remainder of this section, we shall implicitly refer to some given signature $\langle \mathcal{T}, \leq, \wedge, \mathcal{F} \rangle$ where $\langle \mathcal{T}, \leq, \wedge \rangle$ is a type hierarchy, and $\mathcal{F}$ is a set of feature symbols, and we shall also assume a set of variables V.

A feature term t is an expression of the form

$$\#x = A[f_1{:}t_1, \ldots, f_n{:}t_n]$$

[1]Which can be checked at compile time already.

where $\#x$ is a variable in a set of variables V, A is a type symbol in $\mathcal{T}$, $f_1, \ldots, f_n$ (with $n \geq 0$) are features in $\mathcal{F}$, and $t_1, \ldots, t_n$ are feature terms.

We have to add some restrictions that capture properties commonly associated with feature structures:

1. A feature is a selector that gives access to a subterm: it has to be unique for a given term.
2. $\bot$ represents inconsistent information: is it not allowed in a term.
3. A variable is used to capture equational constraints ("reentrancy") in the term: there is at most one occurrence of a variable $\#x$ in a term which is the root of a term different than $\#y = \top$.

Given a signature $\langle \mathcal{T}, \leq, \wedge, \mathcal{F} \rangle$, feature terms are partially ordered by a subsumption relation. This captures the intuitive notion that a term containing a lot of information is more specific, and describes a smaller set than a term which contains less information. A feature term t subsumes a term t', $t \geq t'$ iff:

1. all the paths in t are in t';
2. all equational constraints in t hold in t'.
3. for a given path in t, its type is greater or equal than the corresponding type in t';

Since we have a partial order on feature terms, the meet operation between two feature terms t and t' is defined in the usual way as the greatest common lower bound of t and t'. It is computed using a typed unification algorithm. A feature term is represented as a graph where each node has a type, an equivalence class used to represent equational constraints ("co-references"), and a set of outgoing arcs. The unification algorithm uses the union/find procedure on an inverted set representation of the equivalence classes adapted by [Aït-Kaci 84] after [Huet 76]. The actual algorithm used in the system is optimized using several different techniques in order to minimize copying and to behave as efficiently as a pattern-matcher in cases when no information needs to be added [Emele 1991].

2.3 Inheritance network of feature terms

Type equations

An inheritance network of feature terms is specified by a set of type definitions (type equations). A type definition has the following form: the type symbol to be defined appears on the left-hand side of the equation. The right-hand side is an expression of conjunctions and disjunctions of typed feature terms. Conjunctions are interpreted as meets on typed feature terms. The definition may have conditional constraints expressed as a logical conjunction of feature terms. These conditions are introduced by ':-'. The right-hand side feature term may contain the left-hand side type symbol in a subterm (or in the condition), thus defining a recursive type equation which gives the system the expressive power needed to describe complex linguistic structures. A simple example of a type definition is shown in Figure 1, and the corresponding partial order is displayed in Figure 2 using Hasse diagrams.

A subtype inherits all constraints of its super-types monotonically: the constraints expressed as feature terms are conjoined using typed unification; the conditions are conjoined using the logical conjunction. The compiler makes sure that we have specified an inheritance hierarchy, building an

```
LIST = NIL | CONS.
CONS = [first: T, rest: LIST].

APPEND0 = APPEND[1: NIL, 2: #1= LIST, 3: #1].
APPEND1 = APPEND[1: CONS[first: #x, rest: #11],
                 2: #12=LIST,
                 3: CONS[first: #x, rest: #13]]
          :- APPEND[1: #11, 2: #12, 3: #13].
```

Figure 1: Type definitions for LIST and APPEND using the TFS syntax.

internal representation where for any two types such that $A \leq B$ we have $def(A) \leq def(B)$. If this cannot be done, the hierarchy is inconsistent and an error is reported.

In a feature type definition $s = F$, the equality symbol that separates the left-hand side type symbol s from the right-hand side feature term F is interpreted as an equivalence. Assume that $\underline{F}$ is the meet of F and all feature type definitions of the supertypes of s. The two directions of the biconditional are:

$\Rightarrow$ all feature terms of type s are subsumed by the feature term $\underline{F}$;

$\Leftarrow$ all feature terms which are subsumed by $\underline{F}$ are of type s.

The $\Leftarrow$ part of the condition is checked statically at compile-time, and the $\Rightarrow$ part is checked dynamically at run time. A feature term where all subterms obey these two typing constraints is a well-typed feature term.

Closed terms and static type inference

Typed feature terms are always interpreted as *closed terms*: this is a consequence of the fact that a type definition defines an equivalence relation. All features of a feature term associated with a type are declared to be valid for that type and for all its subtypes only. This is a closure constraint: it is not possible to add throught unification (which is the only operation used on these data structures) an arbitrary feature to a term during computation, but only those declared for the type of the term. Given the information that all features are declared for a type and possibly have specific value restrictions, the compiler is able to infer the most specific type for any given feature term. Before presenting the inference rule for checking the $\Leftarrow$ condition, let us introduce a logical notion for feature terms.

A logical feature expression is an expression of either of the form:

$$X{:}A$$
$$X \doteq Y$$
$$X.l \doteq Y$$
$$\phi_1 \wedge \phi_2$$

where X and Y are variables, A is a type symbol, l is a feature and ϕ_i are logical expressions. The first three expressions are read "X is of type A", "X is equal to Y" and "the value of the feature l of X is Y". The operator $\wedge$ is the logical conjunction. A term $\#x{=}A[f_1{:}B_1, f_2{:}B_2]$ will be written as the expression $X{:}A \wedge X.f_1 \doteq Y_1 \wedge X.f_1 \doteq Y_2 \wedge Y_1{:}B_1 \wedge Y_2{:}B_2$.

The inference rule for checking the $\Leftarrow$ condition is defined as follows. With each feature f which appears in the definition of type A and which has the type restriction B, we associate the expression $X{:}\,A \wedge X.f \doteq Y \wedge Y{:}\,B$. With every feature which does not appear in the definition of any type, we associate the expression $X{:}\,\bot \wedge X.f \doteq Y \wedge Y{:}\,\bot$. The collection of those expressions for a set of type definitions for feature f is

$$\bigvee_i X_i{:}\,A_i \wedge X_i.f \doteq Y_i \wedge Y_i{:}\,B_i$$

Then, a term is correctly typed if adding this type information using the following rule preserves consistency (the result should be different from $\bot$):

$$\frac{X.f \doteq Y \qquad\qquad \bigvee_i X_i{:}\,A_i \wedge X_i.f \doteq Y_i \wedge Y_i{:}\,B_i}{X.f \doteq Y \;\wedge\; \bigvee_i X \doteq X_i \wedge Y \doteq Y_i \wedge X_i{:}\,A_i \wedge X_i.f \doteq Y_i \wedge Y_i{:}\,B_i}$$

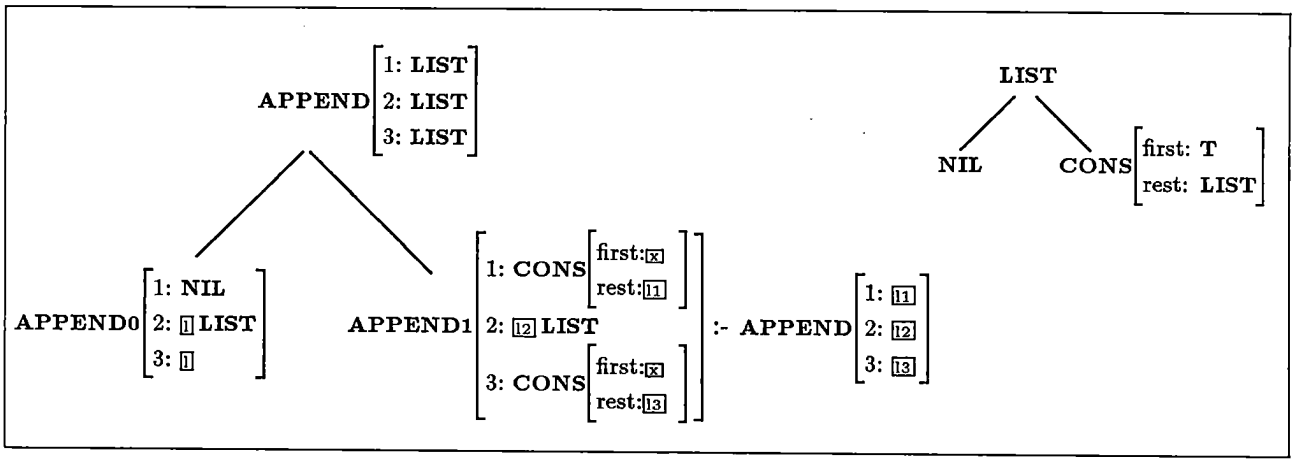


Figure 2: Type hierarchy for LIST and APPEND ($\top$ and $\bot$ omitted).

2.4 The meaning of typed feature structures

A ground feature term is a term where all type symbols in this term are minimal types (the lowest types in the hierarchy immediatly above $\bot$). A well-typed feature term is a term which obeys the type equivalence relation defined by the set of type definitions. The meaning (denotation) of a feature term is represented by the set of all well-typed ground feature terms which are subsumed by it. If this set is empty, the feature term is inconsistent with regard to the set of type definitions. The denotation can be finite, e.g. in the case of a dictionary, but it can also be infinite in the case of recursive types: for example, the set of ground terms subsumed by LIST is the set of all possible ground lists. However, the symbol LIST is itself the best *finite approximation* of the *infinite set* of all possible lists. Thus, instead of enumerating all ground feature terms of the denotation of a non-ground feature term, we will describe its denotation by a set of *appproximations*, themselves represented as feature structures.

Evaluation in the TFS system amounts to finding the necessary and sufficient conditions for describing exactly the denotation of an input feature term (the "query") modulo a set of type equations. Given an inheritance network, the necessary conditions are always checked in a finite amount of time: these are basically type constraints, and the procedure is the same as the one used at compile time to check the consistency of the network. But this is not enough, since the input term will usually describe

a subset (maybe empty) of the denotation of its type. Thus, we also want to find sufficient conditions that describe this subset. These conditions will be produced as specializations of the query: a set of well-typed feature terms which are subsumed by the input term.

2.5 The TFS abstract rewrite machine

In this section, we describe an abstract rewrite machine for evaluating feature terms. The rewrite mechanism is based on a variant of narrowing adapted to feature terms.

A set of type definitions defines an inheritance hierarchy of feature terms which specifies the available approximations. Such a hierarchy is compiled into a rewriting system as follows: each direct link between a type A and a subtype B generates a rewrite rule of the form $A[a] \rightarrow B[b]$ where $[a]$ and $[b]$ are the definitions of A and B, respectively.

The interpreter is given a "query" (a feature term) to evaluate: this input term is first type-checked. If it is consistent, it is already an approximation of the final solution, although a very rough one. The idea is then to incrementally add more information to that term using the rewrite rules in order to get step by step closer to the solution: we stop when we have the best possible approximation.

A rewrite step for a term t is defined as follows: if u is a subterm of t and u is of type A, and there exists a rewrite rule $A[a] \rightarrow B[b]$ such that $A[a] \wedge u \neq \bot$, then the right-hand side $B[b]$ is unified with the subterm u, giving a new term t' which is more specific than t. Rewrite steps are applied non-deterministically everywhere in the term until all types are minimal and no further rule is applicable[2].

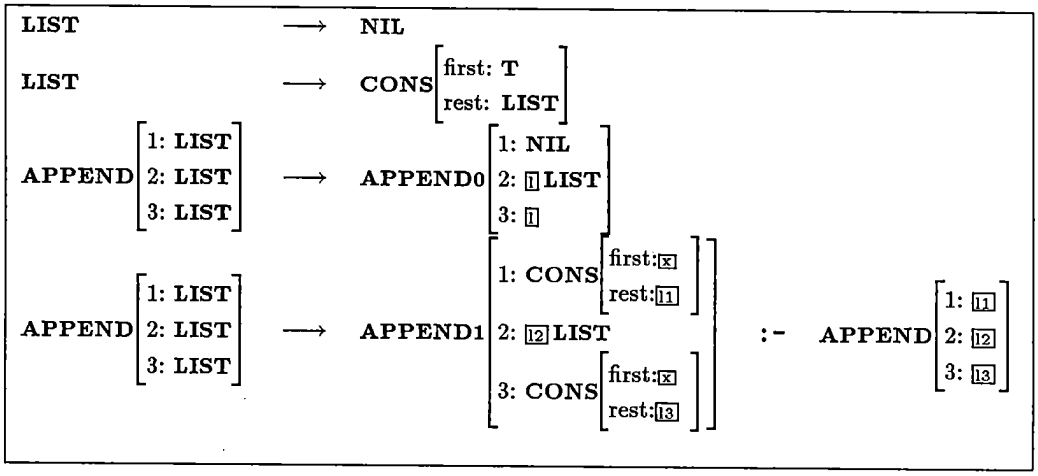


Figure 3: Rewrite rules for LIST and APPEND.

Actually, the rewriting process stops either when no rule is applicable or when all subterms in a term correspond exactly to some approximation defined by a type in the hierarchy and the fixed-point is reached. A term is "solved" when any of its subterms is either

- more specific than the definition of a minimal type, or

[2]Conditions do not change this general scheme (they are evaluated using the same rewriting mechanism) and are omitted from the presentation here for the sake of simplicity. See for example [Dershowitz/Plaisted 88] on conditional rewrite systems, and [Klop 90] for a survey.

- does not give more information than the definition of its type.

This defines an *if and only if* condition for a term to be a solved-form, where any addition of information will not bring anything new. This condition is implemented using a *lazy rewriting* strategy: the application of a rule at a subterm is triggered only when the subterm gets more specific than the left hand-side of the rule. More precisely, given a subterm u of type A and a rewrite rule $A[a] \rightarrow B[b]$, the application of the rule is triggered if $A[a] \wedge u \leq A[a]$.

This lazy rewriting strategy implements a fully data-driven computation scheme and avoids useless branches of computation. Thus, there is no need to have a special treatment to avoid what corresponds to the evaluation of uninstantiated goals in PROLOG, since a general treatment based on the semantics of the formalism itself is built into the evaluation strategy of the interpreter.

The lazy evaluation mechanism has an almost optimal behavior on the class of problems that have an exponential complexity when using the "generate and test" method. It is driven by the availability of information: as soon as some piece of information is available, the evaluation of constraints in which this information appears is triggered. Thus, the search space is explored "intelligently", never following branches of computation that would correspond to uninstantiated PROLOG goals[3].

The choice of which subterm to rewrite is only partly driven by the availability of information (using the lazy rewriting scheme). When there are several subterms that could be rewritten, the computation rule is to choose the outer-most ones (inner-most strategies are usually non-terminating). If there are several outer-most candidates, they are all rewritten in parallel. This is in contrast with PROLOG which evaluates the goals in a clause one after the other, from left to right[4]. Such a parallel outer-most rewriting strategy has interesting termination properties, since there are problems on which any fixed order of rewriting would not terminate where a parallel strategy does[5].

For a given subterm, the choice of which rule (actually which set of rules) to apply is done non-deterministically, and the search space is explored depth-first using a backtracking scheme. This strategy is not complete, though in association with the parallel outer-most rule and with the lazy evaluation scheme, it terminates on "well-defined" problems (a complete breadth-first search strategy can be used for debugging purposes).

3 Language design issues

In this section, we discuss some issues in the design of a language based on feature structures and inheritance. We distinguish between two broad levels in such a language: the object level, related to feature structures, their syntax and semantics, and the network level, related to the organization of feature structures into an inheritance hierarchy of types of feature structures. Since this basis provides a very general framework with all necessary computational expressive power, any enhancement or extensions (other than simple syntactic sugar) can be done by providing new type constructors with the associated unification algorithms.

[3]The lazy evaluation mechanism is not yet fully implemented in the current version (March 1991) of TFS, but with the partial implementation we have, a gain of 50% in speed for parsing has already been achieved in comparison with the previous implementation that used only the outer-most parallel rewriting strategy).

[4]This parallel rewriting strategy is similar to hyper-resolution in logic programming. The lazy evaluation mechanism is related to the 'freeze' predicate of, e.g. Prolog-II and Sicstus Prolog, though in Prolog, it has to be called explicitly.

[5]E.g. the problem of left-recursive rules in naive implementations of DCGs.

3.1 The network level

Using the TFS syntax, there are two means of specifying an inheritance network. The first is to use disjunctive specifications:

```
BOOL = TRUE | FALSE.
```

This expression specifies that the type BOOL covers the two types TRUE and FALSE. Moreover, this expression specifies exaustively the set of types covered by BOOL. Using disjunctive specifications, the network is described in a top-down fashion.

It is also possible to use conjunctive specifications to describe a network in a bottom-up fashion.

```
GREEN = BLUE & YELLOW.
```

In the current TFS syntax, the two means can be combined with an interpretation which is unfortunatly not very intuitive. The following set of definitions

```
A = A1 | A2.
B = B1 | B2.
X = A & B & C & D.
```

is equivalent to

```
A = A1 | A2.
B = B1 | B2.
X = (A1 | A2) & (B1 | B2) & C & D.
```

and a partial order is derived using the factorization of the disjunctive normal form. This allows to specify in a concise way cross-classifications that would need to be expanded explicitly otherwise.

Specialized syntax can be used for specifying networks. Two linguistic examples are HPSG inheritance hierarchies [Pollard/Sag 87] and Systemic Networks (see e.g. [Mellish 88]). [Carpenter 91] shows how partial orders can be extracted from such kinds of specifications.

The simplest and most general means of specifying an inheritance network is to use a covering relation: x covers y, notated $x \rhd y$, when there is no z different from x and y such that $x \leq z \leq y$. A unique partial order is constructed as the transitive closure of the covering relation. If the set of elements in the network is finite, it is always possible to embed such a partial order into a lattice that preserves the meets using a powerset construction, as used in the TFS system.

3.2 The object level

There are three different kinds of syntax for describing feature structures: path equations as in PATR-II (Figure 4), logical expressions of the feature logic (Figure 5), and feature terms using the AVM notation as used in the TFS syntax (Figure 6).

$$
\begin{array}{l}
\langle\rangle == PERSON \\
\langle name\rangle == ID \\
\langle name\ first\rangle == Jo \\
\langle name\ last\rangle == Brown \\
\langle spouse\rangle == PERSON \\
\langle spouse\ name\rangle == ID \\
\langle spouse\ name\ first\rangle == Judith \\
\langle spouse\ name\ last\rangle == \langle name\ last\rangle \\
\langle spouse\ spouse\rangle == \langle\rangle
\end{array}
$$

Figure 4: Path equations.

$$
\begin{array}{l}
X_0{:}PERSON \wedge \\
X_0.name \doteq X_1 \wedge X_1{:}ID \wedge \\
X_1.first \doteq X_2 \wedge X_2{:}Jo \wedge \\
X_1.last \doteq X_3 \wedge X_3{:}Brown \wedge \\
X_0.spouse \doteq X_4 \wedge X_4{:}PERSON \wedge \\
X_4.name \doteq X_5 \wedge X_5{:}ID \wedge \\
X_5.first \doteq X_6 \wedge X_6{:}Judith \wedge \\
X_5.last \doteq X_3 \wedge \\
X_4.spouse \doteq X_0
\end{array}
$$

Figure 5: Expression of a feature logic.

The TFS syntax allows any expression in place of a term. A TFS expression is of either of the form

$$
\begin{array}{l}
e1\ \&\ e2 \\
e1 \mid e2 \\
e1 \setminus e2 \\
e1 :- e2
\end{array}
$$

where & is the meet operator, | the join operator, \ the relative pseudo-complement operator, and :– the conditional operator.

3.3 Linking the two levels: network of objects

The inheritance network of typed feature strutcures is built from a partial order on type symbols and an equivalence relation between type symbols and feature structures specified as a type equation. For example, the TFS definition

```
APPENDO = APPEND[1: NIL, 2: #1=LIST, 3: #1].
```

specifies in a single expression these two kinds of information[6]:

[6]The logical interpretation of the covering relation is the implication: $X{:}APPEND0 \Rightarrow X{:}APPEND$.

```
#x=PERSON[name:   ID[first: Jo,
                      last: #y=Brown],
          spouse: PERSON[name: ID[first: Judith,
                                  last: #y],
                         spouse: #x]]
```

Figure 6: Feature term.

$$APPEND0 \lhd APPEND$$

$$X{:}APPEND0 \Leftrightarrow X.1 \doteq U \wedge U{:}NIL \wedge X.2 \doteq V \wedge V{:}LIST \wedge X.3 \doteq V$$

Furthermore, the axioms on the domain and image restrictions for features are also extracted from the same definition:

- $X.1 \doteq Y \wedge X{:}APPEND0 \wedge Y{:}NIL$
- $X.2 \doteq Y \wedge X{:}APPEND0 \wedge Y{:}LIST$
- $X.3 \doteq Y \wedge X{:}APPEND0 \wedge Y{:}LIST$

This choice leads to a very concise syntax. It could nevertheless be interesting to separate the specification of the partial order on type symbols ¿from the specification of the equivalence relation. This separation would allow a more modular scheme of compilation. The partial order on type symbol could be defined using different kinds of syntax such as the covering relation, the Systemic Network notation or HPSG hierarchies, and specialized graphical interfaces could be used without changing the syntax for the type definition (equivalence relation and feature restrictions). Another advantage is that a type definition can be recompiled without necessarily recompiling the entire network.

3.4 Extensions

There are three ways of extending the language. The first one is to provide additional notation with the same underlying semantics, e.g. logical expressions. The second is to provide a library of pre-defined types that could be defined in the language, but could be implemented in a more efficient way. The last is to provide new types that cannot be defined in the language, together with the associated unification algorithms.

Logical expressions

Since the feature term and the logical notations are equivalent (see e.g. [Aït-Kaci/Podelski 90]), we can use both in our concrete syntax. The compiler will then compile these mixed expressions into a unique normal form used by the interpreter. Changes are located at the compiler level and no modification of the interpreter is required. Logical expressions are specially useful to express complex conditions that are more difficult to specify using only the meet, join and complement operators.

Built-in types

Some types commonly used can be specified directly in the TFS language, but they can advantageously be defined as built-in types and be more efficiently implemented. Examples of such types are characters, strings of characters, or numbers, together with their associated operations (concatenation, addition, etc.). Thus, the semantic of such extensions can be precisely defined in the TFS language itself, and their implementation realized in the underlying implementation language. The interface to these types can then be defined and documented using the TFS syntax, and a specific syntax could be used for ground objects.

For example, we could define a type `STRING` as:

```
STRING = EMPTY-STRING | NON-EMPTY-STRING.
NON-EMPTY-STRING = [first: ASCII, rest: STRING].
```

which defines both the interface and the semantics of that type. A LISP implementation could use directly LISP lists, and the general unification algorithm can be modified to work directly on lists instead of the general internal representation for feature terms. Extensions are located at the compiler level and for efficiency purposes, also in the unification algorithm. A syntax for ground strings is:

```
EMPTY-STRING                                  → ""
NON-EMPTY-STRING[first: a, rest: EMPTY-STRING] → "a"
```

Semantic extensions

Extensions that cannot be easily expressed in the language itself are more significative. For example, a type string where the concatenation operator is associative needs associative unification (this problem is also known as the resolution of word equations), which significantly more complex than the unification on feature terms. Thus, this kind of extension lead to non trivial changes in the language and its implementation. Such a string type would be useful for expressing, for example, morphological patterns as string patterns.

An associative unification algorithm (see e.g. [Abdulrad/Pécuchet 89]) would solve the equation $cons(a, X) = cons(Y, b)$, where the set of strings beginning with an a is equated with the set of strings ending with b, the solution describing the set of strings beginning with a and ending with b. Using the axiom of associativity

$$cons(cons(X, Y), Z) = cons(X, cons(Y, Z))$$

and introducing a new variable, we have $cons(a, cons(Z, b) = cons(cons(a, Z), b)$. The set of substitution describing the answer to the problem is

$$\{X = cons(Z, b),\ \ Y = cons(a, Z)\}$$

Extensions which can be treated as the introduction of a new built-in type, even if they modify the semantics of the language, modify it in a coherent and conservative way. They can be accomodated without changing the basic design of the language.

Conclusion

The TFS language is a formalism based on typed feature structures and it synthezises some of the key concepts of object-oriented languages (abstraction and multiple inheritance, complex objects, classification) and relational languages (logical variables, non-determisnism with backtracking, existential queries). The language is fully declarative: the notions of inheritance, type checking, classification, and evaluation are all based on unification of feature structures extended with types.

Based on feature structures, the language is expressive enough to accomodate very different styles of grammars, and as a "lingua franca" for computational linguistics, it can help to bridge the gap between different approaches to linguistic description.

References

[Abdulrad/Pécuchet 89] Habib Abdulrad and Jean-Pierre Pécuchet. "Solving word equations". *In* Claude Kirchner (ed.), *Unification*, Academic Press, 1990.

[Aït-Kaci 84] Hassan Aït-Kaci. *A Lattice Theoretic Approach to Computation based on a Calculus of Partially Ordered Types Structures.* Ph.D Dissertation, University of Pennsylvania.

[Aït-Kaci 86] Hassan Aït-Kaci. "An Algebraic Semantics Approach to the Effective Resolution of Type Equations". *Theoretical Computer Science 45*, 293-351.

[Aït-Kaci/Podelski 90] Hassan Aït-Kaci and Andreas Podelski. "Is there a meaning to LIFE?". Submitted to *ICLP'91.*

[Borgida et al. 89] Alexander Borgida, Ronald J. Brachman, Deborah L. McGuiness and Lori Alperin Resnick. "CLASSIC: a structural data model for objects". Proc. of the *1989 ACM SIGMOD International Conference on Management of Data*, Portland, Oregon, 1989.

[Bouma 90] Gosse Bouma. "Non-monotonic inheritance and unification". Proc. of the *Workshop on Inheritance in Natural Language Processing*, Institute for Language Technology and AI, Tilburg University, Netherlands, August 1990.

[Brachman/Schmolze 85] Ronald J. Brachman and James G. Schmolze. "An overview of the KL-ONE knowledge representation language". *Cognitive Science* 9, 171-216, 1985.

[Carpenter 90] Bob Carpenter. "Typed feature structures: inheritance, (in)equality and extensionality". Proc. of the *Workshop on Inheritance in Natural Language Processing*, Institute for Language Technology and AI, Tilburg University, Netherlands, August 1990.

[Carpenter 91] Bob Carpenter. "The logic of typed feature structures". Ms., Philosophy Department, Carnegie Mellon University.

[Daelemans 90] Walter Daelemans. "Inheritance and object-oriented natural language processing". Proc. of the *Workshop on Inheritance in Natural Language Processing*, Institute for Language Technology and AI, Tilburg University, Netherlands, August 1990.

[Dershowitz/Plaisted 88] N. Dershowitz and D.A. Plaisted. "Equational programming". *In* Hayes, Michie and Richards (eds.). *Machine Intelligence 11.* Clarendon Press, Oxford, 1988.

[De Smedt/de Graaf 90] Koenraad De Smedt and Josje de Graaf. "Structured inheritance in frame-based representation of linguistic categories". Proc. of the *Workshop on Inheritance in Natural Language Processing*, Institute for Language Technology and AI, Tilburg University, Netherlands, August 1990.

[Emele 1988] Martin Emele. "A typed feature structure unification-based approach to generation". Proc. of the *WGNLC of the IECE*, Oiso University, Japan, 1988.

[Emele 1991] Martin Emele. "Unification with lazy non-redundant copying". Submitted to the *29th Annual Meeting of the ACL*, Berkeley, June 1991.

[Emele/Zajac 1989a] Martin Emele and Rémi Zajac. "RETIF: A Rewriting System for Typed Feature Structures". ATR Technical report TR-I-0071, ATR, Kyoto.

[Emele/Zajac 1989b] Martin Emele and Rémi Zajac. "Multiple Inheritance in RETIF". ATR Technical report TR-I-0114, ATR, Kyoto.

[Emele/Zajac 90a] Martin Emele and Rémi Zajac. "A fixed-point semantics for feature type systems". Proc. of the *2nd Workshop on Conditional and Typed Rewriting Systems – CTRS'90*, Montreal, June 1990.

[Emele/Zajac 90b] Martin Emele and Rémi Zajac. "Typed Unification Grammars". Proc. of the *13th International Conference on Computational Linguistics – COLING'90*, Helsinki, August 1990.

[Emele et al. 90] Martin Emele, Ulrich Heid, Stefan Momma and Rémi Zajac. "Organizing linguistic knowledge for multilingual generation". Proc. of the *13th International Conference on Computational Linguistics – COLING'90*, Helsinki, August 1990.

[Evans/Gazdar 89] Roger Evans and Gerald Gazdar. "Inference in DATR". Proc. of the *4th European ACL Conference*, Manchester, 1989.

[Franz 90] Alex Franz. "A parser for HPSG". CMU report CMU-LCL-90-3, Laboratory for Computational Linguistics, Carnegie Mellon University, July 1990.

[Frazer/Hudson 90] Norman M. Frazer and Richard A. Hudson. "Word Grammar: an inheritance-based theory of language". Proc. of the *Workshop on Inheritance in Natural Language Processing*, Institute for Language Technology and AI, Tilburg University, Netherlands, August 1990.

[Gòguen/Meseguer 87] Jospeh A. Goguen and José Meseguer. "Unifying Functional, Object-Oriented and Relational Programming with Logical Semantics". CSLI Report CLSI-87-93, Stanford University, 1987.

[Huet 76] Gérard Huet. *Résolution d'équations dans les langages d'ordre 1, 2, ..., ω*. PhD thesis, Université de Paris VII, September 1976.

[Klop 90] Jan Willem Klop. "Term rewriting systems". To appear in S. Abramsky, D. Gabbay and T. Maibaum. *Handbook of Logic in Computer Science*, Vol.1, Oxford University Press.

[Lécluse/Richard/Velez 88] Christophe Lécluse, Philippe Richard, Fernando Velez. "O_2, an object-oriented data model". Proc. of the *ACM SIGMOD Conference*, Chicago, Illinois, June 1988.

[Mac Gregor 88] Robert M. Mac Gregor. "A deductive pattern matcher". Proc. of the *National Conference on Artificial Intelligence – AAAI'88*, 403-408, St. Paul, MN, August 1988.

[Mac Gregor 90] Robert M. Mac Gregor. *LOOM User Manual.* USC/ISI Technical Report, La Jolla, CA, 1990.

[Mellish 88] Christopher S. Mellish. "Implementing Systemic Classification by Unification". *Computational Linguistics* 14/1, 1988, pp 40-51.

[Pollard/Sag 87] Carl Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics.* CSLI Lecture Notes 13, Chicago University Press, 1987.

[Pollard 90] Carl Pollard. "Sorts in unification-based grammar and what they mean". In M. Pinkal and B. Gregor (eds.), *Unification in Natural Language Analysis*, MIT Press. (in press)

[Pollard/Moshier 90] Carl Pollard and Drew Moshier. "Unifying partial descriptions of sets". In P. Hanson (ed.) *Information, Language and Cognition*, Vancouver Studies in Cognitive Science 1, University of British Columbia Press, Vancouver. (in press)

[Reape 90] Mike Reape. "Parsing semi-free word order and bounded discontinuous constituency and "shake 'n' bake" machine translation (or 'generation as parsing')". Presented at the *International Workshop on Constraint Based Formalisms for Natural Language Generation*, Bad Teinach, Germany, November 1990.

[Smolka 88] Gert Smolka. "A Feature Logic with Subsorts". LILOG Report 33, IBM Deutschland GmbH, Stuttgart.

[Smolka 89] Gert Smolka. "Feature Constraint Logics for Unification Grammars". IWBS Report 93, IBM Deutschland GmbH, Stuttgart.

[Smolka/Aït-Kaci 88] Gert Smolka and Hassan Aït-Kaci. "Inheritance Hierarchies: Semantics and Unification". *J. Symbolic Computation* 7, 343-370.

[van Hentenryck/Dincbas 87] P. van Hentenryck and M. Dincbas. "Forward checking in logic programming". Proc. of the *4th International Conference on Logic Programming*, Melbourne, May 1987.

[Zajac 89] Rémi Zajac. "A transfer model using a typed feature structure rewriting system with inheritance". Proc. of the *27th Annual Meeting of the ACL*, 26–27 June 1989, Vancouver.

[Zajac 90a] Rémi Zajac. "A relational approach to translation". Proc. of the *3rd International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Language*, 11–13 June 1990, Austin.

[Zajac 90b] Rémi Zajac. "Semantics of typed feature structures". Presented at the *International Workshop on Constraint Based Formalisms for Natural Language Generation*, Bad Teinach, Germany, November 1990.

Feature-Based Inheritance Networks for Computational Lexicons*

Hans-Ulrich Krieger
krieger@dfki.uni-sb.de

John Nerbonne
nerbonne@dfki.uni-sb.de

Deutsches Forschungszentrum für Künstliche Intelligenz
Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, Germany

September 3, 1991

Abstract

The virtues of viewing the lexicon as an inheritance network are its succinctness and its tendency to highlight significant clusters of linguistic properties. From its succinctness follow two practical advantages, namely its ease of maintenance and modification. In this paper we present a feature-based foundation for lexical inheritance. We shall argue that the feature-based foundation is both more economical and expressively more powerful than non-feature-based systems. It is more economical because it employs only mechanisms already assumed to be present elsewhere in the grammar (viz., in the feature system), and it is more expressive because feature systems are more expressive than other mechanisms used in expressing lexical inheritance (cf. DATR). The lexicon furthermore allows the use of default inheritance, based on the ideas of default unification, defined by Bouma [7].

These claims are buttressed in sections sketching the opportunities for lexical description in feature-based lexicons in two central lexical topics: inflection and derivation. Briefly, we argue that the central notion of paradigm may be defined directly in feature structures, and that it may be more satisfactorily (in fact, immediately) linked to the syntactic information in this fashion. Our discussion of derivation is more programmatic; but here, too, we argue that feature structures of a suitably rich sort provide a foundation for the definition of lexical rules.

We illustrate theoretical claims in application to German lexical structure. This work is currently under implementation in a natural language understanding project (DISCO) at the German Artificial Intelligence Center (Deutsches Forschungszentrum für Künstliche Intelligenz).

1 Introduction

The best inheritance mechanisms for representing lexical information have been Flickinger, Pollard and Wasow's [19] work on "structured lexicons", and Evans and Gazdar's [16] work on DATR.

*We thank Rolf Backofen, Stephan Busemann, Bob Carpenter, Bob Kasper, Andreas Kathol, Klaus Netter, Carl Pollard and Harald Trost for conversations about this work. We have also benefited from the reactions of audiences where we presented different parts of it (in the Spring of 1991), in particular at the AQUILEX Workshop, Cambridge; The ASL Workshop on DATR, Bielefeld; and the Linguistics Colloquium at The Ohio State University. This work was supported by a research grant, ITW 9002 0, from the German Bundesministerium für Forschung und Technologie to the DFKI DISCO project.

Both Flickinger's work and DATR aim to supplement feature-based grammars, but both require an explicit translation step to convert lexical information into grammatical features. Furthermore, they are both hampered in expressive power, so that they accommodate some sorts of information poorly, even information which is standardly found in feature systems, e.g., disjunction, negation, and complex feature structures used as values.

The present proposal draws both from the work above on default inheritance networks and from the lexical ideas of feature-based theories in general (cf. the section on PATR-II lexicons in Shieber, [43], pp.54-61). PATR-II present lexicons via a collection of TEMPLATES or MACROS, which are purely syntactic abbreviations for feature-structure descriptions. Pollard and Sag's (1987) sketch of the lexicon in Head-Driven Phrase Structure Grammar (hence HPSG), presented in Pollard and Sag [36] and Sag and Pollard [42], extended these ideas by interpreting lexical definitions as feature-structure descriptions and inheritance specifications as subsumption statements, rather than treating them in a purely syntactic fashion. Pollard and Sag (Chap.8.2) furthermore suggest a use of lexical rules which brings their work closer to standard linguistic views (e.g., LFG, [8]).

This and most other work on feature structures, on the other hand, has failed to allow the use of DEFAULTS or OVERWRITING, which is crucial for a practical lexical tool.[1] The key advantage of default specifications is that they allow the description of SUBREGULARITIES, classes of items whose properties are largely, but not perfectly regular. In a system with default inheritance, these may be regarded not as anomalous, but rather as imperfectly regular, or regular within limits. We shall employ default inheritance regularly, perhaps most crucially in the specification of derivational relations (cf. below and cf. Flickinger et al. [19]; and Gazdar [20], [21]; and Flickinger and Nerbonne [18] for arguments supporting the use of defaults in lexical specifications). This has seemed suspicious within the context of feature systems because these were developed (in part) to allow monotonic processing of linguistic information, and the use of defaults leads to nonmonotonicity.[2] But, as Bouma [7], p.169 points out, the use of lexical defaults is a fairly harmless form of nonmonotoncity, since the lexicon is nonmonotonic only with respect to lexical development—the syntactic use of information specified via lexical default leads to none of the problems associated with nonmonotonic reasoning; e.g., inferences about phrases never need to be retracted, and the NL system may be configured to be perfectly monotonic at run-time. If we employ default inheritance for the specification of lexical information, then the inheritance hierarchy does NOT correspond to a subsumption or subtyping hierarchy—information may be overwritten which renders subsumption invalid. Care needs to be taken that the two notions of hierarchy—the classes involved in the default inheritance relationship and the feature structure types defined there—not be confused (cf. Cook, Hill and Canning [13]). The mechanism we shall employ for the default combination of lexical information is the DEFAULT UNIFICATION developed by Bouma [7]; we may employ this within the lexicon, even while eschewing its use for parsing and generation.

The present work is closest to Pollard and Sag's in that it proceeds from a view of the lexicon in which feature structures bear the burden of linguistic description. It differs from their work in advocating the use of default inheritance, and perhaps more significantly, in using ONLY feature structure descriptions to represent lexical information, including especially lexical rules. Where Pollard and Sag viewed lexical rules as operators on feature structures, we propose defining lexical rules purely in terms of feature structures. On our view, one need not assume a distinct sort of linguistic entity, lexical rule, which maps feature structures into feature structures. Instead, one begins with feature structures and a description language for them, and this suffices to characterize lexical rules. Lexical rules are thus an emergent phenomenon in language, ultimately reducible to feature-structures (cf. Section 4 for a discussion of alternative views of lexical rules).

Our purpose in this paper is to present the feature-based lexicon not as a linguistic theory

[1] But cf. Pollard and Sag [36], p.194, Note 4; Sag and Pollard [42], p.24; and Shieber [43], pp.59-61.

[2] It is probably worth noting that there have nonetheless been several attempts at using nonmonotonic inference rules in unification-based systems. Kay's [30] FUG included an ANY value, which defaulted to $\perp$ unless unified with, in which case it was $\top$; Shieber [43], p.59 presents a scheme for the default interpretation of feature structure templates; Kaplan [25] presents nonmonotonic "constraining equations" for LFG; and Gazdar et al. [22], p.29 *et passim* propose nonmonotonic FEATURE SPECIFICATION DEFAULTS.

of lexical structure, but rather as a framework within which such theories may be formulated, i.e., a tool for lexical description. This means that we shall at points demonstrate the formulation of competing views of lexical phenomena, e.g., matrix-based and form-based views of the inflectional paradigm, and inflectional and derivational views of the passive. It is also worth noting that, although we are quite interested in the question of efficiently processing feature-based descriptions—especially the question of lexical access for generation and recognition, we will not have anything to say about it in this paper.

It is probably worth sketching here in broad strokes the sorts of advantages we shall claim for our proposals. These lie primarily in the expressive power of the feature formalisms. The added expressive capacity is exploited in order to characterize both inflectional paradigms and derivational word formation rules as first-class linguistic objects—feature structures—of the same sort as words or phrases. We believe this proposal is made here for the first time, certainly the first time in a formalized theory. We illustrate the advantages of the added expressive power in analyses of inflectional paradigms as disjunctive further specifications of abstract lexemes, and in analyses of derivationally complex forms as instantiations of rules, where the latter are given a feature-based interpretation.

Our proposals are couched in HPSG terms, both because this is often readily understandable but also because HPSG—and to a lesser extent, FUG (Kay, [29], [30])—have most vigorously explored the hypothesis that feature structures are a sufficient representation scheme for all linguistic knowledge. The idea is that feature structures (also called "functional structures" in FUG, "attribute-value matrices" in HPSG) can encode not only syntactic information, but also semantic, pragmatic, morphological and lexical information. In HPSG, even recursively structured aspects of syntax such as syntactic rules and phrase structure trees are ultimately characterized as constraints on feature structures, so that the attempt to construe lexical information strictly within the limits provided by feature structures would not seem inappropriate.

We commence from the treatment of the lexicon in HPSG I (Pollard and Sag, [36], Chap.8) as an inheritance structure, which we accept (with minor modifications noted below). Now, HPSG I invoked "lexical rules" as operations or functions ON feature structures, but this proposal in many ways violates the spirit which infuses HPSG, that of demonstrating that all linguistic knowledge can be modeled directly in feature structures. Our proposed construal of lexical rules as feature structures obviates any use of lexical rules as operations—also familiar from LFG (Bresnan, [8]) and PATR-II (Shieber et al., [44]), especially D-PATR (Karttunen, [26]).[3] We believe therefore that the present paper is a contribution to feature-based theories, as well, in that it shows how lexical rules can be construed in terms of feature-structure. This reduces the theoretical inventory, which is by itself desirable, and, as we argue below, it has significant descriptive advantages as well.

The structure of the paper is as follows: we first continue the introduction with a discussion of the distinction between morpheme- and lexeme-based analyses. In Section 2, we summarize the use we shall make of feature structures, which is essentially that provided by HPSG. We then explore the advantages of feature-based lexicons in two central lexical topics: the treatment of inflection and derivation. (There is, of course, an influential school of linguistic thought which denies the significance of this distinction, and our tools do not presuppose the distinction—we too can generalize all lexical rules to "word formation" à la Sadock's "autolexical syntax" [41]. But feature structures offer interesting alternatives for inflection.)

1.1 Lexemes and Morphemes

Before we examine the feature-based analysis of inflection and derivation in more detail, it is worth reminding ourselves that the developing research paradigm of computational lexicology—within which this paper might be located—differs from computational morphology (e.g., two-level or finite-state morphology) in that the former takes a LEXEME-based view of lexical variation (such

[3] And nearly everywhere else: GPSG metarules (Gazdar et al., [22]) and Categorial Morphology treatments (Hoeksema, [23]) are quite similar in treating lexical rules as fundamentally distinct from lexical entries.

as inflection and derivation), while the latter generally takes a MORPHEME-based view. While we do not aim here to settle any debate over which is preferable, it may be useful to sketch the differences.[4]

A lexeme is an abstract unit which characterizes what is common among the infectional variants of a word. It contains, for example, what is common among the variants of the verb *institute (institutes, instituting, instituted)*, etc, but NOT *institution, institutional, reinstitute,...*, all of which are related, but distinct lexemes. Note that the lexeme is an abstract unit, which need not even be associated with any particular form. The lexeme-based view of lexical processes such as inflection and derivation maintains that these processes are based on properties of lexemes. We clarify this below after sketching the morpheme-based alternative. A morpheme is also an abstract unit which is the minimal unit to which meaning is attached. Note that a lexeme may contain meaningful parts, e.g., *ab-* + *leit-* + *bar*—this is a lexeme with its own set of inflectional variants (the case, gender, number, grade paradigm of German adjectives). The question of whether lexical processes are morpheme-based or lexeme-based thus centers on the analysis of such complex words. Are these optimally analyzed solely in terms of component morphemes (morpheme-based), or must one take into account intermediate lexemes (lexeme-based)? In the example *ableitbar,* can one derive all properties from *ab-*, *leit-* and *-bar*, or must one take into account the intermediate lexeme *ableiten*?

Returning to our English example to clarify further the distinction between the morpheme-based and lexeme-based views, we note that it is very unlikely that *institute* could be divided into further meaningful parts, so that we may examine the morpheme *institute* (which constitutes all of the material of the lexeme above), and which also appears in *institution, institutional, reinstitute, institutionalize...* As we see, a single morpheme may be shared by many lexemes (but never *vice versa*). The morpheme-based view analyzes lexical processes as depending on the properties of morphemes.

There is a fairly clear division among the practitioners of each type of analysis. Linguists are fairly unanimous in seeing lexical processes as lexeme-based, while computational linguists have generally conducted morpheme-based analyses. The computational advantage of morpheme-based analysis is fairly easy to see: one can develop procedures for isolating the morphemes in a given form, and assign properties to the form based on these. Lexemes, involving (as they potentially do) nontransparent combinations of morphemes, are more difficult to recognize, are therefore more frequently stored *in toto*, and are thus ultimately more demanding of memory resources—there are simply many more lexemes than morphemes (perhaps an order of magnitude more).

Linguists have been assiduous in turning up cases where the larger number of lexemes seems useful in analyzing lexical processes. The suffixation of *-bar* in German, which we examine below in detail, provides examples of the most important sorts of cases.[5] The crucial observation is always of the form that important properties of complex words depend NOT on the morphemes of the complex, but rather on the lexemes. Wellformedness is one such important property. And it turns out that the wellformedness of forms with the suffix *-bar* cannot be predicted only on the basis of the component morphemes. We find patterns such as the following:

[4] Cf. Matthews [35] pp.20ff for a much more thorough defense of this material; cf. Anderson [2], [3] and Zwicky [49], [50] for more recent defenses of the lexeme-based view.

[5] A great deal of what we analyze below may be found in a very thorough study of this process, Jindrich Toman's *Wortbildung* [47].

morpheme	lexeme 1	lexeme 2
meid-	**meidbar*	*vermeidbar* 'avoidable'
lad-	**ladbar*	*aufladbar* 'loadable'
hab-	**habbar*	*handhabbar* 'manageable'
stell-	**stellbar*	*einstellbar,* 'adjustable'
		vorstellbar 'imaginable'
meß-	*meßbar* 'measurable'	**bemeßbar*
arbeit-	**arbeitbar*	*bearbeitbar* 'workable'

This table demonstrates that the wellformedness of *-bar* derivatives cannot be predicted on the basis of the stem morphemes with which they combine, and the last pair suggests that prefix morphemes are likewise poor predictors. (This is true, but we won't adduce further evidence here. It is trivial (but very tedious) for a German speaker with a dictionary to collect this evidence.) It is worth anticipating one reaction to the evidence above: one might object that there are perfectly respectable accounts of the illformedness of some of the examples here—some are not derived from transitive verbs, for example. But this sort of objection is merely a more detailed diagnosis of the same problem: the illformedness arises when the lexeme source of a derived word is not transitive. There is no claim made here that these patterns are ultimately inexplicable, merely that they are inexplicable on the basis of morphemic analysis. The hypothesis of a lexeme base enables more exact hypotheses in this area (e.g., about transitivity).

A second important property of derived words is their meaning, and here again, the best predictor of derived meaning is the meaning of component lexemes rather than that of component morphemes. We can illustrate the evidence for this using the same suffix, *-bar*. The general meaning accruing to a *-bar* adjective is that it describes the property of something which could stand (in object postion) in the relation denoted by the verb. Thus something is *faxbar* if one can fax it. The cases which demonstrate the lexeme basis of derivation all involve some irregularity in meaning. For example, German *eßbar* 'edible' involves a slight narrowing in meaning from the verb *essen* 'eat', since something must be capable of being safely eaten to be *eßbar*. More complicated derivatives involving these two morphemes could either preserve this narrowing, in which case they would appear to support the hypothesis of a lexeme basis, or they might fail to preserve it. The example of *Eßbarkeit* 'edibility' indicates that the narrowing is exactly preserved. And in general, meaning changes are persistent in further derivation. For a further example, note *Kostbarkeit* 'valuableness' derived from the semantically irregular *kostbar* 'valuable'.

A third important property of derived words is their form, and form is likewise lexeme-dependent. The form of the argument here is the same as that above: irregularity is persistent throughout further derivation. Thus irregular *sichtbar* 'visible' (instead of **sehbar*) is found further in *Sichtbarkeit.*[6]

It is worth noting that *-bar* suffixation is a derivational process, and that most of the evidence for a lexeme basis has been accummulated from studies of derivation. We shall likewise present a lexeme-based treatment of inflection below, however. The same sorts of evidence for a lexeme basis may be adduced in the case of inflection, but rather less of it than in the case of derivation. There are, e.g., verbs whose paradigms differ from those of their component morphemes, e.g., the weak verbs *handhaben* and *veranlassen*, which are derived from the strong *haben* and *lassen*. If inflection depends on morphemes, these examples must be analyzed as involving distinct pairs of morphemes.[7] Matthews [35], Chap. VIII, presents arguments of a different sort that inflection should not be reduced to manipulations of morphemes.

[6] One would expect irregular syntactic properties to show the same persistence through derivation, but we do not know of relevant studies.

[7] A further sort of example may be forthcoming if one examines perfect participle formation in German. The generalization to be captured is that a prefix *ge-* is employed when stress is on the first stem syllable (stem or inseparable prefix) of the lexeme. The argument is complicated by the fact that this is often—perhaps always—predictable on the basis of the morphemes involved.

We have continued at some length to justify our choice of lexeme-based analysis here, in order to emphasize that this is a deliberate, and not merely a customary assumption. Morpheme-based work has its purpose under this scheme, however. In particular, since our treatment of derivation allows regular derivations not to be listed in the lexicon, we need a method of recognizing the parts of a regular derivation in order to assign the correct properties to it. As we see it, a fully regular derivation such as *faxbar* 'faxable' need not appear in the lexicon at all. Its properties are fully specified under the specifications for the lexeme *fax-* 'to telefax', the suffix *-bar*, and the morpholological head-complement rule scheme. But in order to recognize *fax-* and *-bar* in *faxbar*, some analysis must be performed, and morpheme-based processing seems well-suited for this purpose.

1.2 Morphotactics and Allomorphy

There is a traditional distinction in morphology between MORPHOTACTICS, the arrangement of morphological elements into larger structures, and ALLOMORPHY or MORPHOPHONEMICS, variations in the shape of morphological units (cf. Anderson [2], p.147). It is our goal here to show how morphotactics may be subsumed into the lexicon, and we shall discuss this at lenghth in Sections 3 and 4 below. But we shall deliberately have very little to say about morphophonemics, which we do not intend to treat here. We take up this distinction more concretely in Section 3.1 below, but only to reinforce the point here: we do not propose subsuming morphophonemics into the lexicon.

2 Background

In this section we review the background material in the theory of feature-based description. This section may be skipped over by those familiar with feature structure theory and HPSG.

2.1 Feature Structures

The fundamental analytical tool of FEATURE-BASED or UNIFICATION-BASED theories is the feature structure, represented by the attribute-value matrix (AVM)—a set of pairs of ATTRIBUTES (such as PER) and VALUES (such as FIRST, SECOND or THIRD). Shieber [43] is the standard introductory reference to feature-based grammars, and we assume basic familiarity with this sort of analysis; here we review only the bare essentials needed for the lexical analysis below. Feature structures are mathematical objects which model linguistic entities such as the utterance tokens of words and phrases. It is important to note that values may themselves be feature structures, so that we allow that an AGR attribute may specify its value using the complex AVM below, resulting in a hierarchical AVM:

$$\left[\text{AGR} \left[\begin{array}{l} \text{PER THIRD} \\ \text{NUM SG} \end{array} \right] \right]$$

A useful conceptualization of feature structures is that of rooted, directed labeled graphs, where values correspond to nodes, attributes to labeled edges, and where the AVM as a whole describes the root. This conceptualization is particularly useful when it comes to specifying values within complex structures, which we do by concatenating attributes to form paths from the root into the interior of the structure. Because AVM descriptions can quickly become quite large, we will employ path descriptors, abbreviating, e.g., the person information in the AVM above to simply:

$$[\ \text{AGR|PER THIRD}\ ]$$

We shall even take the liberty occasionally of suppressing prefixes where no confusion arises, and specifying (as equivalent to the above):

$$[\ \text{PER THIRD}\ ]$$

But we shall take care to do this only where the type of the AVM is clear (so that it is the type of AVM in which AGR occurs, as opposed to one in which HEAD|AGR occurs, etc.). Such abbreviations are generally disambiguated when interpreted in the light of type information. They reduce the complexity of AVM's a great deal, making them easier to read and write.

We shall have frequent occasion to employ AVM's with DISJUNCTIVE value specifications. These are descriptions of objects whose value is included in one of the disjuncts, i.e., it is FIRST or THIRD:

$$[\ \text{AGR|PER}\ \{\text{FIRST, THIRD}\}\ ]$$

In order to link particular choices with formal elements, we make extensive use of DISTRIBUTED DISJUNCTIONS, investigated by Backofen, Euler and Görz [5] and Dörre and Eisele [14]. This technique was developed because it (normally) allows more efficient processing of disjunctions, since it obviates the need to expand them to disjunctive normal form. It adds no expressive power to a feature formalism (assuming it has disjunction), but it abbreviates some otherwise prolix disjunctions:

$$\begin{bmatrix} \text{PATH1: } \{_{\$1}\ a, b\} \\ \text{PATH2: } \{_{\$1}\ \alpha, \beta\} \\ \text{PATH3: } [\ \ldots\ ] \end{bmatrix} = \left\{ \begin{bmatrix} \text{PATH1: } a \\ \text{PATH2: } \alpha \\ \text{PATH3: } [\ \ldots\ ] \end{bmatrix}, \begin{bmatrix} \text{PATH1: } b \\ \text{PATH2: } \beta \\ \text{PATH3: } [\ \ldots\ ] \end{bmatrix} \right\}$$

The two disjunctions in the feature structure on the left bear the same name '$1', indicating that they are a single alternation. The sets of disjuncts named covary, taken in order. This may be seen in the right-hand side of the equivalence. Two of the advantages of distributed disjunctions may be seen in the artificial example above. First, covarying but nonidentical elements can be identified as such, even if they occur remotely from one another in structure, and second, features structures are abbreviated. The amount of abbreviation depends on the number of distributed disjunctions, the lengths of the paths PATH1 and PATH2, and—in at least some competing formalisms—on the size of the remaining structure (cf. [PATH3:] above).[8]

A final point to be appreciated about the use of feature structures is that two different attributes may be specified as having the same value, even when that value is unknown. For example, we might specify subject verb agreement in the following fashion, where the boxed numbers are just "tags" that identify the values as being the same:

$$\begin{bmatrix} \text{AGR } \boxed{1} \\ \text{SUBJECT } [\ \text{AGR } \boxed{1}\ ] \end{bmatrix}$$

Returning to the graph conceptualization above, the need for this sort of specification demonstrates that the class of graphs we're interested in are not simply trees, but objects of the more general class of directed graphs.

What we have written above are AVM's or FEATURE DESCRIPTIONS—they describe the abstract objects (feature structures) we use to model linguistic phenomena. Attribute-value descriptions such as the ones above are standardly interpreted in one of two ways: either directly, as descriptions of linguistic objects (cf. Johnson [24]; Smolka [46]), or algebraically, as specifications of FEATURE STRUCTURES (cf. Pollard and Sag, [36] Chap. 2), which then may be regarded as models of the linguistic objects. The distinction is mathematically interesting, but it will not be pursued here, since it is irrelevant to grammars and lexicons written in this notation. Indeed, we shall often speak informally of the AVM's as if they were the linguistic objects, as is common.

UNIFICATION is the normal means of combining the compatible information in two or more feature structures into a single one. Unification fails when there is incompatible information. We shall not provide a formal definition of the notion here, even though it is used frequently below, since it is defined in the works cited above. Here is an example of two AVM descriptions which are compatible, and a further example of two which are not:

[8] Cf. Backofen et al. [5] for a discussion of a third advantage of distributed disjunctions, namely a normal increase in processing efficiency.

$$[\ \text{AGR}\ [\ \text{PER THIRD}\ ]\ ] \sqcap [\ \text{AGR}\ [\ \text{NUM SG}\ ]\ ] = \left[\ \text{AGR}\ \begin{bmatrix}\text{PER THIRD}\\ \text{NUM SG}\end{bmatrix}\ \right]$$

$$\left[\ \text{AGR}\ \begin{bmatrix}\text{PER THIRD}\\ \text{NUM PL}\end{bmatrix}\ \right] \sqcap [\ \text{AGR}\ [\ \text{NUM SG}\ ]\ ] = \bot$$

(Note the incompatible specification of the value at the AGR|NUM path.)

One point about the use of coreference and disjunction is worth special mention: there is "normally" no way to specify that structure is shared between one disjunct of a (possibly distributed) disjunction and anything outside the disjunction. Thus the feature structure below is misleading:

(1) $$\left[\begin{array}{l}\text{ATTR1}\ \boxed{1}\\ \text{ATTR2}\ \{\ [\ \text{ATTR3}\ \boxed{1}\ ],\ [\ \text{ATTR4}\ \boxed{1}\ ]\ \}\end{array}\right]$$

We said that there is "normally" no such allowable specification, but this should be clarified: the semantics of the formula here is clear enough (it can be readily reduced to disjunctive normal form), but two things must be noted. First, the formulation is somewhat deceptive, in that the value in [ATTR1 $\boxed{1}$] in the formula above is implicitly disjunctive. This may be seen if one considers the result of unifying the above with a (nondisjunctive) description:

$$\left[\ \text{ATTR2}\ \begin{bmatrix}\text{ATTR3}\ a\\ \text{ATTR4}\ b\end{bmatrix}\ \right]$$

which yields:

$$\left[\begin{array}{l}\text{ATTR1}\ \{a, b\}\\ \text{ATTR2}\ \begin{bmatrix}\text{ATTR3}\ a\\ \text{ATTR4}\ b\end{bmatrix}\end{array}\right]$$

This is perhaps not immediately appreciated when such structures are encountered. The second point is related, namely, that the administration of coreference which spans disjunction can become fairly involved. It is worth noting that neither of these points calls the legitimacy of the coreference spanning disjunction into question, they merely point out that it is a sensitive area. For this reason, we shall not avoid it completely here. It will allow us more succinct representations.

The reason why a restriction against coreference spanning disjunction is not felt to hinder expressivity is clear: there is a reasonably succinct alternative to the form (1) above:

$$\left[\begin{array}{l}\text{ATTR1}\ \{_{\$1}\ \boxed{1}, \boxed{2}\ \}\\ \text{ATTR2}\ \{_{\$1}\ [\ \text{ATTR3}\ \boxed{1}\ ],\ [\ \text{ATTR4}\ \boxed{2}\ ]\ \}\end{array}\right]$$

Notice that coreferences do not span disjunctions here, since we employ coreferences only within single alternatives of a distributed disjunction.

The significance of feature structures for the present work is twofold: first, contemporary grammatical work in computational lingusitics is nearly universally conducted in feature-based grammars, so that it is important that lexical specifications for feature-based theories be clearly interpretable. The present work also aims to provide a lexicon for feature-based grammar, where the lexicon may be viewed as a (disjunctive) collection of feature descriptions. Feature structures will be the only content of the lexical specifications in the lexicon; in other words, specifications associate word class types with feature structure descriptions.

This brings us to the second point of connection between feature-based theories and lexicon theory. The work on "structured lexicons" cited in the introduction by Flickinger et al [19], Flickinger [17] and Evans and Gazdar [16] emphasized the value of lexicons in which specifications were as free as possible of redundancy, and these works eliminated redundancy by exploiting a

relation of INHERITANCE between lexical classes, realized as a relation between nodes in a directed graph (inheritance hierarchy). In structured lexicons, the word class TRANSITIVE VERB inherits properties from the word class VERB as well as the word class TRANSITIVE. The usual formulation in feature-based theories takes the inverse relation 'bequeath' as primitive. This relation is very naturally characterized in feature-based theories as the relation 'less-informative-than', or SUBSUMPTION, which we symbolize '$\sqsupseteq$'. For example, the feature structures below stand in this relation:

$$[\ \text{AGR}\ [\ \text{NUM SG}\ ]\ ] \sqsupseteq \left[\ \text{AGR}\ \left[\begin{array}{l}\text{PER THIRD}\\ \text{NUM SG}\end{array}\right]\ \right]$$

We shall therefore formulate statements about inheritance (the inverse of bequeath) using '$\sqsubseteq$'. We implement this notion of inheritance using a procedure, 'unify' (or 'default-unify'), which unifies the information of every superclass with LOCAL idiosyncratic information to determine a fully expanded prototype of the class in question. Feature description languages thus provide a natural formalization for work in structured lexicons.

A final aspect of modern feature theories that we shall have cause to exploit is their use of TYPING (cf. Carpenter, [11] for a presentation). A type system imposed on a system of feature structures has several tasks: first, provides a means of referring to CLASSES of feature structures of a given restricted sort. We shall put this to good use, e.g., in representing derivational relationships as complex inheritance. Second, attributes are restricted by type to being appropriate on a limited class of feature structures; thus the attribute VFORM will be limited in appropriateness to objects of type *verb* or *verbal-head.* Third and finally, the values of attributes will be restricted by type.

2.2 HPSG

Although most of what we propose might be realized in formalisms weaker than HPSG, it is worth noting that we shall employ RECURSIVE TYPE SPECIFICATIONS of a kind found in HPSG, but generally not elsewhere. In HPSG the type *sign* has an attribute SYNTAX|LOCAL|SUBCAT which is restricted in value to lists of signs. This attribute encodes SUBCATEGORIZATION information, which is lexically based in HPSG, much as it is in Categorial Grammar (Bach, [4]). Grammatical heads specify the syntactic and semantic restrictions they impose on their complements and adjuncts. For example, verbs and verb phrases bear a feature SUBCAT whose content is a (perhaps ordered) set of feature structures representing their unsatisfied subcategorization requirements. Thus the feature structures associated with transitive verbs include the information:

$$\left[\begin{array}{l}\textit{trans-verb}\\ \text{SYN|LOC|SUBCAT}\ \langle\left[\begin{array}{l}\textit{NP}\\ \text{CASE ACC}\end{array}\right], \left[\begin{array}{l}\textit{NP}\\ \text{CASE NOM}\end{array}\right]\rangle\end{array}\right]$$

(where *NP* is the type of noun phrase signs, and *trans-verb* the type of transitive verb sign).

The significance of subcategorization information is that the subcategorizer may combine with elements listed in its SUBCAT feature (perhaps only in a particular order) in order to form larger phrases. When a subcategorizer combines with a subcategorized-for element, the resultant phrase no longer bears the subcategorization specification—it has been discharged (cf. Pollard and Sag, 1987, p.71 for a formulation of the HPSG SUBCATEGORIZATION PRINCIPLE). We shall have cause to return to subcategorization in our presentation of derivation.

In order to appreciate the point about recursive specification, let us regard the subcategorization list as represented in [FIRST, REST] form (so that every SUBCAT either is null or occurs in [FIRST, REST] form). Then, the important point is to note that we have a type *list*, one of whose attributes, REST is restricted to values of type *list*, including the empty list. This is a recursive type specification. In general, SUBCAT is restricted to taking values which are of the type *list(sign)*—and this attribute occurs within signs. A similar recursion obtains when we define the type *tree* as a *lexical-sign* or a *sign* whose attribute DAUGHTERS is a list of signs of the type *tree.* We shall employ recursive type specifications in a proposal for the representation of derivational relationships.

3 Inflection

We turn our intention here to the treatment of inflectional paradigms as exemplified by verbal, adjectival and nominal paradigms in German. We illustrate with a fairly traditional representation of the inflectional endings used in the present active paradigm of weak verbs below (and we explore alternatives to this below):

	sg	pl
1st	+ e, *kriege*	+ en, *kriegen*
2nd	+ st, *kriegst*	+ t, *kriegt*
3rd	+ t, *kriegt*	+ en, *kriegen*

3.1 Interface to Morphophonemics

It is worth noting here that the forms found in paradigms may be more complex than simple concatentations of stems and the inflectional endings specified in paradigms. Full forms cannot always be derived from simple concatenation since internal sandhi rules, i.e., rules of morphological juncture of various sorts, may need to be applied. In fact, this is the case in the simple example here. For weak verb stems ending in alveolar stops, a rule of schwa-epenthesis must be invoked in the third-singular position (and this is NOT a general phonological process in German).

	sg	pl
1st	arbeit + e	arbeit + en
2nd	arbeit + st	arbeit + t
3rd	arbeit + t, *arbeitet*	arbeit + en

The status of *umlaut* in strong conjugations (*ich schlage, du schlägst*) is similarly sensitive to morphological juncture—thus it is triggered by the second singular present /st/, but not by the second singular preterite /st/.

We draw attention to these phenomena only to emphasize that, while we are aware of such complications, we do not intend to treat them here because a proper treatment involves morphophonemic detail which is not our primary focus here.[9] Our focus here will be on the morphological interface to syntax, rather than on the interface to morphophonology.

The interface to allomorphy is then quite simple:

$$\left[\text{MORPH}\left[\begin{array}{l}\text{STEM } \boxed{2}\\ \text{ENDING } \boxed{2}\\ \text{FORM } \boxed{2}\,\&\,\boxed{3}\end{array}\right]\right]$$

where '&' designates the concatentation operation between morphs. It is the task of the allomorphy to "spell out" the combination under FORM. We also emphasize that we indulge below in the convenient fiction that the inputs to allomorphy can be adequately specified using strings—even

[9] For the theorist who would like to maintain the interesting claim that feature structures can represent ALL linguistic knowledge, the feature-based treatment of morphophonemics is of great potential interest. Cf. Bird [6]. For efficiency reasons, however, a second path might be chosen, viz., the employment of a hybrid feature-based two-level morphology. This has efficiency advantages, and it is our intention to pursue this line. Cf. Trost [48]. A third possibility would be to simply anticipate the effects of morphophonemics in the specification of paradigms (even if this results in the multiplication of paradigms).

though we are convinced that a more abstract characterization (e.g., in feature structures) is necessary.

We shall not provide a detailed proposal—either here or in the following section on derivation—about the interface of the lexeme-based lexicon to morphophonemics. The main issue we see as significant here is the relationship between the lexeme-based lexicon and morpheme-based generalizations which might arise in morphophonemics. This is a subject of current investigation.

3.2 Approaches to Inflection

The dominant approaches to the treatment of inflection in computational linguistics have been either (i) to model inflection using collections of lexical rules, or (ii) to employ two-level morphology. The deployment of lexical rules may be found in Flickinger's approach (cf. Flickinger, [17], pp.107-110), in the Alvey tools project (cf. Ritchie et al., [38], p.298), and in HPSG (cf. Pollard and Sag, [36], pp.209-213). Paradigmatic morphology improves upon these ideas by defining the paradigm as a sequence of lexical rules on which subsumption relations can be defined (cf. Calder, [9]), but the fundamental analytical tool is still the lexical rule.

In this view, the inflectional paradigm above is described by postulating rules which relate one paradigm element to another (or relating it to another form), including perhaps a rule to derive first singular forms from infinitives. While nothing would prohibit a lexical rule from operating on abstract stems to create forms, this was seldom done (cf. Karttunen's LFG-style treatment of passive in D-PATR, Karttunen, [26], pp.12-14 for an exception). In the following section on derivation (Section 4), we sketch a feature-based theory of lexical rules within which these notions of the paradigm could be recast, but we prefer to demonstrate the flexibilty of the feature-based approach here by developing a more purely paradigmatic view. While it would clearly be possible to formulate a rule-based view of the paradigm in feature structures, the analytically more challenging task is to describe directly the abstract variations which constitute paradigms—i.e. to try to characterize paradigms directly without recourse to lexical rules.

The two-level approach to inflection may be found in Koskenniemi [32] (for an interesting extension in the direction of feature-based processing, cf. Trost [48]). This differs from the current proposal in being morpheme-based. It is, however, compatible with various lexical structures, as is demonstrated by its use in the Alvey project, noted above [38].

The direct characterization of the paradigm has been the alternative approach both in linguistics (cf. Matthews, [34], Chap.IV) and in computational linguistics (cf. Evans and Gazdar's DATR, [16], and Russell et al.'s ELU lexicon, [40]). The fundamental idea in our characterization is due to the work in DATR, in which paradigms are treated as alternative further specifications of abstract lexemes. We express the same fundamental idea in feature structures by defining paradigms as large disjunctions which subsume appropriate lexical nodes. Each disjunct represents conceptually one element of the paradigm, and specifies what is peculiar to that element. The fundamental person-number verb paradigm in German is just an association of forms with the six disjuncts in the disjunctive normal form of the following disjunction:

$$\left[\ \text{AGR}\ \left[\begin{array}{l}\text{PER}\ \{\text{1ST}, \text{2ND}, \text{3RD}\}\\ \text{NUM}\ \{\text{SG}, \text{PL}\}\end{array}\right]\right]$$

3.3 A "Word and Paradigm" Approach

We may now employ distributed disjunctions (cf. Section 2) to link the fundamental alternation in (1) to the expression of forms. We obtain the following description of the present weak paradigm:

$$\left[\begin{array}{l}\text{MORPH}\ \left[\begin{array}{l}\text{STEM}\ \boxed{2}\\ \text{ENDING}\ \boxed{3}\{_{\$1}\ \text{“e”}, \text{“st”}, \text{“t”}, \text{“n”}, \text{“t”}, \text{“n”}\}\\ \text{FORM}\ \boxed{2}\&\boxed{3}\end{array}\right]\\[2ex] \text{SYN|LOCAL|HEAD|AGR}\ \left\{_{\$1}\ \left[\begin{array}{l}\text{PER 1ST}\\ \text{NUM SG}\end{array}\right], \left[\begin{array}{l}\text{PER 2ND}\\ \text{NUM SG}\end{array}\right], \cdots, \left[\begin{array}{l}\text{PER 3RD}\\ \text{NUM PL}\end{array}\right]\right\}\end{array}\right]$$

Each of the disjunctions tagged by '$1' constitutes effectively the same set of alternatives, taken in sequence. Thus we understand the feature structure above as denoting a disjunction of six disjuncts, the first of which has "e" as a value for MORPH|ENDING AND [PER 1ST, NUM SG] (1st-sing) as a value of SYN|LOCAL|HEAD|AGR; etc. This node may be inherited by all German weak verbs.

In general, in representing a paradigm, the EXPONENTS of the paradigm are listed as alternatives under [MORPH|ENDING {$n...}], while their associated PROPERTIES appear elsewhere within the feature structure under the same distributed disjunction '{$n...}' (for this terminology, cf Matthews, [35]).

One advantage of representing paradigms in this fashion—as opposed to representations via lexical rules, as in Flickinger [17] or in the other rule-based approaches cited above—is that the paradigm is represented in the same formalism in which word classes and lexemes are represented. A paradigm may thus participate in the same inheritance relationships that relate word classes and individual lexemes. We may, e.g., represent modal paradigms as inheriting by default from standard paradigms, modifying only the "morph" value:

$$[\ \text{MORPH|ENDING}\ \{_{\$1}\ \text{“”}, \text{“st”}, \text{“”}, \text{“n”}, \text{“t”}, \text{“n”}\ \}\ ]$$

(Note that we assume that the co-naming of disjunctions is inherited, so that the endings specified above are still to be understood as covarying with syntactic agreement features. The specification may be this compact because it expoits the nonlocal information in naming disjunctions.) The modal verb *sollen* is an example of this subparadigm:

	sg	pl
1st	soll +	soll + n
2nd	soll + st	soll + t
3rd	soll +	soll + n

The example is misleading in that we have shown the only modal paradigm with just this distinction from standard paradigms. We are aware that modal paradigms are in general also characterized by stem vowel alternations *kann, können*, etc., and we should have to represent this information as well. We could, e.g., allow lexemes to have a present singular stem and a present plural stem which are normally identical, but which are distinguished for these modals (and a very few other verbs such as *wissen*), or we might even try to describe the vowel alternations directly, if this were not going further into the morphophonemics than we care to (at least in this paper).

Genuine SUPPLETION we likewise propose to treat via default overwrting:

$$\left[\begin{array}{l} \text{MORPH} \left[\begin{array}{l} \text{STEM} \\ \text{ENDING} \\ \text{FORM}\ \{_{\$1}\ \text{“bin”}, \text{“bist”}, \text{“ist”}, \text{“sind”}, \text{“seid”}, \text{“sind”}\} \end{array}\right] \\ \\ \text{SYN|LOCAL|HEAD|AGR} \left\{_{\$1} \left[\begin{array}{l} \text{PER 1ST} \\ \text{NUM SG} \end{array}\right], \left[\begin{array}{l} \text{PER 2ND} \\ \text{NUM SG} \end{array}\right], \ldots, \left[\begin{array}{l} \text{PER 3RD} \\ \text{NUM PL} \end{array}\right]\right\} \end{array}\right]$$

Here we exploit default overwriting to describe the suppletive forms of *sein*. Note that we avoid the sticky question here of what the STEM and ENDING of such suppletive word forms ought to be. Perhaps a more graceful means of avoiding this question would be to deny that the forms are concatenations of STEM and ENDING (which we have not done above—we merely fail to identify them).

A DEFECTIVE PARADIGM, e.g., that of the verbs *dünken* 'to think', or *jemandem an etwas liegen* '(for something) to be important to someone', which occur only in the 3rd person singular, may be analyzed as a more extreme instance of overwriting inheritance—the entire disjunction is

eliminated. (It is worth noting that non-overwriting analyses may also be formulated here, just as in most other places.)

$$\left[\text{SYN|LOCAL|HEAD|AGR } \{_{\$1} \perp, \perp, \left[\begin{array}{l} \text{PER 3RD} \\ \text{NUM SG} \end{array} \right], \perp, \perp, \perp, \} \right]$$

Of course, one can formulate alternative linguistic descriptions of the requirement that this verb appear only in the third-person singular. For example, rather than say that the other forms do not exist, which is roughly the content of the specification above, one could postulate that impersonal verbs subcategorize for an (abstract) subject which is 3rd-sg. Our goal here is not to defend these particular "paradigmatic gap" analyses, but rather to show that the phenomenon may be satisfactorily formulated in feature structures.[10] A futher alternative would be to overwrite the AGREEMENT value nondisjunctively. There are probably alternative descriptions available.

These examples of paradigm inheritance point out limitations to the technique worth noting, even if they eventually prove to be convivial. First, although we can allow the inheritance of distributed disjunctions to model the inheritance of paradigms, there is no way to make sense of an inherited paradigm being larger or smaller than the ancestor from which it inherits—so that we cannot sensibly construe a four-element paradigm as inheriting from a six-element one, or *vice versa*. This limitation arises because a distributed disjunction is always an alternation of elements in order. Second, there is no way to note that a single form in a paradigm is exceptional without respecifying the entire paradigm—the disjunction must be respecified as a whole. This stems from the fact that there is no way to identify a particular alternation within the distributed disjunction. Thus the defective paradigm above had to respecify the entire paradigmatic disjunction (this suggests perhaps using features to identify forms, which is the technique employed in DATR).

This in general is the case, and it brings us to a slight modification of the style of representation we shall employ. We examine this in the following section.

3.4 Matrix- vs. Form-based Approaches

The specifications above are written in what might be called the "matrix" style of the word-and-paradigm model—every cell in the matrix of cross-cutting paradigmatic distinctions is specifically assigned a form value. The incorporation of a further dimension with n distinctions would increase paradigm size by a factor of n—even if NO new forms were introduced. This is the presentation preferred in didactic grammars. Thus we might view a determiner or nominal paradigm in the fashion shown in Figure 1.

Opposed to this "matrix-based" view of the paradigm is a "form-based" view—only distinct forms are assigned feature specifications. The same (weak present) paradigm can be represented as an alternation of only four forms, two of which are (disjunctively) underspecified for agreement values (note that we employ nested distributed disjunctions in order reduce the top-level 6-set to a 4-set). In general, this is the presentation used by linguists:

$$\left[\begin{array}{l} \text{MORPH} \left[\begin{array}{l} \text{STEM} \\ \text{ENDING } \{_{\$1} \text{ “e”, “st”, “t”, “n”}\} \end{array} \right] \\ \\ \text{SYN|LOCAL|HEAD|AGR} \left[\begin{array}{l} \text{PER } \{_{\$1} \text{ 1ST, 2ND, } \{_{\$2} \text{ 3RD, 2ND}\}, \{\text{1ST, 3RD}\}\} \\ \text{NUM } \{_{\$1} \text{ SG, SG, } \{_{\$2} \text{ SG, PL}\}, \text{PL}\} \end{array} \right] \end{array} \right]$$

Which representation is preferable? A feature-based lexicon need not distinguish between these two representations; indeed, they are provably equivalent within the formalism, even if they illustrate two distinct styles in representing paradigms.

[10] Better candidates for genuine paradigmatic gaps in German are e.g., *Eltern* 'parents', which lacks a singular (except in the speech of biologists), or the verb *verscholl, verschollen* 'to get lost', which lacks all finite present forms. The English auxiliaries *come* and *go* are also frequently cited to demonstrate the existence of gaps in paradigms. Thus *I come/go see her daily; *He comes/goes see her daily*, even though *He wants to come/go see her daily.*

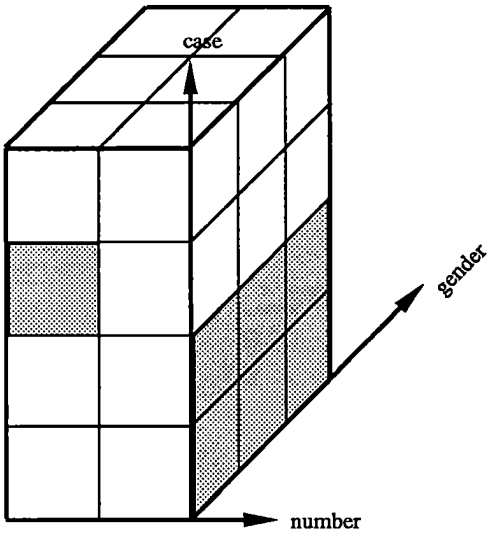


Figure 1: A three-dimensional view of the German determiner (or nondeclinable noun) paradigm, with dimensions corresponding to person, number, and case.

This assertion of equivalence concerns the properties of the feature structures only. The two representations are distinct in their default inheritance properties (which are thus in a sense intensional), so that overwriting defaults from the one or the other structure would be different. But the representations describe the same structures.[11]

3.5 Complex Paradigmatic Elements

In traditional treatments of the paradigm, the passive is normally listed as one of the elements in the paradigm. But passive and active forms differ not merely in the assignment of agreement features (of fairly simple structure) but also in the subcategorization classes to which they belong, and these are quite complex when expressed as feature structures, involving lists of feature structures. The representation of passive as an inflectional variant is something of a challenge for lexical representation schemes.

It is therefore worth noting that even paradigmatic elements as complex as passive may be described as inflectional variants using distributed disjunctions in the way just sketched. That is, we may describe not only the covariation between inflectional affix and a syntactic feature [PASSIVE ±], but even the effect of passive, e.g., on an HPSG subcategorization feature. We hasten to add that feature-based lexicons are not forced to this point of view—one could likewise formulate a derivational rule of passive (cf. Section 4), nor do we wish to advocate the inflectional analysis at this time.[12] But there are clearer examples of languages with passives which are paradigm elements, e.g. Latin (cf. Allen and Greenough [1]), and our primary goal is to establish that this (traditional) analysis is formulatable in a feature-based lexicon. It seems impossible to formulate in other structured lexicons.

[11] While it is pleasant to obtain the intuitively correct result, it also means that there is no way to distinguish the structures described. Any putative distinction related to the two types of representation would have to be explained in another fashion.

[12] Cf. Kathol [28] for a third, perhaps most interesting analysis, under which a single participial form serves in both active and passive voices, so that passive is neither an inflection, nor a derivational alternation.

$$\left[\begin{array}{ll} \text{MORPH} & \left[\begin{array}{l}\text{STEM}\\ \text{PREFIX } \{_{\$1}\ \text{“”}, \text{“ge”}\}\\ \text{ENDING } \{_{\$1}\ \{_{\$2}\ \text{“e”}, \ldots, \text{“n”}\}, \text{“n”}\}\end{array}\right] \\ \text{SYN|LOCAL} & \left[\begin{array}{l}\text{SUBCAT } \{_{\$1}\ \langle \text{NP[ACC]}_{\boxed{4}}, \text{NP[NOM]}_{\boxed{3}}\rangle,\\ \qquad\qquad \langle(\text{PP[VON]}_{\boxed{3}},)\text{NP[NOM]}_{\boxed{4}}\rangle\ \}\\ \text{HEAD|AGR } \left\{_{\$2}\ \left[\begin{array}{l}\text{PER 1ST}\\ \text{NUM SG}\end{array}\right], \ldots, \left[\begin{array}{l}\text{PER 3RD}\\ \text{NUM PL}\end{array}\right]\right\}\end{array}\right] \\ \text{SEM} & \left[\begin{array}{l}\text{PRED}\\ \text{SOURCE } \boxed{3}\\ \text{THEME } \boxed{4}\end{array}\right] \end{array}\right]$$

where NP[ACC] abbreviates [*np*, CASE ACC], etc. The structure above provides for two alternate (sets of) forms, active and passive.[13] This alternation is indicated by the disjunction name '$1'. The second, passive alternative correlates with a subcategorzation in which, e.g., an optional PP[von] phrase fills the same semantic argument slot as the active subject. The first, active alternative in $1 is simply the active paradigm sketched above.[14]

The version of passive shown here is the version typically used to demonstrate grammatical analysis, i.e. passive is taken to be a variant of transtive verbs only, and no connection between passive and perfect participle is noted, even though these are never distinguished in form. Both of these flaws may be remedied, by what amount to essentially disjunctive specifications. For example, we may allow that the second alternative in $1 above correspond either to the passive syntactic structure (given) or to a perfect participle. It is worth clarifying that we do not claim that anything particularly insightful is gained by a disjunctive analysis of this sort; for insight we need some more detailed work in linguistic analysis. It is nonetheless worth noting the possiblity of the further (disjunctive) detail, given the program of specifying all inflectional variants of a given lexeme as alternative further specifications of the lexeme. These further disjunctive and perhaps uninsightful specifications do not stand in the way of realizing this program.

3.6 Alternative Frameworks

We discussed above why we prefer NOT to analyze inflectional variation as an employment of lexical rules (but cf. Section 4 for a proposal for representing in feature structures relations which linguists normally designate as 'lexical rules'). There are two similar proposals in inheritance-based, computational lexicons for analysing inflectional variation as the further specification of an abstract lexeme. We discuss these in the present section.

DATR (cf. Evans and Gazdar [16]) is a graph description language (and inference engine) which can encode information about lexical items. DATR provides a formally clean version of the sort of default inheritance first advocated in de Smedt [45] and Flickinger et al. [19], and as such represents a significant advance in the understanding of default lexical inheritance. Although DATR superficially resembles feature notation, its semantics are in some ways different, so that an interface between DATR and a feature system is required, if DATR is to be used as a lexicon in combination with a feature-based parser (or morphological processor)—in keeping with the intention of DATR's developers. The fundamental advantage of our approach over that of DATR

[13] But it is clear that a description such as the one above cannot hold of the Latin DEPONENT forms—those with passive inflection, "active" meaning, and no active counterpart (*loquor*).

[14] It would be useful to find a way to allow the active paradigm not merely to be reused, but actually to be inherited in cases such as this (Latin). We have not seen a way of specifying this, since it amounts to specifying that a value be inherited as one of several disjuncts "to be further specified".

is that no such interface is required: lexical structures are exactly the structures required in syntax (and morphology).[15]

A further advantage accrues to feature-based approaches, because they come with a logic which has proven useful in linguistic description. Cf., for example, the extensive employment of disjunction above. Disjunction could probably be added to a system such as DATR, but it does not seem that anyone is trying to do this. A more subtle point is the status of coreference, a notion which feature logics are designed to treat in a very particular way, but which seems inexpressible in DATR.[16]

A final point of divergence is the degree of complexity which an inflectional specification may be allowed to have. The DATR scheme is to use specifications of the form:

$$< \text{path} >==< \text{value} >$$

to encode dependencies between properties and exponents, e.g.

$$< \text{present first plural} >== \text{"en"}$$

and this works fine as long as the properties involved fit neatly along a simple path. If properties become complex, on the other hand, as they seem to in the case of the (at least the Latin) passive, then this scheme breaks down. The feature-based model using ditributed disjunctions is freer: dependent properties may be distributed in various positions in a feature structure.

The ELU lexicon (Russell et al., [40]) uses equations describing feature structures for the most part, but the equations are divided into a "main set" and a "variant set", the latter of which (monotonically) describe inflectional variants. The paradigm is then described using a set of implicational equations, expressing information such as "if x = [pers: 3, num: sg], then y ='s' ".[17] The use of distributed disjunctions seems to be more concise method of specifying inflectional alternatives, but the underlying logic of the ELU approach is very similar to ours.[18]

4 Derivation

Looking at work done in the area of word formation, various formalisms and theories can be ordered according to different dimensions. One possible dimension of classification is, for instance, the distinction between procedural vs. declarative formulation of rules relating phonological, morphological and orthographic phenomena (cf. Calder, [9]).[19] Classifying specific treatments is subjective. Most linguists will interpret lexical rules procedurally. But lexical rules can also be regarded purely declaratively, even if the procedural view is the most prominent one.[20]

[15] Cf. Kilbury et al. [31] for a rather bleak view of prospects for interfaces between DATR and feature formalisms.

[16] What makes this point subtle is that DATR does allow the expression of "path equivalences", e.g., a statement such as < agr >==< subject agr >. Thus, if < subject agr person >= first AND < subject agr number >= sg then the same values will accrue to agr. But the relation is not symmetric; assignment of values to agr either override or are overridden by the above. (DATR's '==' is like the assignment operator in imperative programming languages, not like the identity relation.) A further refinement is that, even in the direction in which this does work (like identity) no distinction is made between two paths which have the same values and those which have distinct, but equivalent values. But such "structure-sharing" is very widely exploited in feature-based linguistic analyses.

[17] Information in variant sets is not subject to default overwriting, for reasons which are not explained. Although we find the proposal that some information not be subject to default overwriting congenial, it would seem desirable to view the specification of alternations as an orthogonal point.

[18] We have also benefited from the opportunity to examine unpublished work of Andreas Kathol at The Ohio State University, who has independently developed a similar treatment of inflectional variation—this one based on constrained relations.

[19] Our briefly sketched treatment of derivation (and inflection) in the introductory section is in its essence declarative, because linguistic knowledge is encoded in terms of feature structures only and unification is the sole information-building operation.

[20] Interestingly to note, Pollard & Sag, [36], pp. 209, suggest a third interpretation of lexical rules coming directly from the field of many-sorted abstract data types—an ALGEBRAIC perspective on lexical rules.

4.1 External vs. Internal Lexical Rules

Instead of treating the issue of declarative vs. procedural formulations, we want to turn our attention to another dimension of classification: *Where does word formation take place—within or without the lexicon?* WITHOUT means that the form of lexical rules is different from the structure of lexical entries (lexemes, also possibly morphemes). Lexical rules in PATR-II [44] or D-PATR [26], e.g., look like feature structures and are represented via a collection of path equations, but their interpretation is completely different from that of (normal) feature structures. The same is true, if we move to other theories: *f-structures* differ in form (syntax) and interpretation (semantics) from lexical rules stated in LFG (cf. the articles in Bresnan, [8]). This same observation holds for HPSG [36], Ch. 8.2,[21]for the Alvey tools project [38], for the early days of HPSG [19], for the work of Flickinger [17], and also for Hoeksema's Categorial Morphology [23].

By its nature, an EXTERNAL lexical rule sets up a relation between two lexemes (or classes of lexemes)—or, in the case of feature-based theories, between two feature structure descriptions. But specifying the exact meaning of this mapping is an open question—nearly all theories have different viewpoints when interpreting (external) lexical rules:

- Are external lexical rules functions or perhaps even relations?
- If functions, do they take one argument or arbitrarily many? (*Mutatis mutandis* for relations)
- Are they unidirectional or bidirectional?
- If unidirectional, will they be interpreted declaratively (AVM_1 implies a corresponding AVM_2) or procedurally (lexical rule as an instruction, to build AVM_2 out of AVM_1)?

Instead of treating external lexical rules further, we'd like to propose a novel interpretation of lexical rules, which we call INTERNAL. An internal lexical rule is an information-bearing object, indistinguishable in its form and meaning from other entries of the lexicon—strictly speaking, it just IS a lexical entry.

Derivational rules will be modeled as feature structure descriptions, just as lexical entries are. This is aesthetically pleasing, and has the further advantage of formal clarity. Perhaps most importantly, however, the fact that lexical rules and lexical entries are of the same formal type allows one to liberate yet another level of linguistic structure from PROCEDURAL considerations and therefore allows one to interleave, e.g., morphological and phrasal processing in a way that is otherwise prohibited.[22]

It is worth noting that there is no one-to-one correspondence between traditional lexical rules (what linguists have called lexical rules) and single objects (feature structures) in our approach. Rather, the information in a lexical rule is distributed among lexical entries, principles, and morphological dominance schemata—our closest analogue to "rule".

In general, internal (or external) lexical rules cannot be realized as independent lexemes (or morphemes); instead rules serve as FILTERS (in the sense of well-formedness conditions), used to rule out ill-formed structures (to fail to parse or generate them). A treatment of this kind will be

[21] Feature structures descriptions and lexical rules (form: $AVM_1 \longmapsto AVM_2$) in HPSG have nothing in common, because they differ in form as well as in interpretation. This remark is supported by the following observation: feature structures in HPSG are always typed, and these types can be ordered (partially) via subsumption. But this isn't true for lexical rules. A lexical rule as a whole does not have a type and there's no way to relate it to other feature structures. Under the assumption that a lexical rule can in principle be typed and resides in the lexicon, this type ought to be a subtype of *lexical-sign* according to Pollard & Sag [36] and ought to have exactly the three top-level attributes PHON, SYN, and SEM—but this isn't the case.

[22] One can quibble about our choice of terminology here. Given the possibility of interleaving processing in the way we describe, it might seem as if what we are calling INTERNAL lexical rules are rather more external than other construals would have it. Or one can insist on the standard construal of derivation as a mapping from lexical entries to lexical entries, which implies that the term INTERNAL lexical rule is misleading, since this insistence effectively equates the notion 'lexical rule' with that of an external mapping, always taking a set of feature structures and yielding a feature structure. Our own preference for the term 'internal lexical rule' arises because we see the lexicon in general as constituted by a set of feature structure descriptions—with no fundamental distinction between lexical rules and lexical entries. Lexical rules are simply a particular kind of feature structure description.

presented here for the field of DERIVATION. Another approach, which is also distinguished by its use of internal lexical rules, can be found in the ELU system [40].[23]

4.2 Our Treatment of Derivation

In HPSG-I linguistic knowledge about word formation is encoded through a family of lexical rules, which are not feature structures, but rather essentially external operators working on feature structures. This (for us) unsatisfactory view appears even more questionable given the view of most linguists that form and meaning are much harder to describe for sentences and phrases than for words. If this is the case, one may ask, why does HPSG treat word formation via external lexical rules rather than in a purely feature-based way? Why not formulate RULES and PRINCIPLES for word grammar similar to those stated by Pollard & Sag [36] for phrasal and sentential grammar? We think, there are at least three replies this question might provoke:

1. HPSG is a theory capable only of capturing the form and meaning of sentences in feature structure descriptions, but incapable of describing morphotactical aspects of language.

2. Trying to handle lexical structures (morphotactics) in terms of feature structure descriptions (i.e., via rules and principles) only leads to inefficient implementations.

3. HPSG is a conglomerate of different formalisms and theories (cf. [42], p. 1, and [36], p. 1), saying little or nothing about morphotactics. In the theories from which HPSG borrows, morphotactics were stated in form of external lexical rules. HPSG assumed this view somewhat nonreflectively, because HPSG's primary purpose is the description of phrasal and sentential syntax and semantics.

We're convinced that the first thesis is simply WRONG, and that the second one is (at the moment) probably true (because procedural implementations of lexical rules can be very efficient), while the third statement is definitely correct. Summing up, we think it's a promising task to approach DERIVATION purely in terms of feature structure descriptions—just in the spirit of HPSG. Recall the two equations in [36], p. 147,

$$\text{UG} = \text{P}_1 \sqcap \ldots \sqcap \text{P}_n \tag{2}$$

$$\text{English} = \text{P}_1 \sqcap \ldots \sqcap \text{P}_{n+m} \sqcap (\text{L}_1 \sqcup \ldots \sqcup \text{L}_p \sqcup \text{R}_1 \sqcup \ldots \sqcup \text{R}_q) \tag{3}$$

These may be understood in the following way: universal grammar (UG) consists of a set of principles $\text{P}_1, \ldots, \text{P}_n$, whose conjunction (or unification) must hold true of every structure in every language. In addition, a given language may impose the language-specific constraints $\text{P}_{n+1}, \ldots, \text{P}_{n+m}$. Finally, the grammar requires that every structure instantiate some lexical entry $\text{L}_1, \ldots, \text{L}_p$, or phrasal pattern (rule) $\text{R}_1, \ldots, \text{R}_q$—since a structure need satisfy only one of the lexical or rule descriptions in order to be a well-formed phrase, these constraints obtain disjunctively. A language is then just the set of structures which simultaneously conform not only to all the principles, both universal and language-specific, but also to at least one of the lexical or phrasal descriptions. These fundamental equations define an HPSG grammatical theory for phrases and sentences, and we propose to apply a similar methodology to derivation, relying extensively on RULES, PRINCIPLES, and unification-based INHERITANCE (for an explanation of (2) and (3), cf. Pollard & Sag, [36], p. 147).

In contrast to inflection, derivation cannot rely on NAIVE inheritance alone. Here, 'naive' means that a word like the German *weglaufen* is defined by inheriting (unifying) all the properties from the prefix *weg-*, the verb *laufen*, plus additional idiosyncratic properties of the new complex lexeme, i.e.,

$$\text{weglaufen} = [\textit{weg}] \sqcap [\textit{laufen}] \sqcap [\ldots\ldots]. \tag{4}$$

[23] ELU treats inflection as well as derivation by means of pure inheritance. We are convinced, however, that this approach is not strong enough for derivation (cf. below).

ELU's treatment of derivation (cf. [40], p. 218) is done in such a way, and it may be a reasonable tack to take for the treatment at hand, that of the German separable prefixes. Applied generally, an approach like this leads to several insurmountable problems:

- If we relied on naive inheritance as the (sole) descriptive means, it would seem impossible to explain how the iteration of derivational processes could ever lead to different results. If *anti-* (or take the German *vor-*) is a derivational prefix, and its effect on a stem is described via inheritance, then the effect of inheriting it should be the same, whether there are one, two, or more instances of the SAME prefix in a word because unification is IDEMPOTENT and inheritance defines itself through unification. Thus a complex word like *anti-missile* (or *Vor+version*) would be predicted to be the same as *anti-anti-missile* (or *Vor+vor+version*).[24] Likewise, such an approach is not capable of explaining the INDIRECT recursion occurring in complex compounds such as *institu+tion+al+isa+tion.*

- Sole reliance on naive inheritance leaves little opportunity to explain the hierarchical structure often found in morphology, e.g., the difference in bracketing one finds in complex words containing at least two affixes, e.g., *[un- [do -able]]* as opposed to *[[un- do] -able].* Because inheritance is associative and monotonic (in the absence of overwriting), other mechanisms must be at play. Naive inheritance seems incapable of accounting for any structure, let alone ambiguous hierarchical structure.

- Simple examination of derivational results suggests that treating all of them via naive inheritance from a single lexeme will lead to unwieldy lexicons: a form such as German *Ableit+bar+keit* (derivability) would seem to require that verbal, adjectival, and nominal paradigms be found as heirs of the single lexeme (recall that we dealt with this above by modeling it via mapping from lexeme to lexeme).

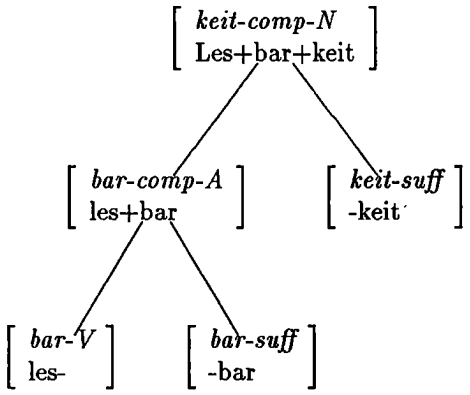
- It turns out also that there are technical problems connected with the treatment of derivation as inheritance. These may be summarized, albeit cryptically, in the following way: we should prefer that the result of a category-changing derivational process, e.g., the process which derives *derive+able* from *derive* and *-able*, be a full-fledged member of the target category (of the derivational process)—in this case, the class of adjectives. Now, if the derivational process is modeled by naive inheritance only, then *derive+able* ought to inherit from the class of verbs (through *derive*), as well. It is easy to continue this line of reasoning further (consider *derive+abil+ity*) to see how this sort of explanation leads one to the postulation of lexemes of dubious lineage, inheriting from too many ancestors (this point is essentially just the converse of the last).

Treating DERIVATION in our approach will lead to complex morphs (e.g., words) consisting of a head daughter HEAD-MORPH and a complement daughter COMP-MORPH (e.g. (5)). The task of the 'morphological-daughters' feature is to encode morphological structure, similarly to how HEAD-DTR and COMP-DTRS do this on the phrasal level (cf. [36]). This is in analogy to the HPSG formulation of phrase structure in features, yielding tree structures.[25]

[24] Permitting iteration of derivational prefixes only to a certain depth (which seems *prima facie* plausible since, e.g., words such as German *Vor+vor+vor+version* are questionable), will solve this problem, if every element of the finite set of complex prefixes is coded as a lexical entry. But this attempt at repair is (i) extremely unsatisfying theoretically and (ii) incomplete, because the depth of composition is a subjective measure.

[25] BINARY trees (together possibly with unary ones for $\emptyset$ derivation) seem to suffice, at least for derivation. Of course, this is an assumption that will be put to the test in applications of this lexical work.

(5)



We include this as an example of the hierarchical structure whose analysis is beyond the descriptive reaches of NAIVE inheritance. The objections to the description of derivation in terms of naive inheritance do not apply here, since, e.g., tree adjunction is not idempotent—so that, e.g., *Vorversion* may be distinguished from *Vorvorversion*; tree adjunction generates hierarchical structures (evident here), and, as we shall see, it distinguishes inheritance (sharing properties) from the requirements that sublexemes come from particular word classes or types (so that the tree structure above cannot be interpreted to mean that the noun *Lesbarkeit* is in any sense a verb of the same type as *lesen* or an adjective of the same type as *lesbar*). The hierarchy here is a PART-OF hierarchy in contrast to the inheritance hierarchy, which constitutes an IS-A hierarchy. The distinction is crucial: the parts of a complex word do not bequeath their properties to the words derived from them.

In general, the head daughter is a bound morpheme of type *affix*, while the complement daughter is of type *word* and is free (cf. Fig. 2). HEAD-MORPH and COMP-MORPH are put together under the label MORPHS (cf. DTRS). With these assertions in mind, we postulate, in analogy to the phrasal 'rules' in HPSG (cf. [36], pp. 149), the following morphological rule schema:

(6) $$[\text{LEX} +] \longrightarrow \text{H, C}[\text{LEX} +]$$

or more formally as a typed feature structure ('$\equiv$' is used for definitional expressions):

(7) $$\text{MHCR} \equiv \left[\begin{array}{ll} \textit{complex} & \\ \text{SYN|LOC|LEX} & + \\ \text{MORPHS} & \left[\begin{array}{l} \textit{morph-head-struct} \\ \text{HEAD-MORPH } [\textit{affix}] \\ \text{COMP-MORPH } [\textit{part-of-speech}] \end{array}\right] \end{array}\right]$$

Just as Pollard & Sag proclaim universal as well as language-specific principles, we will define four 'principles', which are consistent with our linguistic data and specified as typed implications. We don't suppose that these principles will not be overturned by wider ranges of analyses, but we do suppose that they illustrate how the HPSG style of analysis can be extended to word-internal structure. The formulation of the principles presupposes that the underlying feature logic (along the lines of Kasper & Rounds [27], [39]; cf. the section on feature structures and HPSG) is extended by adding FUNCTIONAL DEPENDENCIES (functionally dependent values; for a motivation, cf. Pollard & Sag [36], pp. 48-49; for a formal definition, cf. Reape [37], pp. 73ff).

All morphological HEAD features, as well as (morphological) SUBCATEGORIZATION will be defined, for simplicity, under the path SYN|LOC.[26] PHON will be replaced by MORPH|FORM

[26] There are good reasons to introduce, at least for affixes, an (additional) morphological subcategorization feature under the path MORPH, but we shall not pursue this here. Likewise, morphological head features may be specified under MORPH|HEAD which will seem preferable to those to whom it seems unnatural to specify the category of, e.g., *-bar* as (unsaturated) adjective. This approach is investigated in Krieger [33].

and *headed-structure* is replaced by *morph-head-struct* (morphologically headed structure) which has at least the attributes HEAD-MORPH and COMP-MORPH. The symbol *complex* (complex word) corresponds to the TYPE of the same name in the subsumption lattice (cf. Fig. 2). The CONSTITUENT ORDER PRINCIPLE ([36], pp. 169) was taken over directly from HPSG (MCOP).[27]

(8)
$$\text{MCOP} \equiv \left\{ \begin{array}{l} \left[\begin{array}{l} \textit{complex} \\ \text{MORPHS } [\textit{morph-head-struct}] \end{array} \right] \Longrightarrow \\ \left[\begin{array}{l} \textit{complex} \\ \text{MORPH|FORM order-constituents}(\boxed{1}) \\ \text{MORPHS } \boxed{1} \end{array} \right] \end{array} \right.$$

Likewise for derivation, the formulation of the HEAD FEATURE PRINCIPLE in HPSG ([36], pp. 58) is used directly (MHFP), and only certain attributes and type names were altered. Among other things, MHFP is responsible for deducing the category of the new word from the category of the head daughter.

(9)
$$\text{MHFP} \equiv \left\{ \begin{array}{l} \left[\begin{array}{l} \textit{complex} \\ \text{MORPHS } [\textit{morph-head-struct}] \end{array} \right] \Longrightarrow \\ \left[\begin{array}{l} \textit{complex} \\ \text{SYN|LOC|HEAD } \boxed{1} \\ \text{MORPHS|HEAD-MORPH|SYN|LOC|HEAD } \boxed{1} \end{array} \right] \end{array} \right.$$

The SEMANTICS PRINCIPLE may be taken in its simplest form and slightly modified ([36], pp. 99): the semantics of the mother is equal to the semantics of the head daughter.

(10)
$$\text{MSP} \equiv \left\{ \begin{array}{l} \left[\begin{array}{l} \textit{complex} \\ \text{MORPHS } [\textit{morph-head-struct}] \end{array} \right] \Longrightarrow \\ \left[\begin{array}{l} \textit{complex} \\ \text{SEM } \boxed{1} \\ \text{MORPHS|HEAD-MORPH|SEM } \boxed{1} \end{array} \right] \end{array} \right.$$

The use of binary trees and the head-complement structure for derivation leads to a SUBCATEGORIZATION PRINCIPLE which looks (and is) completely different from the one proposed in HPSG-I for phrases ([36], pp. 71). Identifying the values of MORPHS|COMP-MORPH and MORPHS|HEAD-MORPH|SYN|LOC|SUBCAT in MSCP (cf. (11)) guarantees that the head takes the right complement and binds it. In addition, the function construct-subcat assembles the subcategorization information of the new morphological phrase. The elements of the subcategorization list under path SYN|LOC|SUBCAT are essentially those of the complement—however, construct-subcat regroups them, perhaps omitting some of them (cf. examples). The result of this modification depends not only on the type of the attribute MORPHS|HEAD-MORPH, but also on the sort of the complement morpheme:

(11)
$$\text{MSCP} \equiv \left\{ \begin{array}{l} \left[\begin{array}{l} \textit{complex} \\ \text{MORPHS } [\textit{morph-head-struct}] \end{array} \right] \Longrightarrow \\ \left[\begin{array}{l} \textit{complex} \\ \text{SYN|LOC|SUBCAT construct-subcat}(\boxed{1}) \\ \text{MORPHS } \boxed{1} \left[\begin{array}{l} \text{HEAD-MORPH|SYN|LOC|SUBCAT } \boxed{2} \\ \text{COMP-MORPH } \boxed{2} \end{array} \right] \end{array} \right] \end{array} \right.$$

[27] The function order-constituents in MCOP has to be sensitive to the type of its argument. If the argument is of type *morph-head-struct* (see below), the function is being applied in derivation instead of working on the sentence level. Thus, in contrast to the version of order-constituents in [36], our version of order-constituent is overloaded with respect to its argument—we employ an AD HOC POLYMORPHISM. Cf. Cardelli & Wegner [10]. Alternatively, we could specify a second function, order-morph-constituents.

Although Pollard & Sag [36] strictly type the attributes of feature structures in general, they do not explicitly state that PRINCIPLES as well as RULES may also be regarded as types. But we may interpret them as types which have to satisfy the SUBSUMPTION relation only.[28] In taking this step, one has to integrate them consistently into the subsumption lattice (cf. Fig. 2).

With respect to equation (3), we extend the set of principles and the set of rules by adding MCOP, MHFP, MSP, MSCP, and MHCR. Finally, in typing the antecedents (of the implications), we must take care, since not every principle can be combined with every rule or lexical entry.

Because only morphological head-complement structures are examined in this paper, equation (3) allows us to unify the types associated with (the right-hand sides of) MCOP, MHFP, MSP, MSCP, and MHCR (call the result HCR&Ps), and to regard this feature structure as a restriction for all feature structures belonging to this new (conjunctive) type. All complex morphs (morphs having the attribute MORPHS) must satisfy this type restriction, i.e., must be of the type *HCR&Ps*.

(12) $$\textit{HCR\&Ps} = \textit{MHFP} \sqcap \textit{MSCP} \sqcap \textit{MSP} \sqcap \textit{MCOP} \sqcap \textit{MHCR}$$

(13) HCR&Ps ≡

$$\left[\begin{array}{l} \text{MORPH|FORM order-constituents}(\boxed{1}) \\ \text{SYN|LOC} \left[\begin{array}{l} \text{LEX } + \\ \text{SUBCAT construct-subcat}(\boxed{1}) \\ \text{HEAD } \boxed{2} \end{array}\right] \\ \text{SEM } \boxed{3} \\ \text{MORPHS } \boxed{1} \left[\begin{array}{l} \text{HEAD-MORPH} \left[\begin{array}{l} \text{SYN|LOC} \left[\begin{array}{l} \text{SUBCAT } \boxed{4} \\ \text{HEAD } \boxed{2} \end{array}\right] \\ \text{SEM } \boxed{3} \end{array}\right] \\ \text{COMP-MORPH } \boxed{4} \end{array}\right] \end{array}\right]$$

Trying to encode rules and principles explicitly as elements of a type subsumption lattice (inheritance network) along the lines of HPSG ([36], Ch. 8), requires a REWRITING step: Because of their implicative nature, we cannot state principles DIRECTLY as types—we must rewrite them. 'Rewriting' means first, that only the right side (the consequent) of an implication will be regarded as a type. Second, in order to obtain the force of the antecedent, the type associated with the conjunctive feature structure representing the consequent has to be integrated into the 'right' position in the lattice (cf. *HCR&Ps* in Fig. 2), where 'right' is determined by *taking care that the subsumption relation holds.*[29] Even an equation like (3), containing lots of implications, can then be compiled to form an inheritance hierarchy, consisting only of conjunctive feature types.[30]

The idea of reducing implications to conjunctive types will lead us directly to the structure of the (type/class) subsumption lattice (cf. Fig. 2). Notice that, regarding the laws of feature algebras, we're allowed to 'multiply out' information stored in certain normal forms. This additional step of transformation is necessary to construct hierarchies like those one shown in Fig. 2 and 3.

In the following, we will further motivate and exemplify our approach to derivation by applying it to examples of (morphological) SUFFIXATION and PREFIXATION.

[28] Principles constrain existing types and so must be interpreted as supertypes. In translating a principle—usually expressed as a conditional—into a type, we only use the the right side of the conditional, the consequent. For a motivation, see below.

[29] In general, there's only ONE right position—*the most general position at which the subsumption relation holds.* But this is only true, if we assume a (subsumption) lattice where subsumption is STRICT, i.e., where it is not possible to have two different types standing in a subsumption relation, even though their denotation is the same.

[30] The rewriting step is subtle in that it moves information from object-language implicational statements into restrictions in the type hierarchy which forms the skeleton of the interpretation. On the one hand, because of general laws of interpretation for feature logics, we have the following inference for *Ante*, *Conseq* feature structure terms: from the principle $Ante \Rightarrow Conseq$, we know that $[\![Conseq]\!] \supseteq [\![Ante]\!]$. On the other hand, the principles always ADD information to a feature structure description to which they are applied, so that *Ante* always subsumes *Conseq*, i.e., $[\![Ante]\!] \supseteq [\![Conseq]\!]$. This leads to an effective identification of *Ante* and *Conseq* which is realized in the type hierarchy.

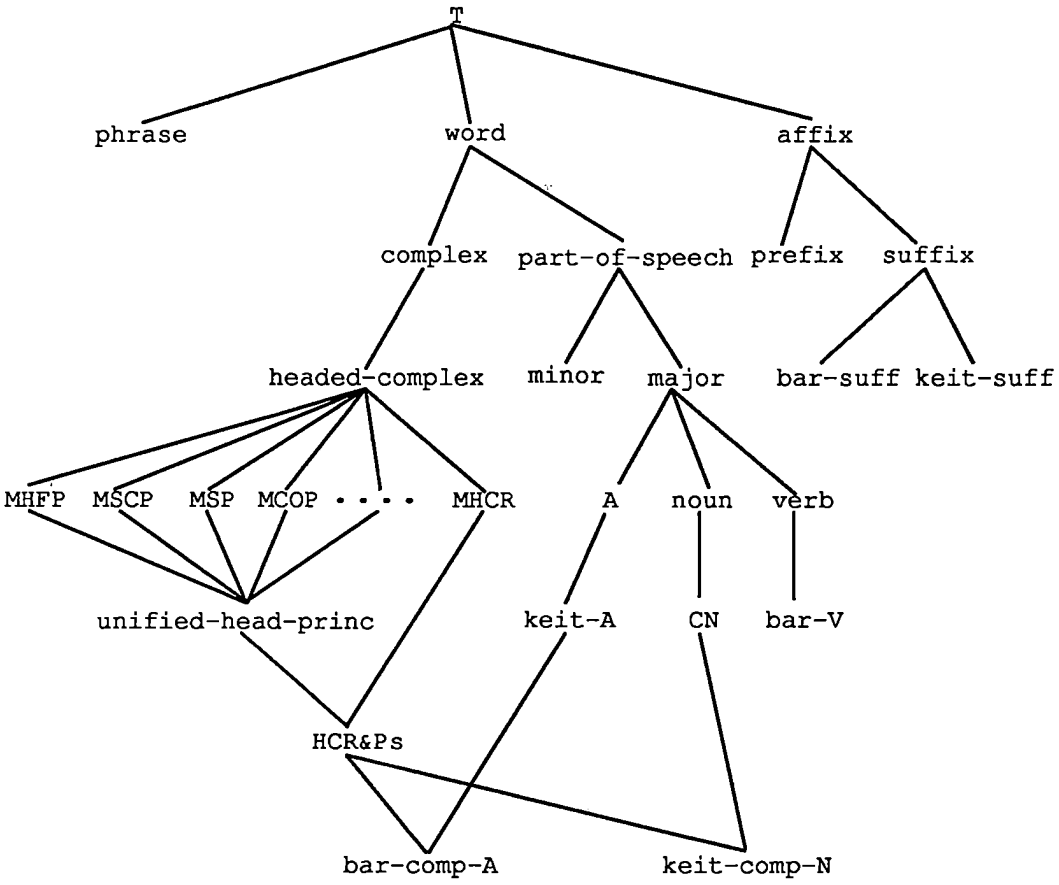


Figure 2: Structure of the inheritance network in case of *-bar* and *-keit* suffixation, including morphological principles and rules. Note that we additionally impose LOCAL constraints on certain classes, especially on *bar-comp-A* and *keit-comp-N*; for motivation, see text. Note further that, although the class of adjectives formed using *-bar* inherits from *A* (adjective) and from *HCR&Ps*, it does NOT inherit from either of its component morphs—*bar-V* or *bar-suffix*.

4.3 *-bar* and *-keit* Suffixation

The treatment of German *-bar* (and also of *-keit*) suffixation is interesting from different points of view and presents severe problems, which can, however, be adequately solved in our approach:

sporadic applicability *-bar* suffixes many verbs, but not all.

partial regularity Many *-bar* derivatives have regular forms and irregular semantics.

category change *-bar* suffixation changes (syntactic) category: `Verb` $\rightsquigarrow$ `Adjective`.

subcategorization change The subcategorization list of `Verb`+*bar* changes: the arity is that of the verb minus 1; the semantic argument positions in the scope of the *-bar* semantics, on the other hand, do NOT change.

Starting with a verb like the German *lesen* (to read), where *-bar* suffixation is regular, we may construct a possible lexicon entry with respect to the inheritance network of Fig. 2.

(14)
$$\text{lesen} \equiv \left[\begin{array}{ll} \textit{bar-V} & \\ \text{MORPH} & \left[\begin{array}{l}\text{STEM "les"}\\ \text{PARADIGM [...]}\end{array}\right] \\ \text{SYN|LOC} & \left[\begin{array}{l}\text{SUBCAT} < \ldots, (\text{NP}_{\boxed{2}}), \text{NP}_{\boxed{1}} > \\ \text{LEX} + \\ \text{HEAD|MAJ V}\end{array}\right] \\ \text{SEM} & \left[\begin{array}{l}\text{RELN read}' \\ \text{SOURCE } \boxed{1} \\ \text{THEME } \boxed{2}\end{array}\right] \end{array}\right]$$

Notice that although *lesen* is syntactically classified as a verb (this is the import of the feature specification $[SYN\text{—}LOC\text{—}HEAD\text{—}MAJ\ V]$), more specifically, it is an instance of the class *bar-V* (verbs that may combine with *-bar*). Note also that we employ here the LEXEME *lesen* rather than, e.g., the infinitive in *lesen*'s paradigm—this is compatible with the fact that only the stem *les-* is found in the derived word.

Moving now to *-bar*, we regard *-bar* (cf. (15)) as the HEAD of the morphological complex with category ADJECTIVE (A); it may function as a head even though it fails to appear as a FREE word. Instead, it occurs only as a BOUND morpheme (instance of the class *bar-suff*; cf. Fig. 2). As a result of the HEAD FEATURE PRINCIPLE the mother obtains automatically the category of the head daughter—and this is exactly what we want, since *les+bar* (*readable*) is an adjective. The head-complement rule, the subcategorization principle and the specification of SYN|LOC|SUBCAT to be an (underspecified) instance of *bar-V* additionally guarantee that *-bar* only combines with *-bar* verbs. Note too, that the value of the attribute LEX in (15) is UNSPECIFIED.[31]

Semantically, *-bar* functions as a modal operator, working on the propositional semantics of *lesen* 'read' to create a proposition asserting the possibility of reading. We note here the co-specification between the semantics of the subcategorized element and the value of the SCOPE attribute in the modal proposition. These assumptions lead us to postulate the following structure for *-bar*:

(15)
$$\text{bar} \equiv \left[\begin{array}{ll} \textit{bar-suff} & \\ \text{MORPH|FORM "bar"} & \\ \text{SYN|LOC} & \left[\begin{array}{l}\text{LEX} \\ \text{HEAD|MAJ A} \\ \text{SUBCAT bar-V}_{\boxed{1}}\end{array}\right] \\ \text{SEM} & \left[\begin{array}{l}\text{OPERATOR } \Diamond \\ \text{SCOPE } \boxed{1}\end{array}\right] \end{array}\right]$$

[31] Under the assumption of Carpenter's "total well-typing" [12], it may be useful to drop the attribute LEX in (15).

The entries for *lesen* and *-bar* together with the head-complement rule and the morphological principles permit us therefore to construct a well-formed feature structure for *lesbar*, and also to reject ill-formed feature structures, so that we can show that (16) is the predicted structure. This meshing of mechanisms ensures that *lesbar* has the right internal structure. The function order-constituents, for instance, determines on the basis of the values of HEAD|MORPH and COMP|MORPH (more exactly, on the basis of the types restricting the attributes) that it has to perform a concatentation (concat) (cf. 16). Additionally, the semantics principle is responsible for the semantics of *lesbar* coming directly from the head daughter *-bar*, while *-bar* takes the complete semantics of *lesen* to fill its attribute SCOPE.

lesbar ≡

(16)
$$\left[\begin{array}{l} \textit{bar-comp-A} \\ \text{MORPH|FORM concat}(\boxed{1},\boxed{2}) \\ \text{SYN|LOC}\left[\begin{array}{l}\text{LEX }+\\ \text{SUBCAT construct-subcat}(\boxed{5})\\ \text{HEAD }\boxed{3}\end{array}\right] \\ \text{SEM }\boxed{4} \\ \text{MORPHS }\boxed{5}\left[\begin{array}{l}\text{HEAD-MORPH}\left[\begin{array}{ll}\text{MORPH|FORM }\boxed{2}\text{ “bar”} & \\ \text{SYN|LOC}\left[\begin{array}{l}\text{SUBCAT }\boxed{6}[\ ]\boxed{7}\\ \text{HEAD }\boxed{3}[\text{MAJ A}]\end{array}\right] & \\ \text{SEM }\boxed{4}\left[\begin{array}{l}\text{OPERATOR }\Diamond\\ \text{SCOPE }\boxed{7}\end{array}\right] & \end{array}\right] \\ \text{COMP-MORPH }\boxed{6}\left[\begin{array}{l}\text{MORPH|STEM }\boxed{1}\text{ “les”}\\ \text{SYN|LOC|LEX }+\\ \text{SEM }\boxed{7}\end{array}\right]\end{array}\right]\end{array}\right]$$

At this point, we have to clarify the complex interaction between the SUBCATEGORIZATION PRINCIPLE and the SEMANTICS PRINCIPLE. The scope on which *-bar* semantically operates is the semantics of *lesen*. The SOURCE role of *lesen* (cf. (14)), whose value is identical to the semantics of the SUBJECT of *lesen*, won't be filled by a phrase licensed by *lesbar* directly, so that it is possible that the SOURCE role in *lesbar* is unfilled. This occurs when no agentive *von* phrase occurs in construction with the adjective, but the attribute is present even where the value is not. In this case, the SOURCE role contains the semantics of an underspecified NP, and the proposition within which it occurs holds whenever there is some value for the NP semantics for which the proposition holds (cf. Flickinger and Nerbonne [18] for a similar treatment of the semantics of the *for* phrase licensed by *easy* adjectives and Karttunen [26] for an analysis of the semantics of the passive *by* phrase along these lines). The intention behind this approach can be appreciated in a concrete example: the sentence *Das Buch ist lesbar* 'The book is readable' doesn't EXPLICITLY state for whom it is possible to read the book. Instead, the reader is mentioned only IMPLICITLY, so that the filler of SOURCE role of *lesbar* might be suspected from extralingustic context, or from the preceding or following discourse, but is not specified directly by the sentence itself.[32] The subcat list of *lesbar* therefore does not include the subject of *lesen*, at least not as an obligatory complement. The object of *lesen* fills the same role filled by the subject of *lesbar*, of course. This is exactly what the function construct-subcat has to accomplish, taking over all other entries from the subcat list of *lesen* to build the right subcategorization list for *lesbar* (cf. (16)).

We provide a single sketch to suggest the possibility of more involved hierarchical structure. In order to construct the word *Les+bar+keit* (*readability*) out of *lesbar*, we have to specify the entry for the suffix *-keit* and the *keit-suff* class (cf. Fig. 2), i.e., it is necessary to state the idiosyncratic properties of *-keit*.

[32] By contrast, there may be a syntactic binding in examples such as *Ich finde das Buch lesbar* 'I find the book readable'.

(17) keit ≡

$$\begin{bmatrix} \textit{keit-suff} \\ \text{MORPH|FORM "keit"} \\ \text{SYN|LOC} \begin{bmatrix} \text{HEAD|MAJ N} \\ \text{SUBCAT keit-A}_{\boxed{1}} \end{bmatrix} \\ \text{SEM} \begin{bmatrix} \text{VAR } \boxed{2} \\ \text{RESTRICTION } \boxed{1} \begin{bmatrix} \text{PRED } \ldots \\ \text{EVENT } \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

By means of the morphological principles and the head-complement rule we may now build an entry for *Lesbarkeit* in the same way shown above (for *lesbar*) with the morphological constituent structure shown in (5).

Among *-bar* verbs such as *lesen*, having perfectly regular *-bar* adjectives (i.e., complex adjectives, containing *-bar* as their head, e.g., *lesbar*), there are others whose derived adjectives are partially irregular, for example with respect to their semantics. As an additional complication, some *-bar* adjectives of these verbs are provided with an additional regular, but non-standard reading. Take for instance the German verb *essen* (to eat):

(18) essen ≡

$$\begin{bmatrix} \textit{bar-V} \\ \text{MORPH|FORM "essen"} \\ \text{SYN|LOC|SUBCAT} < \ldots, (\text{NP}_{\boxed{2}}), \text{NP}_{\boxed{1}} > \\ \text{SEM} \begin{bmatrix} \text{RELN eat}' \\ \text{SOURCE } \boxed{1} \\ \text{THEME } \boxed{2} \end{bmatrix} \end{bmatrix}$$

The non-standard (semantically regular) reading of *eßbar* can be built in a regular way by means of the mechanisms described above, taking *essen* and *-bar* to form a complex word.

(19) $\text{eßbar}^{non-stand}$ ≡

$$\begin{bmatrix} \textit{bar-comp-A} \\ \text{SEM } \boxed{1} \begin{bmatrix} \text{OPERATOR } \Diamond \\ \text{SCOPE} \begin{bmatrix} \text{RELN eat}' \\ \text{SOURCE } \ldots \end{bmatrix} \end{bmatrix} \\ \text{MORPHS|HEAD-MORPH|SEM } \boxed{1} \end{bmatrix}$$

The standard reading of *eßbar* on the other hand is 'edible' (the property of an object which can SAFELY be eaten). Constructing the standard reading (with irregular semantics) for *eßbar* can be done in our approach in two different ways:

1. We do NOT regard *eßbar* as an instance of the class *bar-comp-A*; instead, *eßbar* is entered separately into the lexicon. We then have to specify at least that the semantics of (20) is different from that of (19), although the MORPH and SYN properties seem to remain the same. A treatment of this kind leads us to the question of whether the feature structure (20) actually will have MORPH daughters—since no use need be made of the structure.

2. The semantics of (19) (the entry which was built regularly) is modified by using OVERWRITING, DEFAULT UNIFICATION or other nonmonotonic mechanisms to enforce the standard reading. In this case, *eßbar* (21) belongs to the class *bar-comp-A*, because all other properties remain the same. We would follow Bouma [7] in the use of default unification (as a basis for default inheritance).

(20) eßbar^{stand} ≡

$$\begin{bmatrix} \textit{?-A} \wedge \neg \textit{bar-comp-A} \\ \text{SEM} \begin{bmatrix} \text{RELN safely-eat}' \\ \text{SOURCE } \ldots \end{bmatrix} \\ \text{MORPHS ???} \end{bmatrix}$$

(21)
$$\text{eßbar}^{stand'} \equiv \text{eßbar}^{non-stand} \oplus \left[\text{SEM} \left[\begin{array}{l}\text{RELN safely-eat}' \\ \text{SOURCE } \ldots\end{array}\right]\right]$$
$$= \left[\begin{array}{l}\textit{bar-comp-A} \\ \text{SEM } \boxed{1}\left[\begin{array}{l}\text{RELN safely-eat}' \\ \text{SOURCE } \ldots\end{array}\right] \\ \text{MORPHS|HEAD-MORPH|SEM } \neg\boxed{1}\end{array}\right]$$

The advantage of the second approach is that regular properties of partially regular derivations need not be specified redundantly, as would be the case in the first approach. The use of default specifications thus obtains the same advantages in DERIVATION that Flickinger et al. [19] and Evans & Gazdar [16] have shown in word-class definitions. Defaults, together with the possibility of overwriting defaults in more specific definitions may turn out to be even more important in connection with the analysis of derivational relationships, since these are notoriously irregular in morphological form, syntactic feature assignment, and semantics (cf. Toman's book-length study on *-bar* adjectives [47] for ample illustration).

The typed approach to *-bar* suffixation allows us to prevent ill-formed *-bar* adjectives; e.g., we have to rule out the combination of ***haben*** (to have) together with *-bar*. This is very easy to achieve under the assumption that ***haben*** doesn't belong to the *-bar* verb class *bar-V*, but instead to another class (say *?-V*), thus preventing ***haben*** from combining with *-bar*—therefore *hab+bar* is disallowed.

(22)
$$\text{haben} \equiv \left[\begin{array}{l}\textit{?-V} \wedge \neg\textit{bar-V} \\ \ldots\ldots \\ \ldots\ldots\end{array}\right]$$

It is nevertheless possible to construct ***handhab+bar*** 'manageable' out of ***handhaben*** 'to handle, manage', since ***haben*** and ***handhaben*** are distinct lexemes. By explicitly encoding ***handhaben*** as an entry of type *bar-V*, we can move to a legal description of ***handhabbar***.

(23)
$$\text{handhaben} \equiv \left[\begin{array}{l}\textit{bar-V} \\ \ldots\ldots \\ \ldots\ldots\end{array}\right]$$

(24)
$$\text{handhabbar} \equiv \left[\begin{array}{l}\textit{bar-comp-A} \\ \ldots\ldots \\ \ldots\ldots\end{array}\right]$$

The structure of the class hierarchy (cf. Fig. 2) ultimately leads us to a treatment of suffixation, esp. *-bar* and *-keit* suffixation (and also of prefixation in general), where the whole process can be described within the framework of UNIFICATION-BASED INHERITANCE REASONING. On what grounds are we allowed to state such a thesis? At first sight, this statement seems to stand in contrast with the claim made above, that NAIVE inheritance is not enough. But we do not rely on naive inheritance as the only mechanism. So we turn now to an examination of why this is so.

We noted earlier that *les+bar* and *Les+bar+keit* are legal lexemes because they satisfy all principles whose left sides they match (implying that they have to meet the right sides too), and because they are composed out of lexicon entries by means of rules. In doing realistic parsing or generation, we might assume additional CONTROL MACHINERY outside of the grammar/lexicon, which uses principles and rules to accept or reject, or alternatively, to generate well-formed (complex) phrases.

Because we regard principles as well as rules as types, equation (3) allows us to employ the laws of feature algebras to construct new types (call them PRECATEGORIES), which are subsumed by all principles having a more general left side and by at least one rule (cf. equations (12) and

(13)). Complex words/morphemes like *lesbar* on the other hand will then be subsumed by such precategories.

It is now easy to see that the processes described up to now can be represented entirely via inheritance of a sophisticated kind (effectively constraint resolution). This very interesting observation is motivated as follows: it is possible to define new legal complex word classes by inheriting from precategories as well as from simple lexical categories (cf. subtypes of *part-of-speech* in Fig. 2) and by stating additional local constraints for the class in question. Looking at Fig. 2, *bar-comp-A* (complex adjectives with head daughter *-bar*) and *keit-comp-N* (complex nouns with head daughter *-keit*) are classes of such a kind.

Let's have a closer look at *bar-comp-A* and *keit-comp-N*: *bar-comp-A* inherits from *HCR&Ps* and *A*, but also enforces idiosyncratic constraints, which have to be satisfied by words that are members of this class:

$$\text{bar-comp-A} = \text{HCR\&Ps} \wedge \text{A} \wedge \left[\text{MORPHS} \left[\begin{array}{l} \text{HEAD-MORPH} \ [\ \textit{bar-suff}\ ] \\ \text{COMP-MORPH} \ [\ \textit{bar-V}\ ] \end{array} \right] \right] \tag{25}$$

It's very important to constrain HEAD-MORPH and COMP-MORPH to be of type *bar-suff* resp. *bar-V* respectively,[33] in order to get the right feature structure for *bar-comp-A*. We also require that the adjective class *A* is associated with the following feature structure.

$$\text{A} \equiv \left[\begin{array}{l} \text{MORPH} \ \ldots\ldots \\ \text{SYN|LOC} \left[\begin{array}{l} \text{LEX} + \\ \text{MOD} < \ldots, \text{NP} > \\ \text{HEAD|MAJ A} \end{array} \right] \\ \text{SEM} \ \ldots\ldots \end{array} \right] \tag{26}$$

Since furthermore *HCR&Ps* (cf. (13)) is also associated with a feature structure, it's not difficult to construct the prototypical feature structure for *bar-comp-A* by unifying all the information. But once this is achieved, we may construct an entry for *lesbar* by creating an INSTANCE of the class *bar-comp-A* and stating that the complement daughter of this instance is *lesen*, i.e. COMP-MORPH must have as value a feature structure equal to that of the lexeme *lesen* (cf. (14)).

$$\text{lesbar} = \text{bar-comp-A} \wedge [\ \text{MORPHS|COMP-MORPH}\ \textit{lesen}\ ] \tag{27}$$

Notice that the feature structure for (27) corresponds to the one for *les+bar* (cf. (16)) provided earlier. In entirely the same fashion, we might INSTANTIATE feature structures for new words like *Les+bar+keit*, which belong to the class *keit-comp-N*. For that purpose, we have to define the class *CN*, which *keit-comp-N* inherits from (cf. Fig. 2).

$$\text{CN} \equiv \left[\begin{array}{l} \text{MORPH} \ \ldots\ldots \\ \text{SYN|LOC} \left[\begin{array}{l} \text{LEX} + \\ \text{SUBCAT} < \ldots, \{\text{Det, PosP}\} > \\ \text{HEAD|MAJ N} \end{array} \right] \\ \text{SEM} \ \ldots\ldots \end{array} \right] \tag{28}$$

With these definitions in mind, we're able to state the dependence of the complex word class *keit-comp-N* on *HCR&Ps* and *A*, in perfect analogy to equation (25):

$$\text{keit-comp-N} = \text{HCR\&Ps} \wedge \text{CN} \wedge \left[\text{MORPHS} \left[\begin{array}{l} \text{HEAD-MORPH} \ [\ \textit{keit-suff}\ ] \\ \text{COMP-MORPH} \ [\ \textit{keit-A}\ ] \end{array} \right] \right] \tag{29}$$

[33] Strictly speaking: The value of HEAD-MORPH is a fully expanded instance of type *bar-suff*, whereas COMP-MORPH is bound to an underspecified instance of type *bar-V*, because for instance the value of MORPH|FORM is unspecified.

We may then represent a feature structure like *Lesbarkeit* by instantiating *keit-comp-N* and imposing a local constraint on this instance:

(30) Lesbarkeit = keit-comp-N $\wedge$ [MORPHS|COMP-MORPH *lesbar*]

The restriction that COMP-MORPH must be of type *keit-A* (cf. (29)) also allows that COMP-MORPH may be an instance of type *bar-comp-A* defined earlier (cf. (25)) because *keit-A* is a supertype of *bar-comp-A* (cf. Fig. 2).[34]

This last point should only be seen as a remark to practitioners working on computational lexicons. To enforce, for instance, that elements of the lexicon MUST have their PHON attributes filled, one can use the mechanisms discussed in the footnote.

At the beginning of this section we listed four analytical problems for the description of *-bar* adjectives. During the course of this section we have proposed solutions to these which we summarize here:

Problem	Solution via internal lexical rules
sporadic applicability	use type restrictions
partial regularity	apply non-mon mechanisms or introduce additional classes
category change	treat affix as head of morphological complex
subcategorization change	employ functional dependencies

Before closing this section, we would like to note that the derivational view of passive, which we promised to sketch in the introduction, may be developed straightforwardly on the basis of the analysis of *-bar* sketched here. In particular, the class of verbs involved here is very nearly the same, and the effects on subcategorization (via construct-subcat) identical.

4.4 *Vor-* Prefixation

In this section we investigate the phenomenon of PREFIXATION, focusing for further depth on a specific prefix, namely *vor-*.[35] What prefixes and suffixes have in common is that they serve as

[34] What we said up to now isn't the whole truth. There's an additional restriction we're faced with: given what we have said up till now, nothing prevents us from creating instances which are UNDERSPECIFIED with respect to certain attributes.Such instances do not represent real words and therefore must be forbidden.Take for instance *CN*, the class of common nouns. We might create an instance without specifying the value for MORPH|FORM. Although this instance would be of class *CN*, it couldn't be used by any speaker. Trying to build an instance of type *word* (*lexical-sign*; the most general type of the lexical subsumption hierarchy), which is only constrained to possess the attributes PHON, SYN, and SEM according to Pollard & Sag, would be an extreme case of this shortcoming. This observation holds for PHRASAL SIGNS too, because it is possible to generate sentences without any phonological content, when assuming lexical entries (instances) which are empty with respect to their PHON attribute. As these examples suggest, the possibility of creating underspecified instances depends on there being incorrect grammatical specifications. There are at least two NON-MONOTONIC approaches in order to repair this defect:

1. We introduce a special type, say *undefined*, whose extension is a unique constant (call it NONE). Those attributes we require to be filled at run time (instantiation time) are assigned the type restriction $\neg$ *undefined* and the contradictory value NONE at definition time. Now, if we carry out a type check at run time, we're able to recognize whether the critical attributes have been assigned real values (by overwriting NONE with a different value), i.e. not to be of type *undefined*.
2. We classify the relevant attributes with a special atom called ANY, which functions in the same way as the ANY works in Kay's FUG: ANY may successfully unify with every object, except $\bot$ (Bottom); i.e., the semantics of ANY is that of $\top$ (Top). But when the instantiation of a word is done, all ANYs have to be removed (must be unified 'out'), because ANY henceforth behaves like $\bot$. (Dörre & Eisele, [15], pp. 18, give a formalization of ANY in terms of so-called *meta-constraints*.)

There will be a third possibility of repair, if the underlying (feature) logic allows us to state that certain (underspecified) types (classes) CANNOT be instantiated. Instead, these classes only serve as choice points in our linguistic ontology—reflecting the distinction between REGULARITIES, SUBREGULARITIES, and EXCEPTIONS and enabling a finer granularity of lexical knowledge.

[35] As we mentioned above, we have taken the prefix *Vor-* as an example to show how certain phenomena can be handled in our approach. The assumption that *Vor-* functions semantically as an operator, working on the semantics of the noun (cf. (32)), is of course not an in-depth analysis and may not be useful in real applications. There are other prefixes like *Anti-* or *Ur-* having similar properties, but their semantics is even more complicated.

heads in our simple head-complement approach. *Vor-* prefixation is in many respects different from *-bar* suffixation and has special properties that makes it interesting for expository purposes:

sporadic applicability *Vor-* prefixes many nouns (e.g. *Vorversion*, *Vorgaben*, *Vorzelt*, *Vorzimmer* and *Vorabend*), but not all.

partial regularity Many *Vor-* derivatives have regular morphological forms and irregular semantics.

category constant The (syntactic) category of the *Vor-* derivative does not change.

subcategorization constant The subcategorization list of the derived complex word (*Vor*+Noun) is taken over from the complement, the noun and does not undergo any changes.

iterability The prefix *Vor-* can be applied iteratively.

Let's examine a noun that may combine with *Vor-* to form a complex noun, viz., the German *Version* 'version':

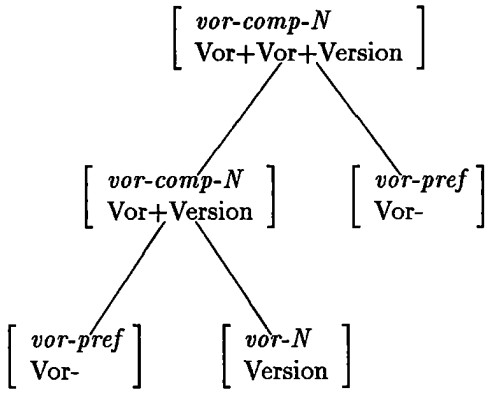
(31)
$$\text{Version} \equiv \left[\begin{array}{l} \textit{vor-N} \\ \text{MORPH|FORM "Version"} \\ \text{SYN|LOC} \left[\begin{array}{l} \text{SUBCAT} < \ldots, \{\text{Det} , \text{PosP}\} > \\ \text{LEX} + \\ \text{HEAD|MAJ N} \end{array}\right] \\ \text{SEM} \left[\text{PRED version}' \right] \end{array}\right]$$

Trying to encode *Vor-* is a bit harder, because *Vor-* not only works on nouns, but also on certain verbs (e.g., *vorgehen*, *vorarbeiten*, or *vorlaufen*). In order to represent this fact, we again make use of distributed disjunctions, which were employed in Section 3 above to encode inflectional paradigms.

(32)
$$\text{Vor} \equiv \left[\begin{array}{l} \textit{vor-pref} \\ \text{MORPH|FORM "Vor"} \\ \text{SYN|LOC} \left[\begin{array}{l} \text{HEAD|MAJ } \{_{\$1}\ \text{N} , \text{V}\} \\ \text{SUBCAT } \{_{\$1}\ \text{vor-N}_{\boxed{1}} , \text{vor-V}_{\boxed{2}}\} \end{array}\right] \\ \text{SEM} \left[\begin{array}{l} \text{OPERATOR vor}' \\ \text{SCOPE } \{_{\$1}\ \boxed{1} , \boxed{2}\} \end{array}\right] \end{array}\right]$$

It is important to understand the intention behind the use of the distributed disjunction in *Vor-*: if *Vor-* combines with a *Vor-* noun, it will be classified as a noun, but if it binds a *Vor-* verb, it creates a verb. Moreover, the head feature principle takes care that the mother of the morphological phrase will be assigned the same category that the prefix *Vor-* bears. That's the main reason why recursion is possible—the new word, e.g., *Vor+Version*, will again be classified as a noun and could then combine with a new *Vor-* (cf. (33)) in the same way as described before (actually it is now a COMPLEX noun with internal structure; cf. Fig. 3).

(33)



The value of SCOPE under path MORPHS|HEAD-MORPH|SEM now will be assigned by means of structure sharing, coming directly from the semantics of the value of SYN|LOC|SUBCAT, no matter which value of the distributed disjunction is taken (cf. (32)). Finally, by virtue of the semantics principle, *Vorversion* will get its semantics from its head daughter *Vor-*. construct-subcat is again responsible for constructing the right subcategorization list for *Vorversion*: because *Vor-* is the head morph, construct-subcat can detect that the value of SYN|LOC|SUBCAT has to be equal to the entire subcat list of the complement. With these things in mind, we may now construct, with the assistance of the above mentioned principles and the head-complement rule, an admissible feature structure for *Vorversion*, which has the following form:

(34) Vorversion ≡

$$
\left[\begin{array}{l}
\textit{vor-comp-N} \\
\text{MORPH|FORM concat}(\boxed{1}, \boxed{2}) \\
\text{SYN|LOC} \left[\begin{array}{l} \text{LEX} + \\ \text{SUBCAT construct-subcat}(\boxed{5}) \\ \text{HEAD } \boxed{3} \end{array}\right] \\
\text{SEM } \boxed{4} \\
\text{MORPHS } \boxed{5} \left[\begin{array}{ll}
\text{HEAD-MORPH} & \left[\begin{array}{l} \textit{vor-pref} \\ \text{MORPH|FORM } \boxed{1} \text{ "Vor"} \\ \text{SYN|LOC} \left[\begin{array}{l} \text{SUBCAT } \boxed{6}[\ ]_{\boxed{7}} \\ \text{HEAD } \boxed{3}[\text{MAJ N}] \end{array}\right] \\ \text{SEM } \boxed{4} \left[\begin{array}{l} \text{OPERATOR vor}' \\ \text{SCOPE } \boxed{7} \end{array}\right] \end{array}\right] \\
\text{COMP-MORPH } \boxed{6} & \left[\begin{array}{l} \textit{vor-N} \\ \text{MORPH|FORM } \boxed{2} \text{ "Version"} \\ \text{SYN|LOC} \left[\begin{array}{l} \text{LEX} + \\ \text{SUBCAT} < \ldots, \{\text{Det , PosP}\} > \end{array}\right] \\ \text{SEM } \boxed{7} \end{array}\right]
\end{array}\right]
\end{array}\right]
$$

It may be useful to split up the entry for *Vor-* (cf. (32)), distributing its semantics among two feature structures—one (Vor^1), which combines only with nouns, another one (Vor^2), which binds instead verbs. But this kind of representation is rather a matter of style.

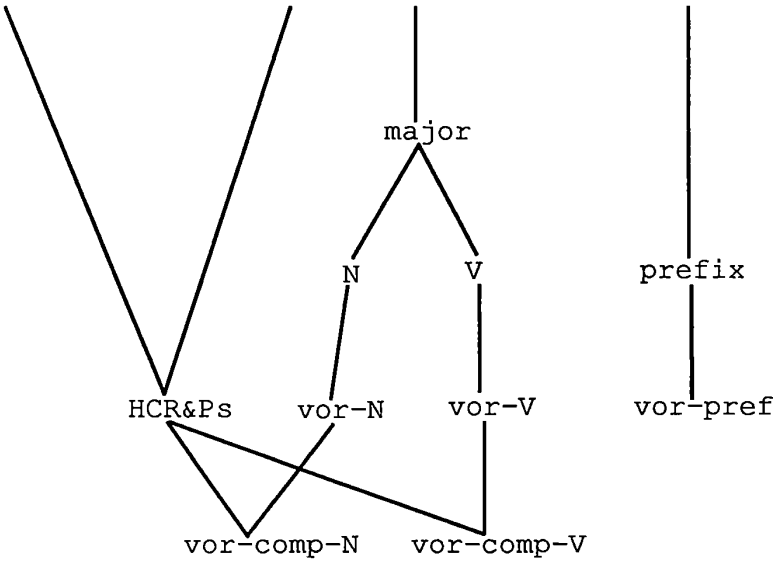


Figure 3: Structure of the inheritance network in case of *Vor-* prefixation, regarding the principles and the rule. Note, that we additionally impose LOCAL constraints on certain classes, especially on *vor-comp-N* and *vor-comp-V*.

(35)
$$\text{Vor}^1 \equiv \left[\begin{array}{l} \textit{vor-pref} \\ \text{MORPH|FORM "Vor"} \\ \text{SYN|LOC} \left[\begin{array}{l} \text{HEAD|MAJ N} \\ \text{SUBCAT vor-N}\boxed{1} \end{array}\right] \\ \text{SEM} \left[\begin{array}{l} \text{OPERATOR vor}' \\ \text{SCOPE } \boxed{1} \end{array}\right] \end{array}\right]$$

(36)
$$\text{Vor}^2 \equiv \left[\begin{array}{l} \textit{vor-pref} \\ \text{MORPH|FORM "Vor"} \\ \text{SYN|LOC} \left[\begin{array}{l} \text{HEAD|MAJ V} \\ \text{SUBCAT vor-V}\boxed{1} \end{array}\right] \\ \text{SEM} \left[\begin{array}{l} \text{OPERATOR vor}'' \\ \text{SCOPE } \boxed{1} \end{array}\right] \end{array}\right]$$

We described *-bar/-keit* suffixation above by means of unification-based inheritance; once we assume an inheritance network such as Fig. 3, we may analyze *Vor-* prefixation (including recursion) similarly.

Analogous to (25) and (29), we may state the right definitions for *vor-comp-N* and *vor-comp-V* with respect to Fig. 3.

(37)
$$\text{vor-comp-N} = \text{HCR\&Ps} \wedge \text{vor-N} \wedge \left[\begin{array}{l} \text{MORPHS} \left[\begin{array}{l} \text{HEAD-MORPH } [\textit{vor-pref}] \\ \text{COMP-MORPH } [\textit{vor-N}] \end{array}\right] \end{array}\right]$$

(38)
$$\text{vor-comp-V} = \text{HCR\&Ps} \wedge \text{vor-V} \wedge \left[\begin{array}{l} \text{MORPHS} \left[\begin{array}{l} \text{HEAD-MORPH } [\textit{vor-pref}] \\ \text{COMP-MORPH } [\textit{vor-V}] \end{array}\right] \end{array}\right]$$

The iterated application of *Vor-* will be guaranteed by using the RECURSIVE type definition of *vor-comp-N*: the value of the attribute MORPHS|COMP-MORPH in *vor-comp-N* is required to be of type *vor-N*. Because *vor-N* subsumes *vor-comp-N*, we're allowed in particular to require that the value of MORPHS|COMP-MORPH should be an instance of type *vor-comp-N*, and this corresponds to a potentially infinite iteration of *Vor-*. But how do we block infinite recursion in cases of concrete words? Only instances of minimal type *vor-N* will stop parsing or generation, because those instances don't have any internal constituent structure (no MORPHS attribute), i.e., there's no way to expand them further.[36] It is the indirect self-reference that is responsible for the recursive nature of *vor-comp-N*. If we now still require that the value of MORPHS.COMP-MORPH not be underspecified, the aim of describing *Vor-* prefixation by inheritance only is fully realized.

Constructing an entry for *Vorversion* is done in a trivial way by instantiating *vor-comp-N* and by imposing an additional restriction on that instance, namely that the complement daughter must hold a feature structure representing *Version*.

(39) Vorversion = vor-comp-N $\wedge$ [MORPHS|COMP-MORPH *Version*]

5 Summary and Conclusions

In this section we summarize the results of the present study, pointing out areas which we have yet to investigate, but which seem pertinent or promising; the further investigation of these areas offers the best practical means to advances in this approach to computational lexicology.

The results of the present study may be viewed as follows: feature-based formalisms (such as PATR-II or the HPSG formalism) have been successful in the description of SYNTAGMATIC grammatical relations—the relations between the various syntactic parts of an utterance token. The present study attempts to demonstrate that the feature description languages developed for this purpose may also be applied fruitfully to PARADIGMATIC relations—the relations between words and their common alternatives in utterances.[37] We have examined proposals here for the representation of inflectional and derivational relations. These relations are purely lexical, since they may never result in syntactic relations, so that a distinctly lexical status accrues to them even in a highly "lexicalized theory" such as HPSG. This is the sense in which the present contribution claims to develop further the theory of the LEXICON for feature-based theories of language. It is of course clear to us that several aspects of this theory—most clearly its syntactic content—have been under development for some time, but the larger theoretical picture had not been clarified.

We have attempted to provide that clarification, so that our proposals here have thus been programmatic, but we provide concrete elaborations in two central areas of paradigmatic relations, inflection and derivation. Our proposal for inflection may be seen as a variant of one first proposed in DATR: we characterize the inflectional variants of a lexeme as alternative (disjunctive) realizations. The basic insight of DATR is easily accommodated within the language of feature structure descriptions. Alternative realizations of lexemes—paradigms—are represented using the technical tool of distributed disjunctions (although there are several equivalent means of representation). Our proposal for derivation may be seen as an application of the HPSG treatment of syntactic structure in feature structure formalism. Just as HPSG characterizes phrase structure rules via descriptions of the output phrases created by the rule, so we propose characterizing (derivational) word formation rules via recursive constraints on complex lexemes—those which linguists would regard as created by the "rule". This contrasts with the usual treatment in feature-based theories, which construes derivational rules (in fact, normally ALL lexical rules) as mappings within the algebra of feature structures. The latter proposal relies on subsidiary functions or relations to characterize lexical rules, while our own characterization remains within the

[36] When we say *an instance of minimal type vor-N*, we exclude instances of types more specific than *vor-N*. This corresponds roughly to CLASSIFICATION in KL-ONE-like knowledge representation systems. By making a distinction between class and instance, and assuming totally well-typed feature structures, this goal is easily reached.

[37] We appreciate Carl Pollard's suggesting this contrast to us.

language of feature-structure description. Our proposal is probably preferable in direct proportion to the degree to which derivational structure employs mechanisms which feature formalisms describe well—inheritance, typing, and shared structure. We suggest that the inheritance exploited in structured lexicons finds very apt application in derivation as well.

We do not imagine that this paper represents a mature presentation of what a feature-based lexicon ought to be capable of. In particular, we continue to puzzle over several areas: the correct representation of compounding; the treatment of idioms ; the most attractive analysis of so-called "zero-derivation" (e.g., the relation between verbal participles and the adjectives derived from them); the proper interface to allomorphy—both theoretically and practically; the role of morpheme-based generalizations; and the question of flexible access (for both parsing and generation) to lexical structure. We regard these as challenging issues for future work in feature-based lexical analysis.

The further investigation of these areas, together with continuing work on elaborations and alternatives to the analyses suggested here, offers the best practical means to advances in this approach to computational lexicology.

References

[1] J.H. Allen and H.B.Greenough. *New Latin Grammar.* Ginn and Company, Boston, 1903.

[2] Stephen R. Anderson. Inflection. In Michael Hammond and Michael Noonan, editors, *Theoretical Morphology*, pages 23–43. Academic Press, Orlando, 1988.

[3] Stephen R. Anderson. Morphological theory. In Frederick J. Newmeyer, editor, *Linguistics: The Cambridge Survey*, volume 1, pages 146–91. Cambridge University Press, Cambridge, 1988.

[4] Emmon Bach. Categorial grammars as theories of language. In Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler, editors, *Categorial Grammars and Natural Language Structures*, pages 17–34. Reidel, Dordrecht and Boston, 1988.

[5] Rolf Backofen, Lutz Euler, and Günter Görz. Towards the integration of functions, relations and types in an AI programming language. In *Proceedings of GWAI-90*, Berlin, 1990. Springer.

[6] Steven Bird. Prosodic morphology and constraint-based phonology. Research Paper EUCCSRP-38, Centre for Cognitive Science, University of Edinburgh, 1990.

[7] Gosse Bouma. Defaults in unification grammar. In *Proceedings of the 28th Annual Meeting of the ACL*, pages 165–172. Association for Computational Linguistics, 1990.

[8] Joan Bresnan, editor. *The Mental Representation of Grammatical Relations.* MIT Press, Cambridge, Mass., 1982.

[9] Jonathan Calder. Paradigmatic morphology. In *Proceedings of the 5th Annual Meeting of the European Association for Computational Linguistics*, pages 58–65, 1989.

[10] Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, 17(4):471–522, 1985.

[11] Bob Carpenter. *The Logic of Typed Feature Structures.* Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, to appear, 1992.

[12] Bob Carpenter, Carl J. Pollard, and Alex Franz. The specification and implementation of constraint-based unification grammar. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, 1991.

[13] William Cook, Walt Hill, and Peter Canning. Inheritance is not subtyping. In *ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 1990.

[14] Jochen Dörre and Andreas Eisele. Determining consistency of feature terms with distributed disjunctions. In Dieter Metzing, editor, *Proceedings of GWAI-89 (15th German Workshop on AI)*, pages 270–279, Berlin, 1989. Springer-Verlag.

[15] Jochen Dörre and Andreas Eisele. A comprehensive unification-based grammar formalism. Technical Report Deliverable R3.1.B, DYANA, Centre for Cognitive Science, Edinburgh, 1991.

[16] Roger Evans and Gerald Gazdar. The DATR papers. Technical Report SCRP 139, School of Cognitive and Computing Sciences, University of Sussex, 1990.

[17] Daniel Flickinger. *Lexical Rules in the Hierarchical Lexicon.* PhD thesis, Stanford University, 1987.

[18] Daniel Flickinger and John Nerbonne. Inheritance and complementation: A case study of *easy* adjectives and related nouns. *Computational Linguistics*, 18, 1991.

[19] Daniel Flickinger, Carl Pollard, and Thomas Wasow. Structure-sharing in lexical representation. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, 1985.

[20] Gerald Gazdar. Linguistic applications of default inheritance mechanisms. In Peter Whitelock, Mary McGee Wood, Harald L. Somers, Rod Johnson, and P.Bennett, editors, *Linguistic Theory and Computer Applications.* Academic Press, London, 1987.

[21] Gerald Gazdar. An introduction to DATR. In Roger Evans and Gerald Gazdar, editors, *The DATR Papers*, number SCRP 139 in Cognitive Science Research Reports, pages 1–14. School of Cognitive and Computing Sciences, University of Sussex, 1990.

[22] Gerald Gazdar, Ewan Klein, Geoffrey Pullum, and Ivan Sag. *Generalized Phrase Structure Grammar.* Harvard University Press, 1985.

[23] Jack Hoeksema. *Categorial Morphology.* Garland, New York, 1985.

[24] Mark Johnson. *Attribute Value Logic and the Theory of Grammar.* Center for the Study of Language and Information, Stanford, 1988.

[25] Ronald Kaplan and Joan Bresnan. Lexical-functional grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, Mass, 1982.

[26] Lauri Karttunen. D-PATR: A development environment for unifiaction-based grammars. Technical Report CSLI-86-61, Center for the Study of Language and Information, Stanford University, 1986.

[27] Robert T. Kasper and William C. Rounds. A logical semantics for feature structures. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pages 257–266, Columbia University, 1986.

[28] Andreas Kathol. Verbal and adjectival passives in german. In *MIT Working Papers in Linguistics*, volume 14. MIT, 1991.

[29] Martin Kay. Functional unification grammar: A formalism for machine translation. In *Proceedings of COLING*, pages 75–78, 1984.

[30] Martin Kay. Parsing in functional unification grammar. In David Dowty and Lauri Karttunen, editors, *Natural Language Parsing.* Cambridge University Press, Cambridge, England, 1985.

[31] James Kilbury, Petra Naerger, and Ingrid Renz. DATR as a lexical component for PATR. In *Proceedings of the 6th Annual Meeting of the European Chapter of the Association for Computational Linguistics*, 1991.

[32] Kimmo Koskenniemi. Two-level model for morphological analysis. In *Proceedings of the Eighth Annual Joint Conference on Artificial Intelligence*, pages 683–685, 1983.

[33] Hans-Ulrich Krieger. Eliminating complex non-reversible functions in derivational morphology. Technical Report xx, Deutsches Forschungsinstitut für Künstliche Intelligenz, Saarbrücken, Germany, 1991.

[34] P.H.Matthews. *Inflectional Morphology: A Theoretical Study Based on Aspects of Latin Verb Conjugation.* Cambridge University Press, Cambridge, England, 1972.

[35] P.H.Matthews. *Morphology.* Cambridge University Press, Cambridge, England, 1974.

[36] Carl Pollard and Ivan Sag. *An Information-Based Theory of Syntax and Semantics, Vol.I.* University of Chicago Press, 1987.

[37] Mike Reape. An introduction to the semantics of unification-based grammar formalisms. Technical Report R3.2.A, DYANA, University of Edinburgh, 1991.

[38] Graeme D. Ritchie, Stephen G. Pulman, Alan W. Black, and Graham Russell. A computational framework for lexical description. *Computational Linguistics*, 13(3-4):290–307, 1987.

[39] William C. Rounds and Robert T. Kasper. A complete logical calculus for record structures representing linguistic information. In *Proceedings of the 15th Annual Symposium of the on Logic in Computer Science*, Cambridge, 1986.

[40] Graham Russell, John Carroll, and Susan Warwick. Multiple default inheritance in a unification-based lexicon. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, 1991.

[41] Jerry Sadock. Autolexical syntax: A proposal for the treatment of noun incorporation and similar phenomena. *Natural Language and Linguistic Theory*, 3:379–439, 1985.

[42] Ivan Sag and Carl Pollard. Head-driven phrase structure: An informal synopsis. Technical Report CSLI-87-89, Center for the Study of Language and Information, Stanford University, 1987.

[43] Stuart Shieber. *An Introduction to Unification-Based Approaches to Grammar.* Center for the Study of Language and Information, Stanford University, 1986.

[44] Stuart Shieber, Hans Uszkoreit, J. Robinson, and M. Tyson. The formalism and implementation of PATR-II. In *Research on Interactive Acquisition and Use of Knowledge.* AI Center, SRI International, Menlo Park, Cal., 1983.

[45] Koenraad De Smedt. Using object-oriented knowledge representation techniques in morphology and syntax programming. In *Proceedings of the 1984 European Conference on Artificial Intelligence*, pages 181–184, 1984.

[46] Gert Smolka. A feature logic with subsorts. Technical Report 33, WT LILOG–IBM Germany, 1988.

[47] Jindrich Toman. *Wortsyntax: Eine Diskussion ausgewählter Probleme deutscher Wortbildung.* Niemeyer, Tübingen, 1983.

[48] Harald Trost. The application of two-level morphology to non-concatenative german morphology. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING)*, 1990.

[49] Arnold M. Zwicky. How to describe inflection. In *Proceedings of the 11th Annual Meeting of the Berkeley Linguistics Society*, pages 372–86, 1985.

[50] Arnold M. Zwicky. Inflectional morphology as a subcomponent of grammar. In *Proceedings of the 3rd International Morphology Meeting*, 1988.

A Practical Approach to Multiple Default Inheritance for Unification-Based Lexicons

Graham Russell
Afzal Ballim
John Carroll*
Susan Warwick-Armstrong

ISSCO, 54 route des Acacias,
1227 Geneva, Switzerland
elu@divsun.unige.ch

September 2, 1991

1 Introduction

Natural language lexicons form an obvious application for techniques involving default inheritance developed for knowledge representation in AI. Many of the schemes that have been proposed are highly complex – simple tree-form taxonomies are thought to be inadequate, and a variety of additional mechanisms are employed. As Touretzky et al. (1987) show, the intuitions underlying the behaviour of such systems may be unstable, and in the general case they are intractable (Selman and Levesque, 1989).

It is an open question whether the lexicon requires this level of sophistication – by sacrificing some of the power of a general inheritance system one may arrive at a simpler, more restricted, version, which is nevertheless sufficiently expressive for the domain. The particular context within which the lexicon described here has been devised seems to permit further reductions in complexity. It has been implemented as part of the ELU[1] unification grammar development environment for research in machine translation, comprising parser, generator, lexicon, and transfer mechanism.

2 Overview of Formalism

An ELU lexicon consists of a number of 'classes', each of which is a structured collection of constraint equations and/or macro calls encoding information common to a set of words, together with links to other more general 'superclasses'. Lexical entries are themselves classes,[2] and any information they contain is standardly specific to an individual word;

*current address: Cambridge University Computer Laboratory, New Museums Site, Pembroke Street, Cambridge CB2 3QG, UK.

[1] "Environnement Linguistique d'Unification" (Estival, 1990). See also Johnson & Rosner (1989) for a description of the earlier UD system on which ELU is based.

[2] Thus no distinction is made between classes and 'instances', as in e.g. KL-ONE (Schmolze & Lipkis, 1983)

lexical and non-lexical classes differ in that analysis and generation take only the former as entry points to the lexicon.

Class Definition

A class definition consists of the compiler directive '`#Class`' (for a non-lexical class) or '`#Word`' (for a lexical class), followed by:

(i) the name of the class

(ii) a (possibly empty) list of its direct superclasses

(iii) a (possibly empty) 'main' equation set

(iv) zero or more 'variant' equation sets

Superclass Declaration

The superclass declaration is a list of the names of any direct superclass of the current class. This is used in computing the relative precedence of classes in the lexicon for the purpose of default inheritance (see section 2.2); it may be empty if the class has no superclasses, i.e. if it is one of the most general in the lexicon, and thus inherits no information. More specific classes appear to the left of more general ones.

Main Equation Set

Following the superclass declaration are zero or more equations or macro calls representing *default* information, which we refer to as the 'main' equation set. These may be overridden by conflicting information in a more specific class. Each equation in a main set functions as an independent constraint, in a manner which will be clarified below.

Variant Equation Sets

Following the (possibly empty) main equation set are zero or more sets of equations or macro calls representing variants within the class which, loosely speaking, correspond to alternatives at the same 'conceptual level' in the hierarchy. Equations within a variant set are absolute constraints, in contrast to those in the main set; if they conflict with information in a more specific class, failure of unification occurs in the normal way. Also, unlike the main set, each variant set functions as a single, possibly complex, constraint (see section 3). A feature structure is created for each variant set that successfully unifies with the single structure resulting from the main set. Each variant set is preceded by the vertical bar '`|`'.

String Concatenation

Construction and decomposition of complex words are carried out by the string concatenation operator '`&&`'. An equation of the form

```
X = Y && Z
```

unifies `X` nondeterministically with the result of concatenating `Y` and `Z`.

2.1 Multiple Inheritance and Ambiguity

A class may inherit from more than one direct superclass. In general, multiple inheritance of this kind necessitates more complex methods of searching a hierarchy; much of the complexity of inheritance reasoners lies in finding and determining what to do in these cases of *ambiguity*.[3] Multiple inheritance is not an *a priori* necessity for lexical specification, so it is worth considering whether any phenomena occur in this domain that might make multiple inheritance desirable, rather than the simpler tree-structured hierarchy.

However, natural language lexicons do appear to require description in terms of 'tangled hierarchies', at least if certain types of generalization are not to go unexpressed. It has often been observed, for example, that syntactic and morphological properties are in many respects disjoint; the subcategorization class of a verb cannot be predicted from its conjugation class, and vice versa. Multiple inheritance permits the two types of information to be kept separate by isolating them in distinct sub-hierarchies. This compartmentalization is implicitly related to the independence of the sub-hierarchies; if superclasses B and C of some class A are independent in this way, no conflict will arise when A inherits from B and C.

The present system disallows ambiguity of inheritance by enforcing a total ordering on the superclasses of any given class or word, and by making clear to users how this ordering is derived, so that they may more accurately control and exploit it in the organization of the hierarchy. As a substitute, the variant set mechanism is introduced; this allows variants to be represented directly within a class, rather than by creating alternate, unordered, superclasses, and corresponds to a strong element in traditional grammatical description, that such mutually exclusive variant classes should nevertheless be grouped together in a single compound statement or paradigm. A concrete version of this may be seen in the inflection tables to be found in reference and pedagogical grammars of foreign languages.

Users are able to simulate ambiguity when required, but are responsible for determining when this should occur. In effect, the ELU lexicon abandons some of the generality of an inheritance reasoner (that it reasons correctly over arbitrary inheritance networks according to certain "intuitions") by making creators of lexicons perform the "intuitive" work themselves. The creator of the lexicon is then forced to consider the desired relations, rather than relying on the semantics of the inheritance system to produce them.

2.2 Class Precedence

A system such as the ELU lexicon, which permits the defeasible inheritance of information from more than one superclass, must provide a way of resolving the conflicts that arise when information present in two or more superclasses is mutually incompatible, e.g. when the result obtained when A inherits from one superclass B before inheriting from another, C, differs from that obtained by inheriting first from C and then from B. It is in such cases that the notion of "precedence" comes into play; if the system is constrained so that information is inherited first from B, we say that B "has precedence over", or "is more specific than" C.

A familiar example of this type of situation from the AI literature is the so-called "Nixon diamond" (Touretzky, 1986). Nixon is both a Quaker and a Republican; Quakers are (typically) pacifists, while Republicans are (typically) not; the question to be answered

[3] cf. examples of cascaded ambiguity and On-Path versus Off-Path preemption in Touretzky et al. (1987)

is whether Nixon is a pacifist. This problem may be represented by the configuration shown in fig. 1. If the links to the 'Pacifist' class are both defeasible, which should take precedence, the positive link from 'Quaker', or the negative link from 'Republican'?

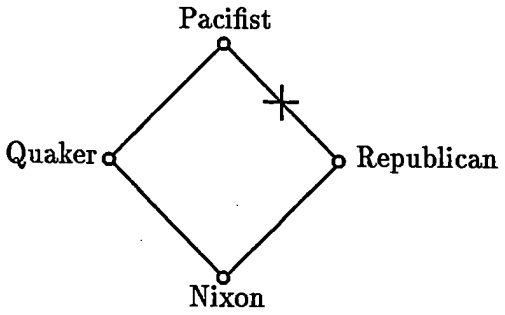


Figure 1: The Nixon Diamond

Within the ELU lexicon, a natural way of representing the same information is to dispense with a 'Pacifist' class, and instead to make (non-) pacifisthood a defeasible property of Quakers and Republicans, as shown in the example below:

```
#Word  Nixon (Quaker Republican)
    |
    <name> = 'Nixon'

#Class Quaker ()
    <pacifist> = yes
    |
    <denomination> = 'Quaker'

#Class Republican ()
    <pacifist> = no
    |
    <party> = 'Republican'
```

Here, the 'lexical class' `Nixon` has two immediate superclasses, `Quaker` and `Republican` – as we shall see below, the order in which these are declared is significant. It also contains the constraint that the value of the `<name>` path must be `Nixon`. `Quaker` imposes two constraints; the value of `<denomination>` must be `Quaker`, and the value of `<pacifist>` must be `yes`, unless that would conflict with a value assigned in some more specific class. The constraints embodied in `Republican` are that the value of `<party>` must be `Republican`, while, again unless differently assigned in a more specific class, `<pacifist>` has the value `no`.

What will be the result of looking up '`Nixon`' in this lexicon? The paths `<name>`, `<party>` and `<denomination>` are unproblematic; the only conflict arises with `<pacifist>`. As indicated above, its value will depend on which of the two superclasses of `Nixon` is the more specific; the declaration `(Quaker Republican)` states not only what the immediate superclasses of `Nixon` are, but also that `Quaker` is to be regarded as more specific than `Republican`. Thus it is `Quaker` that will provide the value for `<pacifist>`. If the opposite answer were required, the appropriate declaration would be `(Republican Quaker)`.

The ELU lexicon employs the class precedence algorithm of the Common Lisp Object System (CLOS) to derive a total order on the superclasses of each lexical class.[4] The

[4] See Steele (1990: 728ff.) for a precise statement of the algorithm, and Keene (1989: 118ff.) for discussion.

resulting 'class precedence list' (CPL) contains the lexical class itself and all of its superclasses, from most specific to most general, consistent with the local order given by class declarations. The effect of the CPL can be seen most clearly in connection with a graphical representation like that in fig. 2, which represents the partial order generated by the local immediate superclass ordering constraints in a lexicon.

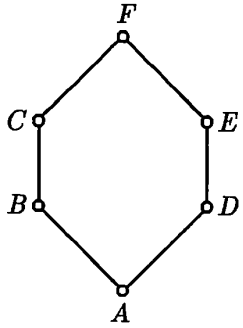


Figure 2: A partially ordered set of classes

Note that the left-to-right order of the two 'branches' in fig. 2 reflects the order in which `B` and `D` appear in the superclass declaration of `A`. The CPL is constructed by traversing the graph in a depth-first, left-to-right manner, at any joins (such as F in this case) splicing in the next leftmost path before continuing with depth-first computation. The CPL of A in fig. 2 is therefore $\langle A, B, C, D, E, F \rangle$. This procedure deterministically selects one total ordering from the set of orderings compatible with the superclass declarations; if none can be derived, the system gives an error during compilation.

Patterns of inheritance between superclasses of a lexical class are determined solely by the linear CPL. Note that this scheme excludes a number of configurations that have featured in the AI literature. Dual paths from one class to another, of the sort shown in fig. 3, cannot arise in the compiled lexicon; given a CPL $\langle c_1, \ldots c_n \rangle$, the only path from c_i to c_k is through every c_j, $0 \leq i < j < k \leq n$.

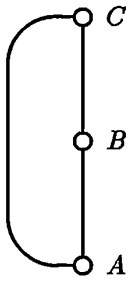


Figure 3: An impossible hierarchy

Another consequence is that cyclic hierarchies are precluded – no total order can be constructed in which $A < B$ and $B < A$. Intuitively, there is no reason for defining a network with cyclic paths when traversing the same portion of the network repeatedly can add no more information.

Nor is there any means of expressing negative links of the kind shown in fig. 1. The significance of this point is that the presence of exception links is another factor in the

complexity of a hierarchy; moreover, this type of negation is of dubious utility in the present context, for two reasons. First, the precedence of default information from subclasses enables exceptionality to be expressed without explicit negation of inheritance. The second reason is connected with the nature of unification. The absence of a path-value pair $P = < p, v >$ in a FS F cannot be interpreted as a positive constraint that (some extension of) F does not have the property represented by P; a later unification may lead to P being added to F. The desired effect can only be achieved by the presence in F of a distinct path-value pair $<p, v'>$, where v and v' do not unify. Conflicting information of this type can be introduced by means of the standard positive inheritance link.

In comparison with inheritance systems in general, then, the ELU lexicon is rather restrictive. Hierarchies are constrained to be acyclic, unipolar and unambiguous, these limitations reflecting the desire not only to reduce the complexity of the system, but also to eliminate from it inessential or redundant aspects.

2.3 An Informal Account of Lexical Access

Lookup of a lexical item with CPL $\langle c_1, \ldots c_n \rangle$ proceeds as follows: starting with an empty FS, the system first applies any default equations in c_1, then applies any variant sets in c_1 to the result. The system then repeats this process on each resulting FS, for the classes c_{i+1} to c_n in turn. The result of lookup is the set of FSs produced by the most general class c_n; this set we term the *global extension* of the lexical class c_1.

A set of default equations $D = \{d_1, \ldots d_n\}$ applies to a FS F as follows: each d_i that does not conflict with some existing information in F is unified with F, subject to the condition that any reentrant substructure of F should be compatible with D, and that any reentrant subset of D should be compatible with F.[5] A set of equations that satisfies this condition may be applied without regard to order. Any variant sets that exist in the current class are then applied to the resulting FS F'

The result of applying variant sets $v_1, \ldots v_n$ to a FS F is the set of FSs $\{f_1, \ldots f_m\}$, where each f_i is the result of successfully unifying F with some different v_j. Unification failure in variant sets produces a null result, so $m \leq n$. Variant sets have two effects: they enforce strict constraints which cannot be overridden, and multiple variant sets 'multiply' FSs, e.g. to produce different members of an inflectional paradigm.

3 The System in More Detail

Following the relatively informal presentation in the previous section, we continue by refining some of the notions introduced there.

We define the *global extension* of a lexical class in terms of two auxiliary notions, *default extension* and *superclass extension*.[6]

Default Extension

The *default extension* of a FS ϕ with respect to a set of FSs Ψ is

$$\phi \sqcup \bigsqcup \{\psi \in \Psi \mid \phi \sqcup \psi \neq \bot\}$$

[5] Bouma (1990: 166) discusses the motivation for this condition.

[6] '$A \sqcup B$' here denotes the unification of A and B, '$\top$' denotes the most general, 'empty' FS, which unifies with all others, and '$\bot$' denotes the inconsistent FS, equated with failure of unification.

if both $R(\phi) \sqcup \bigsqcup \Psi \neq \bot$ and $\phi \sqcup R(\bigsqcup \Psi) \neq \bot$, and $\bot$ otherwise, where $R(\phi)$ denotes the restriction of ϕ to reentrant paths, i.e. the most general FS such that $\forall p, q\,[\phi(p) \equiv \phi(q) \rightarrow R(\phi)(p) \equiv R(\phi)(q)]$.[7]

Each of the FSs in Ψ that can unify with ϕ does so – those that cannot, because they conflict with information already present, are ignored. This is the basis of the defaulting behaviour of the lexicon. The condition referring to reentrant paths takes account of the potential order-sensitivity of the defaulting operation – only those main sets having this property can be applied without regard to the relative order of the individual constraints within them. If the condition is met then the application of defaults always succeeds, producing a feature structure which, if no member of the set of equations is applicable, is identical to ϕ; otherwise the lookup fails.

Superclass Extension

The *superclass extension* of a FS ϕ with respect to a class C having a main equation set M and variant sets $v_1, \ldots v_n$ is

$$\Sigma(\phi, C) = \{\psi \mid 1 \leq i \leq n \wedge v_i' \sqcup \phi' = \psi \wedge \psi \neq \bot\},$$

where M' is the smallest set of FSs such that each $m \in M$ describes some $m' \in M'$, ϕ' is the default extension of ϕ with respect to M', and v_i' is the feature structure described by v_i.

$\Sigma(\phi, C)$ is formed by applying to ϕ any default equations in the main set of C, and then applying to the result each variant set in C; for variant sets $v_1, \ldots v_n$, the result of this second stage is the set of FSs $\{\psi_1, \ldots \psi_m\}$, where each ψ_i is the result of successfully unifying ϕ with some different v_j.

Global Extension

The *global extension* of a lexical class L having the CPL $C = \langle c_1, \ldots c_n \rangle$ is Γ_n, where $\Gamma_0 = \{\top\}$, and

$$\Gamma_{i>0} = \bigcup \{\Psi \mid \forall \phi \in \Gamma_{i-1}, \Psi = \Sigma(\phi, c_i)\}.$$

To speak in procedural terms, $\top$ is the empty FS which is input to C; each c_i in C yields as its superclass extension a set of FSs, each member of which is input to the remainder of C, $\langle c_{i+1}, \ldots c_n \rangle$. The global extension of L is then the yield of the most general class in its CPL – expressed in a slightly different way, the global extension of L is the result of *applying* to $\top$ the CPL of L. The set of lexical items admitted by a lexicon consists of the union of the global extensions of all lexical classes in the lexicon.

The implementation of defaults in ELU ensures the monotonicity of structure-building – given a CPL $\langle c_1, \ldots c_n \rangle$, any FS F admitted by a class c_i subsumes every FS that can be created by applying to F the classes $\langle c_{i+1}, \ldots c_n \rangle$. Shieber (1986:59ff) and Karttunen (1986:76) describe default inheritance systems based on a nonmonotonic 'overwriting' operation; in both cases, the statements exhibiting default behaviour are ones that have the ability to override others over which they have precedence, whereas in the approach presented here the default statements are ones that can be overridden by others which have precedence over them.

[7] Here, '$\phi(p)$' denotes the value of the path p in the FS ϕ, and '$\equiv$' denotes token-identity of its operands.

4 An Example Analysis

This section briefly illustrates some aspects of the ELU lexicon introduced above with an analysis of English verbal morphology.

In most cases, lexical items that realize certain morphosyntactic properties in irregular forms do not also have regular realizations of those properties; thus **sinked*, on the analogy of e.g. *walked*, is not a well-formed alternative to *sank* or *sunk*. This phenomenon has frequently been discussed in both theoretical and computational morphology, under the title of 'blocking', and it appears to provide clear motivation for a default-based hierarchical approach to lexical organization. There are exceptions to this general rule, however, and inheritance mechanisms must be sufficiently flexible to permit deviation from the strict pattern.

Consider the small class of English verbs including *dream*, *lean*, *learn* and *burn*; for many speakers, these have alternate past finite and past participle forms: e.g. *dreamed* and *dreamt*. The following simplified fragment produces the correct analyses, in which the value of `<morph>` expresses inflectional information, and that of `<form>` is the corresponding word-form:

```
#NLexicon Eng-Irreg-Verbs

#Word walk (Verb)
      <stem> = walk

#Word sink (Verb)
      <stem> = sink
      P_Fin_Form = sank                    PSP_Form = sunk

#Word dream (DualPast Verb)
      <stem> = dream

#Class DualPast ()
      |
      PSP_Form = <stem> && t
      P_Fin_Form = <stem> && t
      <morph> = pastfinite/pastnonfinite
      |

#Class Verb (VFin VNonFin)
      <cat> = v

#Class VFin ()
      P_Fin_Form = <stem> && ed
      |
      <morph> = present_nonsg3          <form> = <stem>
      |
      <morph> = present_sg3             <form> = <stem> && s
      |
      <morph> = pastfinite              <form> = P_Fin_Form
      |
      <morph> = pastnonfinite

#Class VNonFin ()
      PSP_Form = <stem> && ed
      |
      <morph> = pastnonfinite           <form> = PSP_Form
      |
      <morph> = present_nonsg3/present_sg3/pastfinite
```

Being the lexical class of a regular verb, `walk` contains a minimum of idiosyncratic information, *viz.* that the value of the feature `<stem>` is the string *walk*. Its direct superclass, `Verb`,

contributes the information that the value of <cat> is v, and in turn specifies inheritance from its direct superclasses, VFin and VNonFin. Each of these contains a single main set equation, assigning a default value to a variable; both P_Fin_Form and PSP_Form, having no conflicting value, are set to the concatenation of *walk* and *ed*. The first three variant sets of VFin establish correspondences between values of <morph> and <form>, while the third unifies with variants treated by VNonFin. The latter class contains just two variant sets, the first of which produces the correct form for past nonfinite verbs, while the second permits analysis of the finite forms treated by VFin – it contains a disjunctive constraint, to the effect that the value of <morph> must unify with at least one of present_nonsg3, present_sg3, and pastfinite.

The main set equations in sink assigning values to P_Fin_Form and PSP_Form override those in its superclasses VFin and VNonFin, so that the variants in the latter class which give rise to past participle and past tensed forms associate the appropriate information with the strings *sunk* and *sank*, respectively.

The lexical class dream is exceptional in having as one of its direct superclasses DualPast, which contains two variant sets, the second of which is empty (recall that variant sets are preceded by the vertical bar '|'). Moreover, this class is more specific than the other superclass Verb, and so the equations in the first variant set assigning to PSP_Form and P_Fin_Form the string formed by concatenating the value of <stem> and *t* (e.g. *dreamt*) have precedence over the contradictory statements in the main sets of VFin and VNonFin. The absence of contradictory specifications in the second variant set permits the equations in the main sets of VFin and VNonFin to apply. In addition to specifying exceptional properties, therefore, the definition of DualPast also permits the inheritance of properties from more general classes, i.e. those of regular verbs like *walk*; among these is that of forming the two past forms by suffixing *ed* to the stem, which produces the regular (*dreamed*, etc.) past forms.

5 Summary

The popularity of unification as a tool for computational linguistics stems from its declarative, monotonic semantics; however, the price to be paid for the benefits of a pure unification framework is the lack of a satisfactory treatment of exceptions (negation, defaults, etc.). The popularity of default inheritance as a tool for knowledge representation stems from its ability to encode, in a straightforward manner, the type of nested generalization with exceptions that natural language lexicons exhibit; however, in achieving this expressive power one introduces a degree of order-dependence into the system. The approach presented here attempts to combine the advantages of unification and default inheritance, while minimizing the disadvantages arising from their interaction.

Properties of general default systems that lead to intractability are absent; the total ordering imposed on superclasses by the CPL eliminates cycles, ambiguity and the redundancy of multiple paths, while the suppression of negative inheritance links removes a further source of complexity. Facilities have been dispensed with not only because they are computationally problematic, but also as a result of the application in question – as we observe in sections 2.1 and 2.2, ambiguity and negation are redundant in the present context.

References

Bouma, G. (1990) "Defaults in Unification Grammar", *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, Pittsburgh, June 6th–9th 1990. 165–172.

Estival, D. (1990) *ELU User Manual.* Technical Report 1, ISSCO, Geneva.

Johnson, R. and M. Rosner (1989) "A Rich Environment for Experimentation with Unification Grammars", *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, Manchester, April 10th–12th 1989. 182–189.

Karttunen, L. (1986) "D-PATR: A Development Environment for Unification-Based Grammars", *Proceedings of the 11th International Conference on Computational Linguistics*, Bonn, August 25th–29th 1989. 74–80.

Keene, S. (1989) *Object-Oriented Programming in Common Lisp.* Addison-Wesley, Reading, Massachussetts.

Schmolze, J. G. and T. A. Lipkis (1983) "Classification in the KL-ONE Knowledge Representation System", *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, August 8th–12th 1983. 330–332.

Selman, B. and H. J. Levesque (1989) "The Tractability of Path-Based Inheritance", *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, August 20th–25th 1989. 1140–1145.

Shieber, S. M. (1986) *An Introduction to Unification-Based Approaches to Grammar*, CSLI, Stanford.

Steele, G. L. (1990) *Common Lisp: The Language* (second edition), Digital Press, Bedford, Massachussetts.

Touretzky, D.S. (1986) *The Mathematics of Inheritance Systems*, Pitman Publishing, London.

Touretzky, D. S., J. F. Horty and R. M. Thomason (1987) "A Clash of Intuitions: The Current State of Nonmonotonic Multiple Inheritance Systems", *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, August 23rd–27th 1987. 476–482.

The ACQUILEX LKB: a system for representing lexical information extracted from machine-readable dictionaries

Ann Copestake Valeria de Paiva Antonio Sanfilippo

Introduction

The following papers describe the LKB, a lexical knowledge base system which has been designed as part of the ACQUILEX project[1] to allow the representation of syntactic and semantic information extracted from machine readable dictionaries (MRDs) on a large scale. It uses a typed unification-based representation language which incorporates default inheritance. The LKB's knowledge representation language (LRL) can be viewed as an augmentation of a typed graph-based unification formalism (Carpenter 1990) with minimal default inheritance; default inheritance is formalised in terms of default unification of feature structures.

Although there has been previous work on building lexicons for NLP systems from MRDs (eg Carroll and Grover 1989), most attempts at extracting semantic information have not made use of a formally defined representation language; typically a semantic network or a frame representation has been suggested, but the interpretation and functionality of the links has been left vague. Several networks based on taxonomies have been built, and these are useful for tasks such as sense-disambiguation, but are not directly utilisable as NLP lexicons. For a reusable lexicon, a declarative, formally specified, representation language is essential. A large lexicon has to be highly structured; it is necessary to be able to group lexical entries and to represent relationships between them. But, unless these notions of structure are properly specified, a lexicon based on them will be impenetrable except (perhaps) to its creators. We therefore take the idea of semantic structuring seriously, and use taxonomic information as one of the ways of providing such structure, but we do this within the context of a formally specified representation language.

We chose to use a graph unification based representation language for the LKB, because this offered the flexibility to represent syntactic and semantic information, and the interaction between them, in a way which could be easily integrated with much current work on unification grammar, parsing and generation. In contrast to DATR (Evans and Gazdar 1990) for example, the LRL is not specific to lexical representation. This made it much easier to incorporate a parser in the LKB (which is almost essential for developing a type system and for testing lexical entries) and to experiment with notions such as lexical rules and inter-lingual links between lexical entries. Although this means that the LRL is

[1] 'The Acquisition of lexical knowledge for Natural Language Processing systems' (Esprit BRA-3030) concerned with the extraction of information from machine readable dictionaries (MRDs)

perhaps too general for its main application, the type system provides a way of flexibly constraining the representation.

The main structure is provided by the type system. Our formalisation of typed feature structures is based on Carpenter(1990, 1991) although there are some significant differences. The type system can be regarded as a way of providing (non-default) inheritance, combined with error-checking. The notion of types, and features appropriate for a given type, gives some of the functionality of frame representation languages, such as KL-ONE; in particular, classification of a feature structure is possible.

We augment the formalism with a default inheritance mechanism. This can be used to organise the lexicon in a completely user-defined way, to allow morphological or syntactic information to be concisely specified, for example, as has been done with DATR and other systems (for example, Russell et al (1991), Krieger and Nerbonne (1991)). However much of the motivation behind our formalisation of default inheritance comes from consideration of the sense-disambiguated taxonomies semi-automatically derived from MRDs, which we are using to structure the LKB (see Copestake 1990). The top level of the inheritance structure, which cannot be automatically derived from MRDs, is, in effect, given by the type system.

Thus the operations that the LRL supports are (default) inheritance, (default) unification and lexical rule application. It does not support any more general forms of inference and is thus designed specifically to support processes which concern lexical rather than general reasoning. The type system provides the non-default inheritance mechanism and constrains default inheritance. We use lexical rules as a further means of structuring the lexicon, in a flexible, user definable manner, but lexical rules are also constrained by the type system. The LKB's lexical rule mechanism has been described in Copestake & Briscoe(1991) and Briscoe & Copestake(1991) and is not further discussed here.

The first paper, "Types and Constraints in the LKB", discusses the theoretical background to typed feature structures and the way that they are formalised in the LRL. In "LKB Encoding of Lexical Knowledge from Machine-Readable Dictionaries" the verb type system is described and its application to the acquisition of psychological predicates from MRDs is discussed. "Defaults in the LRL" sketches the default unification and default inheritance mechanisms and the way they interact with the type system, and the final paper "Using the LKB" describes the implemented system and the utility of typing for our application.

Types and Constraints in the LKB

Valeria de Paiva
Computer Laboratory, University of Cambridge

Introduction

This paper describes - from the mathematical perspective - the system of typed feature structures used in the ACQUILEX Lexical Knowledge Base (LKB). For linguistic (mainly lexical) motivation and implementation details one should look at the introduction to *The LKB: a System for Representing Lexical Information Extracted from Machine-Readable Dictionaries* by Copestake. For a full working example of how the system may be used, refer to *LKB Encoding of Lexical Knowledge from Machine-Readable Dictionaries* by Sanfilippo (this volume).

In this note we concentrate on describing the type system the LKB takes as input, making explicit the necessary conditions on the type hierarchy and explaining how - mathematically - our system of constraints works. It is assumed that the reader is familiar with basic unification-based formalisms like PATR-II, as (very well) explained in Shieber (1986). It must also be said from the start that our approach is, basically, a modification of Carpenter's *typed quasi-feature structures*, as developed in the collection of papers from the Leuven Summer School, Carpenter (1990) and in the book (in preparation) *The Logic of Typed Feature Structures.*

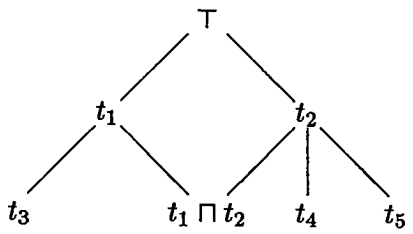
The LKB works basically through unification on (typed) feature structures. Since most of the time we deal with *typed* feature structures - see precise mathematical definition in section 2 - we will normally drop the qualifier and talk about feature structures. When necessary, to make a distinction, we call PATR-II *untyped* feature structures. Feature structures are defined over a (fixed) *finite* set of features FEAT and over a (fixed) type hierarchy $\langle \mathsf{TYPE}, \sqsubseteq \rangle$. Given FEAT and $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ we can define $\mathcal{F}$ the collection of all feature structures over FEAT and $\langle \mathsf{TYPE}, \sqsubseteq \rangle$. But we are interested in feature structures which are *well-formed* with respect to a set of constraints. To describe constraints and well-formedness of feature structures we specify a function $C: \langle \mathsf{TYPE}, \sqsubseteq \rangle \rightarrow \mathcal{F}$, which corresponds to an association of a constraint feature structure $C(t_i)$ to each type t_i in the type hierarchy TYPE. The constraint feature structure $C(t_i)$ imposes conditions or constraints on all well-formed feature structures of type t_i. We call the combination of FEAT, $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ and the constraint function C the *type system.*

It should be clear from the previous work in unification-based formalisms that modifications to the type system account for a whole gamut of linguistic theories.

We will initially define the type hierarchies $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ we deal with and formalise our notion of feature structures and some operations over them. Next we describe our kind of constraints and what it means for a feature structure to be well-formed in our system. Then we discuss briefly internal and external logics of feature structures. A short section concludes comparing this with related work, especially Carpenter's.

1 The Type Hierarchy

We will adopt the (HPSG) notation with the most general type at the top of any diagram. The type hierarchy is ordered by $\sqsubseteq$ (which can be read "is more specific than"). For example:



We will say that the type hierarchy is a partially ordered set (or poset) $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ with two extra properties. Before describing these properties we recall that if $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ is a poset, it satisfies:

- (reflexivity) $t \sqsubseteq t$ for any t in $\langle \mathsf{TYPE}, \sqsubseteq \rangle$.
- (anti-symmetry) If $t \sqsubseteq s$ and $s \sqsubseteq t$ in $\langle \mathsf{TYPE}, \sqsubseteq \rangle$, then $s = t$.
- (transitivity) If $t_1 \sqsubseteq t_2$ and $t_2 \sqsubseteq t_3$ then $t_1 \sqsubseteq t_3$.

It is also an easy consequence of the definition of a poset that the order '$\sqsubseteq$' has *no cycles*, i.e if $t_1 \sqsubset t_2$ then $t_2 \not\sqsubseteq t_1$ - where we write $t_1 \sqsubset t_2$ for $t_1 \sqsubseteq t_2$ and $t_1 \neq t_2$. (By RA say $t_2 \sqsubseteq t_1$, then using $t_1 \sqsubseteq t_2$ and anti-symmetry we have $t_1 = t_2$, contradiction!)

Following Carpenter we call a subset $S \subseteq \mathsf{TYPE}$ *consistent*[1] iff there is some t_0 in TYPE such that $t_0 \sqsubseteq t$ for any t in S. In the example above for instance the sets $\{t_1, t_2\}$ and $\{t_2, t_4\}$ are consistent sets, but $\{t_4, t_5\}$ is not, so the first two sets have meets, respectively $t_1 \sqcap t_2$ and t_4, while the third set has not. Then we can define:

Definition 1 *The type hierarchy* $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ *is a (non-empty) poset with two extra properties:*

1. *Every* consistent *set of types* $S \subseteq \mathsf{TYPE}$ *has a unique greatest lower bound or meet (notation* $\sqcap S$*).*
2. *The partial order* $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ *has no unary branches, ie no type may have exactly one immediate subtype. If* $t_2 \sqsubset t_1$ *and there is no intermediate type* s *such that* $t_2 \sqsubset s$ *and* $s \sqsubset t_1$ *then there must be some other subtype* t_3 *such that* $t_3 \sqsubset t_1$ *and* $t_3 \not\sqsubseteq t_2$*.*

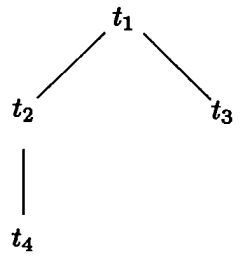
Note that the empty set $\emptyset$ is (vacuously) consistent, as if there is any t_0 in TYPE it satisfies the condition that $t_0 \sqsubseteq t$ for all t's in the empty set. Hence the partial order $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ must have a maximal element $\top$ which is the meet of the consistent set $\emptyset$, $\top = \sqcap\emptyset$. This element $\top$ is such that $t \sqsubseteq \top$ for any t in TYPE. Thus the first property says that the type hierarchy $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ is (the dual of) a bounded complete poset, cf. definition in Gunther and Scott (1991).

[1]The usual term in Lattice Theory is bounded, but consistent seems more expressive.

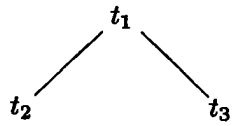
The first property could be re-stated as saying that $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ is a "consistently complete meet-semilattice". But beware as a consistently complete meet-semilattice is *not* a meet-semilattice, since it does not have all binary meets, only the *consistent* ones.

If the bounded complete poset $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ is finite then all (non-empty) joins are defined. Thus we have a poset $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ with two operations, a *partial* operation of taking binary meets $\sqcap$ - or greatest lower bounds - and a total operation of taking joins $\sqcup$ - or lowest upper bounds.

The geometric meaning of the second property of $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ is that posets like

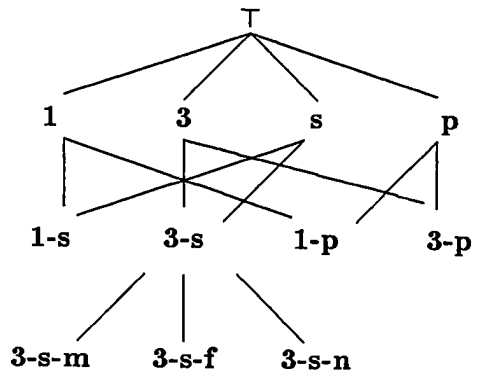


are not allowed. The no-unary-branching condition is desirable because the type system is "intuitively complete", where by complete we mean that whatever is said in the partial-order is all we know about the types being described. Hence if we say



the interpretation we have in mind is that t_2 things are t_1 and t_3 things are t_1 and things which are t_1 are either t_2 or t_3 but nothing else. Thus if we did have the situation above where t_4 is the only subtype of t_2 we would be stating that everything which was of type t_2 was also of type t_4 (as well as the inverse). To specify both in the hierarchy could lead to inconsistency (with respect to the specification of constraints for example) so unary branches are disallowed.

We can make the meet $\sqcap$ operation total if we add the join of the empty-set $\bot = \sqcup\emptyset$ to $\langle \mathsf{TYPE}, \sqsubseteq \rangle$. But even if we do add $\bot$ to make $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ a lattice, this lattice need not be distributive, not even modular, as the example below from Carpenter (1990) shows



Adding $\bot$ to the poset above, we have:

$$(\mathbf{3-s-m} \sqcup \mathbf{3-s-f}) \sqcap \mathbf{3-s-n} = \mathbf{3-s-n}$$

$$\neq (\mathbf{3-s-m} \sqcap \mathbf{3-s-n}) \sqcup (\mathbf{3-s-f} \sqcap \mathbf{3-s-n}) = \bot$$

Some implementations of systems similar to ours assume a lattice of types and a lattice of feature structures. This can always be achieved by a process of completion of the partial order and several different completion processes are possible, see for instance Davey and Priestley (1990). If we do add *only* $\bot$ to $\langle \mathsf{TYPE}, \sqsubseteq \rangle$, we call the resulting type hierarchy $\langle \mathsf{TYPE}, \sqsubseteq \rangle_\bot$. In this case we have an inclusion,

$$\langle \mathsf{TYPE}, \sqsubseteq \rangle \xrightarrow{incl} \langle \mathsf{TYPE}, \sqsubseteq \rangle_\bot$$

Condition 1 on the definition of the type hierarchy $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ seems necessary for the constructions we want to make, at least if one insists on a unique value for the unification of feature structures. Condition 2 on the other hand is interesting, but not necessary. In his most recent work Carpenter drops this condition.

2 Feature Structures

In this section we define formally the feature structures we shall be dealing with and compare our definition with the traditional (untyped) PATR-II style one, as in Moshier and Rounds (1987). We define the collection $\mathcal{F}$ of feature structures over the (fixed) set of features FEAT and the (fixed) type hierarchy $\langle \mathsf{TYPE}, \sqsubseteq \rangle$. Our feature structures are an acyclic variant of Carpenter's (1990) (typed) *quasi-feature structures.*

Definition 2 *A* feature structure *is a tuple* $F = \langle Q, q_0, \delta, \alpha \rangle$ *where*

- Q *is a (non-empty) finite set of (connected, acyclic) nodes;*
- $q_0 \in Q$ *is the initial (or root) node;*
- $\alpha: Q \to \langle \mathsf{TYPE}, \sqsubseteq \rangle$ *is a* total *node typing function and* $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ *is a type hierarchy as in the previous section;*
- $\delta: \mathsf{FEAT} \times Q \to Q$ *is a partial transition function, where* FEAT *is a (non-empty) finite set.*

The collection of all possible feature structures for a given set FEAT *and poset* $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ *is denoted* $\mathcal{F}$.

An example of a feature structure is:

$$F_1 = \begin{bmatrix} \mathbf{phrase} & \\ \text{AGR} = & \begin{bmatrix} \mathbf{agr} & \\ \text{PERS} = & \mathbf{1} \\ \text{NUM} = & \mathbf{sing} \end{bmatrix} \end{bmatrix}$$

A notational convention is that types are written in **boldface** and features are written in SMALL CAPITALS within attribute-value matrices (with the exception of the type $\top$). In mathematical definitions t's are used as variables for types, f's as variables for features and F's as variables for feature structures.

The intuition behind this definition goes back to Kasper and Rounds formalisation of the logic of feature structures. The main idea being that an attribute-value matrix like

$$\left[\text{AGR} = \left[\begin{array}{l}\text{PERS} = \mathbf{3}\\ \text{NUM} = \mathbf{sing}\end{array}\right]\right]$$

could be thought of as a deterministic automaton, Kasper and Rounds (1986). By a 'connected set of nodes' we mean that every node $q \in Q$ is reachable from the initial node q_0 by δ. More precisely, there exists a sequence of features $\langle f_1 \dots f_{n-1}\rangle$ in FEAT^* and a sequence of nodes $\langle q_0, q_1, \dots q_n\rangle$ such that $\delta(q_i, f_{i+1}) = q_{i+1}$ and $q_n = q$.

Recall that in the traditional definition of a feature structure as in, for instance, Carpenter's paper in this volume (after Moshier and Rounds (1987)), one has a *partial (injective) atomic value* function α from nodes to atoms. But only nodes for which no features are defined by the transition function can have atomic values, so that if $\alpha(q)$ is defined then $\delta(f, q)$ is undefined for all $f \in \mathsf{FEAT}$. Some types in the definition above will correspond to the "nodes that do not have features" in the traditional definition and we shall call them *atomic* types. For instance **sing** and **1** in the example above are atomic types.

The main differences between the traditional definition and the one above are that:

- In our definition *all* nodes, not only *some* of the terminal ones, have types.
- The set of types TYPE is now endowed with a partial order.

People of a very abstract turn of mind could write Moshier-Rounds definition as a triple of functions,

$$1 \xrightarrow{q_0} Q \times \mathsf{FEAT} \xrightarrow{\delta} Q \xrightarrow{\alpha} \mathsf{ATOMS}$$

where a map $1 \xrightarrow{q_0} Q$ picks up one object, q_0, in the set Q; the arrows $\rightharpoonup$ for δ and α are partial maps; α is injective and the domain of definition of α is given by

$$dom(\alpha) = \{q \in Q \mid \delta(q, f) \text{ is undefined } \forall f \in \mathsf{FEAT}\}$$

They could also write our definition as

$$1 \xrightarrow{q_0} Q \times \mathsf{FEAT} \xrightarrow{\delta} Q \xrightarrow{\alpha} \langle \mathsf{TYPE}, \sqsubseteq\rangle$$

where, in contrast, the function α is total and TYPE is endowed with a partial order. Pollard and Moshier's *ordinary feature structures* (1991) are slightly different in that the function α is partial, non-terminal nodes can have *SORTS* (and *SORTS* may have a partial ordering on them) and the acyclicity condition is dropped. In all cases one should remember that the set Q is 'rooted' (or connected) by the transition function.

One of the immediate consequences of our definition is that, as every feature structure has a unique initial node q_0, every feature structure has a type. We say that

Definition 3 *The* type of the feature structure $F = \langle Q, q_0, \delta, \alpha\rangle$ *is the type of its initial node, that is* $\alpha(q_0)$.

Note that this definition induces a function *type-of*: $\mathcal{F} \to$ TYPE. For example the type of feature structure F_1 in the example above is **phrase.**

Corresponding to the distinction between atomic and non-atomic types we have atomic and non-atomic feature structures. The feature structure F_1 in the example above is a non-atomic feature structure, whereas the feature structure consisting of the single type [**sing**] is an atomic one.

One similarity between the definitions above is that they can be extended to "paths" π in FEAT*. That is, every feature structure F over FEAT gives rise to a map

$$Q \times \mathsf{FEAT}^* \xrightarrow{\delta^*} Q$$

where

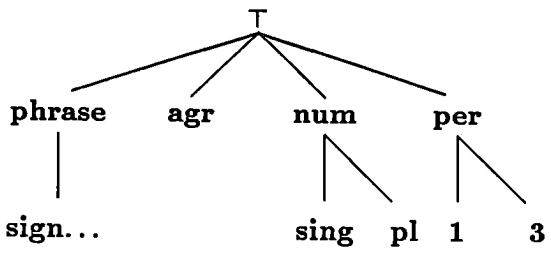
- $\delta^*(q, \lambda) = q$ if λ is the empty path,
- $\delta^*(q, \pi.f) = \delta^*(\delta^*(q, \pi), f)$.

3 Subsumption of Feature Structures

In this section we describe the order on the collection $\mathcal{F}$ of feature structures and describe the operation of restricting a feature structure F to a node q or a path π over FEAT.

It is clear that, as in the traditional setting, we have a natural order in the collection of feature structures $\mathcal{F}$. We call this order $\sqsubseteq$, overloading the symbol. Intuitively $\sqsubseteq$ means is subsumed by ("is-more-specific-than"). Subsumption is like usual subsumption of feature structures with the added condition that the order on types is 'preserved' (see precise definition below).

For example, given the following type hierarchy:



Then we have:

$$\begin{bmatrix} \textbf{phrase} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{sing} \end{bmatrix} \end{bmatrix} \sqsubseteq \begin{bmatrix} \textbf{sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \top \end{bmatrix} \end{bmatrix}$$

Intuitively, if F_1 and F_2 are feature structures of types t_1 and t_2 respectively, then $F_1 \sqsubseteq F_2$ only if $t_1 \sqsubseteq t_2$. This order in the collection of feature structures $\mathcal{F}$ is mathematically expressed using feature structure morphisms, following Moshier and Rounds.

Definition 4 *Given feature structures F_1 and F_2, $\langle Q_1, q_0, \delta_1, \alpha_1 \rangle$ and $\langle Q_2, q_0', \delta_2, \alpha_2 \rangle$, respectively, in $\mathcal{F}$ we say a total map $h: Q_1 \to Q_2$ is a* feature structure morphism *iff*

- *h sends the initial node q_0 to the initial node q_0', that is, $h(q_0) = q_0'$.*
- *h preserves the partial map structure of F_1, that is the following diagram "commutes",*

$$\begin{array}{ccc} Q_1 \times \mathsf{FEAT} & \xrightarrow{\delta_1} & Q_1 \\ {\scriptstyle h \times \mathsf{FEAT}}\downarrow & & \downarrow{\scriptstyle h} \\ Q_2 \times \mathsf{FEAT} & \xrightarrow[\delta_2]{} & Q_2 \end{array}$$

  By "commutes" we mean that if $\delta_1(q, f)$ is defined, which we write as "$\delta_1(q, f) \downarrow$" then $\delta_2(h(q), f) \downarrow$ and $h(\delta_1(q, f)) = \delta_2(h(q), f)$.

- *h 'preserves' the order in $\langle \mathsf{TYPE}, \sqsubseteq \rangle$, that is $\alpha_2(h(q)) \sqsubseteq \alpha_1(q)$.*

For feature structures F_1 and F_2 in $\mathcal{F}$, we say $F_1 \sqsubseteq F_2$ iff there is a feature structure morphism *$h: F_2 \to F_1$.*

This notion of morphism is a natural extension of the definition of homomorphism in Moshier and Rounds for untyped feature structures. The main difference is that for untyped feature structures, if $\alpha_1(q)$ is defined, then $\alpha_2(h(q))$ is defined and *equal* (rather than less or equal) to $\alpha_1(q)$.

Note the 'opposite' of the Carpenter (1990) (or Pollard and Moshier (1991)) order in the definition above. With our definition of feature structures, the least informative feature structure is $[\top]$, that is $F \sqsubseteq [\top]$ for any F in $\mathcal{F}$. But note as well that the subsumption order is not simply a containment order. For example in the feature structures below, $F_1 \supseteq F_2$, but $F_1 \not\sqsubseteq F_2$.

$$F_1 = \left[\begin{array}{l} \textbf{sign} \\ \textsc{agr} = \left[\begin{array}{l} \textbf{agr} \\ \textsc{pers} = \textbf{1} \\ \textsc{num} = \textbf{sing} \end{array}\right] \end{array}\right] \supseteq F_2 = \left[\begin{array}{l} \textbf{agr} \\ \textsc{pers} = \textbf{1} \end{array}\right]$$

Looking at the definition of morphism of feature structures abstractly we have:

$$\begin{array}{ccccccc} 1 & \longrightarrow & Q_1 \times \mathsf{FEAT} & \xrightarrow{\delta_1} & Q_1 & \xrightarrow{\alpha_1} & \mathsf{TYPE} \\ \| & & {\scriptstyle h \times \mathsf{FEAT}}\downarrow & & {\scriptstyle h}\downarrow & \sqsubseteq & \| \\ 1 & \longrightarrow & Q_2 \times \mathsf{FEAT} & \xrightarrow[\delta_2]{} & Q_2 & \xrightarrow[\alpha_2]{} & \mathsf{TYPE} \end{array}$$

First note that the order in $\mathcal{F}$ is *not* a partial order, but only a *pre-order*. We can have $F_1 \sqsubseteq F_2$ and $F_2 \sqsubseteq F_1$ without F_1 and F_2 being the same, in this case they are called alphabetic variants, which we write as $F_1 \sim F_2$, following Carpenter. We can make this pre-order a poset by taking equivalence relations the usual way. The equivalence classes of feature structures are called by Moshier (1988) *abstract feature structures.*

We need some extra trivial definitions. Given a feature structure $F = \langle Q, q_0, \delta, \alpha \rangle$ and a node q in Q we can define $F|_q$ the restriction of F to q, as the feature structure that starts in q and is the restriction of F - as a partial map. More formally:

Definition 5 *Given a feature structure F of the form $\langle Q, q_0, \delta, \alpha \rangle$ and a node q in Q we define $F|_q$ the restriction of F to q, as the feature structure $F' = \langle Q', q_0', \delta', \alpha' \rangle$, such that:*

1. *The new initial node q_0' is q,*
2. *The set of nodes Q' is the subset of the nodes of Q reachable from q, ie*

$$Q' = \{q \in Q |\ \delta(\pi, q) \text{ is defined, for all } \pi \in \mathsf{FEAT}^*\}$$

3. *The transition function $\delta'(f, q)$ is the restriction of δ to Q'.*
4. *The typing function α' is the restriction of α to Q'.*

We can also define the restriction $F@\pi$ of a feature structure F to a path $\pi \in \mathsf{FEAT}^*$. The definition above would only change in the two first clauses; the new initial node is $q_0' = \delta(\pi, q_0)$ and the new set of nodes is the subset of Q reachable from q_0'. Viewed this way each feature f or path π determines a partial function from feature structures to feature structures, the basis of other formalisations of feature structure logics cf. Smolka (1988).

Another definition extracts features from a node.

Definition 6 *Given a feature structure F and a node q in F, we define* features of the node q in F *or $Feat(\langle F, q \rangle)$, as the set of features labelling the edges coming out of the node q. Thus if F is given by $\langle Q, q_0, \delta, \alpha \rangle$ and $\delta(f, q)$ is defined then the feature f is in $Feat(\langle F, q \rangle)$ or*

$$Feat(\langle F, q \rangle) = \{f \in \mathsf{FEAT} \mid \delta(f, q) \text{ is defined}\}$$

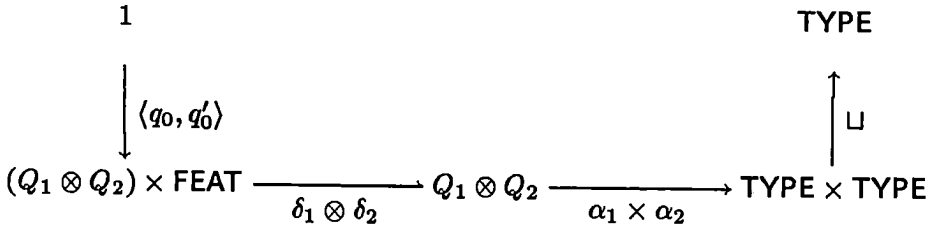
We call $Feat_0(F)$ the set of features that appear on the top level of the feature structure F, that is $Feat_0(F) = Feat(\langle F, q_0 \rangle)$.

It is reasonable to ask *why* do we want the morphisms in $\mathcal{F}$ in the direction of definition 4, which is not the direction chosen by Carpenter or Pollard and Moshier. The same question could be asked about the direction of the order on TYPE. The answer is given by the use of feature structures. The main operation one wants to perform with feature structures is *unification*, which we describe in the next section. Unification of F_1 and F_2 is the *conjunction* of the information contained in F_1 and F_2. Taking the order on TYPE and the morphisms as defined here, unification corresponds to 'meet' in the pre-order $\mathcal{F}$, the natural choice for logical conjunction.

4 Operations on Feature Structures

We want to define two main operations on feature structures, *generalisation* and *unification.* We give algebraic definitions of both unification and generalisation, following Carpenter (1990), but since algebraically *generalisation* is easier, we start with this operation.

The operation of generalisation $\sqcup\colon \mathcal{F} \times \mathcal{F} \to \mathcal{F}$ is much more natural from the algebraic viewpoint than the more useful unification. Given feature structures F_1 and F_2, respectively, $\langle Q_1, q_0, \delta_1, \alpha_1 \rangle$ and $\langle Q_2, q_0', \delta_2, \alpha_2 \rangle$ in $\mathcal{F}$, we can take the product $\delta_1 \times \delta_2$ of the partial maps δ_1 and δ_2 and transform it in a 'product of automata' as follows:



To be precise:

Definition 7 *Given feature structures F_1 and F_2 their generalisation $F_1 \sqcup F_2$ is the feature structure*

$$F_1 \sqcup F_2 = \langle Q_1 \otimes Q_2, \langle q_0, q_0' \rangle, \delta_1 \otimes \delta_2, \alpha_1 \otimes \alpha_2 \rangle$$

given by:

- *The new initial node is the pair $\langle q_0, q_0' \rangle$.*
- *The new transition function $\delta_1 \otimes \delta_2$ is the subset of the product function $\delta_1 \times \delta_2$ given by the composition:*

  $$Q_1 \times Q_2 \times \mathsf{FEAT} \xrightarrow{\Delta} Q_1 \times \mathsf{FEAT} \times Q_2 \times \mathsf{FEAT} \xrightarrow{\delta_1 \times \delta_2} Q_1 \times Q_2$$

  Thus $\delta_1 \otimes \delta_2$ is given by restricting $\delta_1 \times \delta_2$ to the pairs $\langle \delta_1(q_1, f), \delta_2(q_2, f') \rangle$ where the feature 'read' is the same, i.e $f = f'$.
- *The set of nodes $Q_1 \otimes Q_2$, is the subset of the product of nodes $Q_1 \times Q_2$ rooted by the transition function $\delta_1 \otimes \delta_2$ above. Thus (q_1, q_2) is in $Q_1 \otimes Q_2$, if there exists a path $\pi \in \mathsf{FEAT}^*$ such that*

  $$\delta_1 \otimes \delta_2(\langle q_0, q_0' \rangle, \pi) = (q_1, q_2)$$

- *The new typing function $\alpha_1 \otimes \alpha_2$ is given by first composing $\alpha_1 \times \alpha_2$ with the function generalisation on types $\sqcup\colon \mathsf{TYPE} \times \mathsf{TYPE} \to \mathsf{TYPE}$; and then restricting the result to the nodes in $Q_1 \otimes Q_2$.*

An easy example should help to make things clear. Suppose we have F_1 and F_2 below and we know **phrase** $\sqsubseteq$ **sign**,

$$F_1 = \begin{bmatrix} \textbf{phrase} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{sing} \end{bmatrix} \end{bmatrix} \quad F_2 = \begin{bmatrix} \textbf{sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{pl} \end{bmatrix} \end{bmatrix}$$

Generalising we end up with $F_1 \sqcup F_2$ given by:

$$F_1 \sqcup F_2 = \begin{bmatrix} \textbf{sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{num} \end{bmatrix} \end{bmatrix}$$

Thus generalisation corresponds to taking the product of the partial maps restricting to the diagonal in FEAT and making the resulting structure 'rooted', ie. getting rid of the unreachable nodes. In the example above we have 16 nodes in $Q_1 \times Q_2$, but 12 are isolated, thus only 4 appear in $Q_1 \otimes Q_2$.

Note that $F_1 \sqcup F_2$ is the *lowest upper bound* of F_1 and F_2 in the subsumption order. That is $F_1 \sqsubseteq F_1 \sqcup F_2$ and $F_2 \sqsubseteq F_1 \sqcup F_2$ and if $G \sqsubseteq F_1$ and $G \sqsubseteq F_2$ then $G \sqsubseteq F_1 \sqcup F_2$. Generalisation is a *total* function. If the feature structures are incomparable, eg [**sing**] and [**pl**], we just end up with the feature structure $[\top]$.

Another operation we could define looks very much like generalisation, but uses the 'meet' $\sqcap$ operation on types, which makes it a partial map, if we use the type hierarchy $\langle \mathsf{TYPE}, \sqsubseteq \rangle$,

$$\oplus : \mathcal{F} \times \mathcal{F} \rightharpoonup \mathcal{F}$$

$$\begin{array}{ccccc} 1 & & & & \mathsf{TYPE} \\ \downarrow \langle q_0, q_0' \rangle & & & & \uparrow \sqcap \\ (Q_1 \otimes Q_2) \times \mathsf{FEAT} & \xrightarrow[\delta_1 \otimes \delta_2]{} & Q_1 \otimes Q_2 & \xrightarrow[\alpha_1 \times \alpha_2]{} & \mathsf{TYPE} \times \mathsf{TYPE} \end{array}$$

But if we use $\langle \mathsf{TYPE}, \sqsubseteq \rangle_\perp$ it is another total operation. In the example above, we end up with $F_1 \oplus F_2$ as

$$\begin{bmatrix} \textbf{phrase} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \perp \end{bmatrix} \end{bmatrix}$$

Note that the operation $\oplus$ has not been discussed in the literature, probably because it is not very useful.

Unification

Unification of feature structures is defined as a *partial* function denoted by $\sqcap: \mathcal{F} \times \mathcal{F} \rightharpoonup \mathcal{F}$. The definition for (typed) feature structures follows broadly the definition for untyped feature structures. Carpenter presents a very simple algorithm - attributed by him to Moshier - to compute it. Intuitively, the difference from usual feature structures unification is that

> If F_1 and F_2 are feature structures of types t_1 and t_2 respectively, then $F_1 \sqcap F_2$ has to have type $t_1 \sqcap t_2$. Thus if $t_1 \sqcap t_2$ does not exist then unification fails.

For example suppose we have F_1 and F_2 below and we know **phrase** $\sqsubseteq$ **sign**:

$$F_1 = \begin{bmatrix} \textbf{phrase} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \end{bmatrix} \end{bmatrix} \quad F_2 = \begin{bmatrix} \textbf{sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{NUM} = \textbf{pl} \end{bmatrix} \end{bmatrix}$$

Then

$$F_1 \sqcup F_2 = \begin{bmatrix} \textbf{phrase} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{pl} \end{bmatrix} \end{bmatrix}$$

But if F_1 and F_2 are as below:

$$F_1 = \begin{bmatrix} \textbf{phrase} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{sing} \end{bmatrix} \end{bmatrix} \quad F_2 = \begin{bmatrix} \textbf{sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{pl} \end{bmatrix} \end{bmatrix}$$

The unification $F_1 \sqcap F_2$ fails, as the information F_1 and F_2 convey about the feature NUMber is not consistent. Finally unifying

$$F_1 = \begin{bmatrix} \textbf{phrase} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{sing} \end{bmatrix} \end{bmatrix} \quad F_3 = \begin{bmatrix} \textbf{sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \top \\ \text{NUM} = \textbf{sing} \end{bmatrix} \end{bmatrix}$$

we end up with:

$$F_1 \sqcup F_2 = \begin{bmatrix} \textbf{phrase} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{sing} \end{bmatrix} \end{bmatrix}$$

Now we define unification algebraically in two steps. Recall that to unify feature structures F_1 and F_2 we want to 'union' the partial maps δ_1 and δ_2, making sure that

- the two initial nodes are made the 'same';
- once a feature f appears in both feature structures in a consistent way, this feature is only once in the unification.

Trying to make the initial nodes the same, we have to check that the relation δ_3 that arises from this 'identification' and subsequent ones, is really a partial map, not a relation.

Note that, given two feature structures F_1 and F_2, we can (without loss of generality) consider the sets of nodes Q_1 and Q_2 disjoint. We then write $Q_1 + Q_2$ for the disjoint union of Q_1 and Q_2.

Define the union $\delta_1 + \delta_2$ of the transition functions δ_1 and δ_2 by

$$(Q_1 + Q_2) \times \mathsf{FEAT} \xrightarrow{\delta_1 + \delta_2} (Q_1 + Q_2)$$

As a graph the partial map $\delta_1 + \delta_2$ is disconnected, has two initial nodes and a feature f may appear in both components. Thus to make the initial nodes the same we define an equivalence relation on the set of nodes $Q_1 + Q_2$.

Definition 8 *Given feature structures $F_1 = \langle Q_1, q_0, \delta_1, \alpha_1 \rangle$ and $F_2 = \langle Q_2, q_0', \delta_2, \alpha_2 \rangle$ in $\mathcal{F}$, we define the equivalence relation '$\bowtie$' on the set $Q_1 + Q_2$ as the least equivalence relation such that:*

- $q_0 \bowtie q_0'$;
- $\delta_1(f, q) \bowtie \delta_2(f, q')$ *iff both are defined and* $q \bowtie q'$.

Because we need to identify nodes in a coherent fashion, the unification operation is more complicated from the algebraic point-of-view than the operation of generalisation. One observation is that if one has a (partial or not) map $f: A \to A$ and we take an equivalence relation on A, it is obvious how to define an induced map $[f]: A/\sim \to A/\sim$, we just say $[f]([a]) = [f(a)]$. But if the given map $f: A \to B$ has a different codomain it is not so obvious how to define an induced map $[f]$. One solution is to say $[f]$ is definable iff $[f]([a]) = [f(a)]$ makes sense, ie if whenever $a \sim a'$ then $f(a) = f(a')$. This is taking the identity equivalence relation on B. That is exactly what is done with unification of untyped feature structures, where B is the set of $ATOMS$ and the map α names the atomic nodes. When we merge the graphs of F_1 and F_2, we say $F_1 \sqcap F_2$ is defined if $\alpha([q']) = a$ for any $q' \in [q]$. But if one has a partial order on the set of types TYPE we have more possibilities.

We define the unification of F_1 and F_2 as follows:

Definition 9 *Given feature structures $F_1 = \langle Q_1, q_0, \delta_1, \alpha_1 \rangle$ and $F_2 = \langle Q_2, q_0', \delta_2, \alpha_2 \rangle$ in $\mathcal{F}$, their unification $F_1 \sqcap F_2$ is the feature structure*

$$F_1 \sqcap F_2 = \langle Q_\bowtie, [q_0], \delta_\bowtie, \alpha_\bowtie \rangle$$

where:

- *The set of nodes $Q_\bowtie$ is given by the equivalence classes $(Q_1 + Q_2)/\bowtie$.*
- *The new initial node is the equivalence class $[q_0]$.*

- *The transition function $\delta_{\bowtie}$ is given by the equivalence class of the union of the transition functions $\delta_1 + \delta_2$, when it is defined, that is:*

$$\delta_{\bowtie}([q], f) = [\delta_1 + \delta_2(q, f)] \text{ if } \delta_1 + \delta_2(q, f) \text{ is defined}$$

- *The new typing function $\alpha_{\bowtie}$ is the 'meet' of the types in the equivalence class of q, that is $\alpha_{\bowtie}([q]) = \sqcap\{\alpha_i(q') | q' \bowtie q\}$*

provided that $F_1 \sqcap F_2$ is not cyclic. If $F_1 \sqcap F_2$ is cyclic we say that unification fails.

But the same way we could do generalisation with $\sqcup$ or $\sqcap$ on types, we can do unification with either. Looking at it from the graph-theoretical viewpoint we are glueing or merging the graphs, if they are consistent and then choosing either $\sqcup$ or $\sqcap$ for the result type. The operation described above - true unification - chooses the meet $\sqcap$ of types. We could as well define an operation $\oslash: \mathcal{F} \times \mathcal{F} \rightharpoonup \mathcal{F}$, doing unification of graphs but choosing the join $\sqcup$ of types.

Unification could also be defined through the subsumption order of feature structures, which is a theorem in Carpenter (1990). The unification may FAIL, but if it does succeed, the result of the unification is the meet (or greatest lower bound) of the feature structures being unified. Thus $F_1 \sqcap F_2 \sqsubseteq F_1$ and $F_1 \sqcap F_2 \sqsubseteq F_2$ and if $G \sqsubseteq F_1$ and $G \sqsubseteq F_2$, then $F_1 \sqcap F_2 \sqsubseteq G$. That is very reasonable as the unification gives us the *conjunction* of the information in F_1 and F_2, if they are consistent.

It is worth noting that a product is used for 'join' of information and a coproduct, albeit a complicated one, is used for a 'meet' of information. This is reminiscent of the situation in Domain Theory; the similarity between feature structures and domains has been pointed out and used by several people in different ways, see Pereira and Shieber (1984), Carpenter (1990), Rounds (1988) and Pollard and Moshier (1991) .

Comparing Feature Structures

The structure on TYPE repeats itself on $\mathcal{F}$, which is why we have used the same symbols. Namely $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ is a partial order, where $\sqcap$ is called unification of types and $\sqcup$ is called generalisation of types. Also $\langle \mathcal{F}, \sqsubseteq \rangle$ is a pre-order, where $\sqcap$ is given by unification and $\sqcup$ is given by generalisation. The same way we said before that two types were consistent if $t_1 \sqcap t_2$ existed, we can now say that F_1 and F_2 are consistent if their unification $F_1 \sqcap F_2$ exists. Moreover $\langle \mathcal{F}, \sqsubseteq \rangle$ is a bounded complete pre-order. If $F \sqsubseteq F_1$ and $F \sqsubseteq F_2$ then $F_1 \sqcap F_2$ exists and $F \sqsubseteq F_1 \sqcap F_2$. If we deal with $\langle \mathsf{TYPE}, \sqsubseteq \rangle_{\perp}$ we can say that types are consistent if $t_1 \sqcap t_2$ exists and is different from $\perp$.

Apart from being typed the feature structures above are very similar to the traditional ones in Shieber's book (1986). In particular, we do not support *cyclic* feature structures, so, as mentioned before, the set of nodes Q is a *acyclic* connected or rooted graph. There are two main reasons to allow cyclic feature structures. One is implementational, since the check for cycles (no occurs-in check) during unification is computationally expensive. The other one is more conceptual, as mathematically one of the problems with the assumption that feature structures are acyclic is that you can start with two acyclic feature structures, and their unification is cyclic. This problem can be 'solved' by checking for cyclicity a posteriori, which is not very elegant.

On the other hand, if one accepts cyclic feature structures, apart from problems with checking for well-formedness (next section), one does not have a join semi-lattice if the set of nodes is finite, cf. Pollard and Moshier, pg 297. Also, as Pollard and Sag (1987) put it

"In general, cyclic graphs present certain mathematical and computational complexities which are best avoided, although linguistic applications for them have been suggested from time to time."

One of the differences between the feature structures here and the ones in PATR-II is that, because of the type hierarchy, we can support in the formalism disjunction of atomic values. That happens because we can 'complete' the hierarchy $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ with more 'generic' types. For example we can add a type **num** above the types **sing** and **pl**, which stands for either of the types singular or plural. In the traditional definition of a feature structure, since $\delta: Q \times \mathsf{FEAT} \rightarrow Q$ is a partial map, to say that the feature NUMBER could have values **sing** or **pl** on a node q would not be possible - a partial map cannot have two values at some node. Another way to deal with this problem is to introduce a notion of *set-valued* feature structure. This is done, using distinct, but similar, approaches in Rounds (1988) and Pollard and Moshier (1991). The system we describe also supports atomic negation where this is to be regarded purely as a notational device. If $t_1, t_2, \ldots t_n$ are the subtypes of some atomic type t then $\neg t_1$ is to be interpreted as $t_2 \sqcup \cdots \sqcup t_n$.

If we write $\langle \mathcal{UF}, \sqsubseteq \rangle$ for PATR-II untyped feature structures (using Moshier-Rounds definition) and their subsumption order, then we have a map that 'forgets' the (non-atomic) types and the ordering among them

$$\langle \mathcal{F}, \sqsubseteq \rangle \stackrel{Erase}{\longrightarrow} \langle \mathcal{UF}, \sqsubseteq \rangle$$

but preserves subsumption. We also have a function

$$\langle \mathcal{UF}, \sqsubseteq \rangle \stackrel{trivtyp}{\longrightarrow} \langle \mathcal{F}, \sqsubseteq \rangle$$

which assigns the trivial type '$\top$' to every non-terminal node.

5 Constraints

Until now the typing of feature structures is doing nothing useful. Any arbitrary assignment of types is possible. Thus the idea here is to 'carve out' from the pre-order of all feature structures $\langle \mathcal{F}, \sqsubseteq \rangle$ a subset, the subset of the *well-formed* feature structures $\langle \mathcal{WF}, \sqsubseteq \rangle$ and these will be well-formed with respect to a given constraining function.

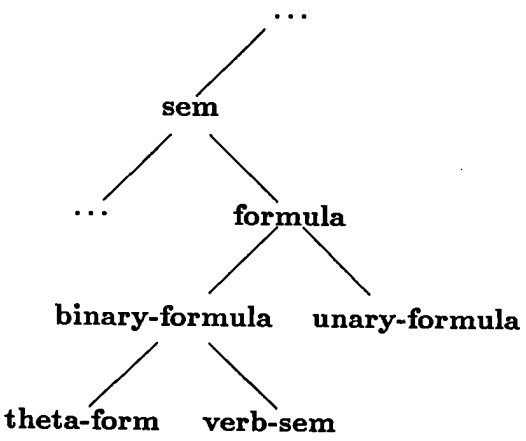
Here we depart substantially from Carpenter's work. To make the type system work Carpenter describes an "appropriateness specification", that is a partial map

$$\mathit{Approp}: \mathsf{TYPE} \times \mathsf{FEAT} \rightarrow \mathsf{TYPE}$$

(satisfying some conditions) which says that for certain types some features are appropriate yielding some other types. His partial map *Approp* is equivalent to a total function

$$\bar{\mathit{A}}\mathit{pprop}: \mathsf{TYPE} \rightarrow [\mathsf{FEAT} \rightharpoonup \mathsf{TYPE}]$$

which corresponds to associating to each type a list of its appropriate features with types. Each list of features with types can be seen as a very simple one-level only feature structure. For example, if we have a fragment of a type hierarchy as follows:



then an appropriateness specification for the type **formula**, which has features IND, PRED and ARG1, could be:

$$\begin{bmatrix} \textbf{formula} \\ \text{IND} = \textbf{entity} \\ \text{PRED} = \textbf{logical-pred} \\ \text{ARG1} = \textbf{sem} \end{bmatrix}$$

The function $\overline{A}pprop$ is Carpenter's constraining function and *well-typed* and *totally well-typed* features structures in Carpenter's notation are so with respect to the given function $\overline{A}pprop$.

Our function corresponding to Carpenter's appropriateness specification is more general than his. We generalise his idea by associating with each type a whole feature structure in our constraint specification function. Thus every type in $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ must have exactly one associated feature structure which acts as a constraint on all feature structures of that type. This associated feature structure is given by the function

$$C\colon \langle \mathsf{TYPE}, \sqsubseteq \rangle \rightarrow \langle \mathcal{F}, \sqsubseteq \rangle$$

but one can think about the constraint specification function C as the set of basic feature structures $C(t_1), C(t_2), \ldots, C(t_k)$ - the constraint feature structures - corresponding to the enumeration of the types $t_1, t_2, \ldots, t_k$ in $\langle \mathsf{TYPE}, \sqsubseteq \rangle$. As an example of a re-entrant constraint feature structure we could have the constraint for the type **theta-formula**

$$\begin{bmatrix} \textbf{theta-formula} \\ \text{IND} = <0>\textbf{=eve} \\ \text{PRED} = \textbf{theta-relation} \\ \text{ARG1} = <0> \\ \text{ARG2} = \textbf{dummy-or-obj} \end{bmatrix}$$

We think of Carpenter's appropriateness conditions as being information which can be extracted from the constraint feature structures $C(t_i)$ by reading only their first level. Thus our equivalent of Carpenter's "appropriateness conditions" are only indirectly specified but we actually refer to the "appropriate features" of a type which essentially means the top level features of the constraint feature structures $C(t_i)$.

The constraints imposed on a type are inherited by all subtypes of this type — (cf Carpenter's upward closure and monotonicity). In mathematical terms that means that the function C is monotonic, a very reasonable assumption, since its domain is the poset $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ and its codomain the pre-order $\mathcal{F}$ ordered by subsumption $\langle \mathcal{F}, \sqsubseteq \rangle$. Thus:

Monotonicity Given types t_1 and t_2 if $t_1 \sqsubseteq t_2$ then $C(t_1) \sqsubseteq C(t_2)$

Of course a subtype may introduce new features — thus if we have the same fragment of a type hierarchy as before and the type **formula** had as its constraint feature structure the previous example, then its subtype **binary-formula** could have as constraint:

$$\left[\begin{array}{l} \textbf{binary-formula} \\ \text{IND} = \textbf{entity} \\ \text{PRED} = \textbf{logical-pred} \\ \text{ARG1} = \textbf{sem} \\ \text{ARG2} = \textbf{sem} \end{array}\right]$$

But not all monotonic functions $C\colon \langle \mathsf{TYPE}, \sqsubseteq \rangle \rightarrow \mathcal{F}$ determine a constraint function. Another obvious condition on constraints is:

Type For a given type t, if $C(t)$ is the feature structure F given by $\langle Q, q_0, \delta, \alpha \rangle$ then $\alpha(q_0) = t$.

Mathematically this means that composing the function C with the function *type-of* gives the identity on the set TYPE; in other words we have a retraction,

$$\mathsf{TYPE} \underset{C}{\overset{\textit{type-of}}{\leftrightarrows}} \mathcal{F}$$

Note however that composing *type-of* with C only gives the identity if you started with a very special feature structure, ie one of the basic constraint feature structures $C(t_i)$. The condition **Type** is part of the 'modelling convention' in Pollard and Moshier (1991).

We also want a condition similar to Carpenter's minimal - for us maximal - introduction. That is we want a feature to be only introduced at one point in the type hierarchy - it will be inherited as an appropriate feature by subtypes of that type. Recall from section 3 that $Feat_0(F)$ is the set of features that appear on the top level of the feature structure F and that $F|_q$ is the feature structure F starting from the node q. We need another simple definition:

Definition 10 *Given a type $t \in$ TYPE and a candidate constraint function $C(t)$ let the set of* appropriate features of the type *t be the set of features $AppFeat(t)$ that appear on the top level of the constraint $C(t)$, that is $Feat_0(C(t))$.*

Note that a feature f may be present in several sets $AppFeat(t)$, for different t's and that C uniquely determines $AppFeat(t)$. Then our version of maximal introduction, can be described as follows:

Maximal Introduction Given *AppFeat* obtained from $C(t)$ say C satisfies a maximal introduction condition if for every feature $f \in$ FEAT there is a unique type $t = Maxtype(f)$ such that $f \in AppFeat(t)$ and there is no type s such that $t \sqsubset s$ and $f \in Appfeat(s)$.

Observe that given a candidate for a constraint specification function $C(t)$ we can always extract its first level function, which we call

$$\mathit{AppSpec}\colon \mathsf{TYPE} \rightharpoonup [\mathsf{FEAT} \to \mathsf{TYPE}]$$

as it corresponds to one of Carpenter's appropriateness specification. For Carpenter an appropriateness specification has to satisfy two conditions, the first corresponds to our **Monotonicity** and the second to (for us) **Maximal Introduction**. Our condition **Type** is not necessary in Carpenter's presentation because for each type he gives directly the list of appropriate features and their types. Also since C is monotonic and *AppSpec* is only a restriction of C *AppSpec* is also monotonic. Thus given a constraint specification function $C(t)$ we get automatically an appropriateness specification in Carpenter's sense.

But another condition on the constraining function C seems very reasonable. This condition says that the constraining feature structures $C(t_i)$ must be compatible with each other.

Compatibility If $C(t_1) = F_1$ and some t_2 appears in F_1, that is, if F_1 is the feature structure $\langle Q_1, q_0, \delta_1, \alpha_1\rangle$ and $\alpha_1(q) = t_2$ for some q in Q_1, then $C(t_2) = F_2$ is such that $F_1|_q \sqsubseteq F_2$. Moreover, $\mathit{Feat}_0(F_1|_q) = \mathit{Feat}_0(F_2)$.

Also note that consistency of the constraining feature structures $C(t_i)$, for consistent types t_i is enforced simply by monotonicity of the function C. If types t_1 and t_2 are consistent - as types - $t_1 \sqcap t_2$ exists and $t_1 \sqcap t_2 \sqsubseteq t_1$ and $t_1 \sqcap t_2 \sqsubseteq t_2$. Since C is monotonic $C(t_1 \sqcap t_2) \sqsubseteq C(t_1)$ and $C(t_1 \sqcap t_2) \sqsubseteq C(t_2)$. Thus the unification of $C(t_1)$ and $C(t_2)$ as feature structures, $C(t_1) \sqcap C(t_2)$ exists ($\mathcal{F}$ is bounded complete) and is such that $C(t_1 \sqcap t_2) \sqsubseteq C(t_1) \sqcap C(t_2)$. Thus the constraint feature structures $C(t_1)$ and $C(t_2)$ are consistent as feature structures, simply by monotonicity of C.

The compatibility condition is reminiscent of Sheaf Theory, as it says that where the constraining features structures $C(t_1), C(t_2), \ldots, C(t_k)$ overlap they agree with each other.

Definition 11 *A function* $C\colon \langle \mathsf{TYPE}, \sqsubseteq\rangle \to \langle \mathcal{F}, \sqsubseteq\rangle$ *is a constraint specification function with respect to* FEAT *and* $\langle \mathsf{TYPE}, \sqsubseteq\rangle$ *if it satisfies* **Monotonicity**,**Type**, **Maximal Introduction** *and* **Compatibility**.

If C is a constraint function wrt the type hierarchy TYPE, then we say:

Definition 12 *A given feature structure* $F = \langle Q, q_0, \delta, \alpha\rangle$ *in* $\mathcal{F}$ *is a* well-formed feature structure *wrt the constraint specification* C *iff for all* $q \in Q$,

- $F|_q \sqsubseteq C(\alpha(q))$ *and*
- $\mathit{Feat}_0(F|_q) = \mathit{Feat}_0(C(\alpha(q)))$.

We call the collection of all well-formed feature structures $\mathcal{WF}$. $\mathcal{WF}$ *is a pre-order as the order in* $\langle \mathcal{F}, \sqsubseteq\rangle$ *restricts to* $\mathcal{WF}$.

Recap:

1. We wanted to carve out from the collection of all feature structures $\langle \mathcal{F}, \sqsubseteq\rangle$ a collection of well-formed ones $\langle \mathcal{WF}, \sqsubseteq\rangle$ with good properties.

2. To do that we use a constraining function C: TYPE $\rightarrow \langle \mathcal{F}, \sqsubseteq \rangle$, which tells us which feature structures are well-formed, which ones are not - definition above. To calculate whether any feature structure F is well-formed we have to calculate some subsumptions and some sets of features.

3. But not any function C: TYPE $\rightarrow \mathcal{F}$ is a constraining function. To be a constraining function C must satisfy the four conditions **Monotonicity,Type,Compatibility** and **Maximal Introduction.**

Note that the constraint feature structures $C(t_i)$ are all well-formed by definition, using the compatibility condition, but the definitions are not circular as the process of checking compatibility of $C(t_i)$'s bottoms-out at the atomic types. Also the function *AppFeat* that we used to define **Maximal Introduction**, could be obtained by forgetting some information present in *AppSpec*, namely the target type.

Type Checking and Type Inferencing

Maximal (for Carpenter minimal) introduction allows some untyped feature structures to be introduced into the system by the user which are then given the most general possible type. It is not necessary to put types on a path specification for example — so:

```
<agr pers> = 1
```

would expand out into the feature structure

$$\begin{bmatrix} \textbf{sign} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \end{bmatrix} \end{bmatrix}$$

assuming that the feature AGR was introduced at the type **sign** and that the feature PERS was introduced at the type **agr.**

As each feature in FEAT has a maximal type $Maxtype(f)$ at which it can be introduced, given a set of features $S \subseteq$ FEAT, either the set $T = \{Maxtype(f) \mid f \in S\}$ is inconsistent or it has a greatest lower bound $\sqcap T$ where that set of features S will become valid. To show that one uses the bounded completeness of TYPE again. This is interesting because we are not assuming any structure on the set of features, FEAT, but the maximal introduction condition induces a notion of 'consistency' of sets of features.

The same way Carpenter has a **Type Inference** theorem we have a *Well-formed Inference* proposition, which says:

Proposition 1 *Given a constraint specification function C there is a partial map*

$$Fill: \langle \mathcal{F}, \sqsubseteq \rangle \rightarrow \langle \mathcal{WF}, \sqsubseteq \rangle$$

such that for each F in $\langle \mathcal{F}, \sqsubseteq \rangle$, Fill returns a well-formed feature structure $F' = Fill(F)$ or fails.

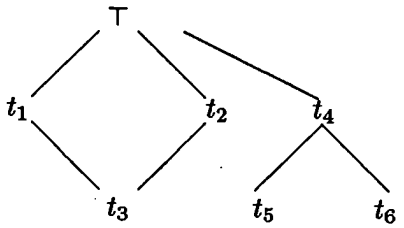
But note that the procedure to transform any feature structure into a well-formed one may fail. As Carpenter puts it not all feature structures are *typeable.*

Unification of well-formed feature structures

Unification of two well-formed feature structures will involve, in general, unifying with the constraint feature structure associated with the meet of their types in order to produce a well-formed result.

If F_1 and F_2 are well-formed feature structures of types t_1 and t_2 respectively, then $F_1 \sqcap F_2$, if it exists, has type $t_1 \sqcap t_2$. Since F_1 and F_2 are well-formed, in particular we know that $F_1 \sqsubseteq C(t_1)$ and $F_2 \sqsubseteq C(t_2)$. Thus if F_1 and F_2 are consistent, $F_1 \sqcap F_2 \sqsubseteq C(t_1) \sqcap C(t_2)$. But to be well-formed $F_1 \sqcap F_2$ has to satisfy $F_1 \sqcap F_2 \sqsubseteq C(t_1 \sqcap t_2)$ and $C(t_1 \sqcap t_2)$ might be more specific than $C(t_1) \sqcap C(t_2)$.

Consider the following example. If we have a type hierarchy as:



And the constraints on types t_1,t_2 and t_3 are (assume the other types are atomic, i.e their constraints are themselves) $C(t_1) = \begin{bmatrix} t_1 \\ f_1 = t_4 \end{bmatrix}$ $C(t_2) = \begin{bmatrix} t_2 \\ f_2 = \top \end{bmatrix}$ $C(t_3) = \begin{bmatrix} t_3 \\ f_1 = t_5 \\ f_2 = \top \\ f_3 = \top \end{bmatrix}$

Then $t_3 = t_1 \sqcap t_2$ but $C(t_3) \sqsubset C(t_1) \sqcap C(t_2) = \begin{bmatrix} t_3 \\ f_1 = t_4 \\ f_2 = \top \end{bmatrix}$

If we have well-formed feature structures

$F_1 = \begin{bmatrix} t_1 \\ f_1 = t_6 \end{bmatrix}$ $F_2 = \begin{bmatrix} t_2 \\ f_2 = \top \end{bmatrix}$

Then their unification exists: $F_1 \sqcap F_2 = \begin{bmatrix} t_3 \\ f_1 = t_6 \\ f_2 = \top \end{bmatrix}$

But $F_1 \sqcap F_2$ is not a well-formed feature structure of type t_3 as $F_1 \sqcap F_2 \not\sqsubseteq C(t_3)$. Moreover it cannot be extended to a well-formed feature structure, because its value for f_1 is inconsistent with the constraint for t_3.

The problem of starting with well-formed feature structures and not getting a well-formed unification[2] can be solved by saying that the well-formed unification of F_1 and F_2 is the well-formed feature structure $F_1 \sqcap F_2 \sqcap C(t_1 \sqcap t_2)$, if it exists. Another possibility would be to ask C to preserve meets, $C(t_1 \sqcap t_2) = C(t_1) \sqcap C(t_2)$.

Thus the unification operation is not a closed operation in $\mathcal{WF}$ the same way unification is not a closed operation wrt well-typed or totally well-typed structures for Carpenter.

Well-formed unification of well-formed feature structures will result in a structure which is totally well-typed (strongly typed) in Carpenter's sense in that all the features which are possible for that type will be present in the feature structure.

[2]Carpenter's appropriateness specification also behave in this way.

This example illustrates that although the ordering on constraints given by subsumption must be consistent with the type hierarchy, that is $t_1 \sqsubseteq t_2$ implies $C(t_1) \sqsubseteq C(t_2)$, we do not have that $C(t_1 \sqcap t_2) = C(t_1) \sqcap C(t_2)$ nor that $t_1 \sqsubset t_2$ implies $C(t_1) \sqsubset C(t_2)$.

6 Internal and External Logics

One can think about logic in the context of feature structures in two orthogonal ways. One way is to think about the collection $\mathcal{F}$ as a set with some algebraic operations and try and see how these operations compare with the algebraic interpretations of traditional logical connectives. In this sense every set which has the structure of a Boolean algebra is a model of classical propositional logic, the same way a set which has a Heyting algebra structure is a model of intuitionistic propositional logic, etc... That is what we are calling the "internal logic", as it is logical structure that is already present in the algebraic definitions.

The second way is to produce a logical calculus (or a set of formulas) from the feature structures. Thus we can read the paths as atomic formulae and add the traditional logical connectives linking these formulae. That was Kasper and Rounds's idea in their seminal paper (1986) and much has been done about adding more logical connectives, for instance Moshier and Rounds (1987) add intuitionistic implication and negation to the logic of feature structures. To make the distinction clearer Carpenter talks about the language of attribute-value "descriptions" for the external logic. Descriptions are then a neat notation for picking up feature structures and we can talk about disjunctive descriptions - even if they cannot be represented by a single feature structure.

7 Internal Logic

The internal logic of feature structures is a very odd propositional logic, where conjunction is partial, that is conjunction only exists for certain pairs of feature structures, the consistent ones.

Note that we could talk about one feature structure implying another $F_1 \Rightarrow F_2$, where we would define $F_1 \Rightarrow F_2$ as the largest feature structure X such that $F_1 \sqcap X \sqsubseteq F_2$, when $F_1 \sqcap X$ is defined, Pollard and Sag (1987). Then we would have a (closed) logic of "partial implication and partial conjunction".

Of course generalisation gives us a 'kind of' disjunction. But the 'disjunction' given by generalisation in $\mathcal{F}$ is not the disjunction one is expecting. For instance, intuitively one expects that if F_1 and F_2 are as below,

$$F_1 = \begin{bmatrix} \textbf{phrase} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{1} \\ \text{NUM} = \textbf{sing} \end{bmatrix} \end{bmatrix} \quad F_2 = \begin{bmatrix} \textbf{phrase} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{2} \\ \text{NUM} = \textbf{sing} \end{bmatrix} \end{bmatrix}$$

Then $F_1 \vee F_2$ should be

$$\begin{bmatrix} \textbf{phrase} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{`1} \vee \textbf{2'} \\ \text{NUM} = \textbf{sing} \end{bmatrix} \end{bmatrix}$$

But $F_1 \sqcup F_2$ is

$$\begin{bmatrix} \textbf{phrase} \\ \text{AGR} = \begin{bmatrix} \textbf{agr} \\ \text{PERS} = \textbf{pers} \\ \text{NUM} = \textbf{sing} \end{bmatrix} \end{bmatrix}$$

Hence $(F_1 \vee F_2) \sqsubseteq (F_1 \sqcup F_2)$, which means that $\sqcup$ is not fine-grained enough to model disjunction. Thinking about the internal logic of feature structures in $\mathcal{F}$ one is reduced to a logic of partial conjunction, partial implication and total (but very strange) "form-of-disjunction".

We can make conjunction total, by adding an inconsistent feature structure. Supposing we have $\langle \mathsf{TYPE}, \sqsubseteq \rangle_\perp$ instead of $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ we have an atomic feature structure $[\perp]$. We could use this feature structure to make unification total, that is we could define $F_1 \sqcap F_2 = [\perp]$, if $F_1 \sqcap F_2$ fails. This is analogous to the situation with types.

Thinking about identities for the operations in $\mathcal{F}$, recall that $[\top]$ behaves as the identity for unification as $F \sqcap [\top] = [\top] \sqcap F = F$ for any F in $\mathcal{F}$. But $[\perp]$ is not an identity for generalisation. If $F \sqcup G = G$ then $F \sqsubseteq G$. If an identity I for generalisation existed it would satisfy $I \sqcup F = F$ for any F, which would imply $I \sqsubseteq F$, for all F's in $\mathcal{F}$, the characteristic of *false*, the identity for disjunction. Then we want to complete the definition of I by saying which is the map $?: 1 \times \mathsf{FEAT} \to 1$. But we cannot define a morphism of feature structures from F to I for every F.

$$\begin{array}{ccccc} Q \times \mathsf{FEAT} & \xrightarrow{\delta} & Q & \xrightarrow{\alpha} & \mathsf{TYPE} \\ \downarrow{\scriptstyle !} & & \downarrow{\scriptstyle !} & & \| \\ 1 \times \mathsf{FEAT} & \xrightarrow{?} & 1 & \longrightarrow & \mathsf{TYPE} \end{array}$$

Because if the diagram above were a morphism "$?(!(q), f)$" would have to be defined and equal to "$!(\delta(q, f))$" for any feature $f \in \mathsf{FEAT}$. That means that the partial map "?" would have $?(*, f) = *$ for all features and that is not a partial map, hence not a feature structure.[3] Even if $[\perp]$ is not ideal as 'the' inconsistent feature structure, say F_1 and F_2 are consistent if $F_1 \sqcap F_2$ exists and is different from $[\perp]$.

Thus if we use $\langle \mathsf{TYPE}, \sqsubseteq \rangle_\perp$ we have total conjunction, but no constant *false*, and disjunction and generalisation are not the same. A very poor logical set up. But of course there are external logics. One of the first external logics was described by Kasper-Rounds and is the subject of section 8.

8 Logic of Descriptions

We recall sections (6,7) of Carpenter (1990), to make some comparisons. Quoting from Carpenter:

[3]Note that we assume that the set FEAT has at least two elements

> A language of descriptions provides a way to talk about feature structures [...].
>
> The well-formed expressions of our attribute-value description language will be taken to *describe* feature structures. We define a notion of logical satisfaction that derives from thinking of feature structures as models of the formulas that describe them.

In the next definition we restrict the set of descriptions in Carpenter[90], but allow some extensions later on.

Definition 13 *The set of* descriptions *over the partial order* $\langle \mathsf{TYPE}, \sqsubseteq \rangle$ *of types and the collection* FEAT *of features is the least set* DESC *such that*

- $t \in \mathsf{DESC}$ *if* $t \in \mathsf{TYPE}$
- $\pi{:}\phi \in \mathsf{DESC}$ *if* $\pi \in \mathsf{FEAT}^*$ *and* $\phi \in \mathsf{DESC}$
- $\pi_1 \doteq \pi_2 \in \mathsf{DESC}$ *if* $\pi_1, \pi_2 \in \mathsf{FEAT}^*$
- $\phi \wedge \psi \in \mathsf{DESC}$ *if* ϕ *and* $\psi \in \mathsf{DESC}$

The idea of providing descriptions as formulas of a logic to be satisfied by some feature structures is already in Pereira and Shieber (1984) , but they, as well as many other people have a richer set of formulas.

Since we restricted the formulas in DESC to the $\wedge$-fragment of propositional logic in the definition above, it does not matter how satisfaction is defined, as the $\wedge$-fragment of classical logic is equivalent to the $\wedge$-fragment of intuitionistic logic. But if we want to add disjunction or implication or negation to DESC, a choice of logical framework becomes necessary. Also different notions of satisfaction will lead to different logical formalisms, which explains why there so many papers in the literature on this topic.

Definition 14 *The* satisfaction relation *relates the collection of feature structures* $\mathcal{F}$ *and the set of descriptions* DESC. *It is the least relation* '$\models$' *such that, if* F *is the feature structure* $\langle Q, q_0, \delta, \alpha \rangle$ *and* $\phi \in \mathsf{DESC}$

- $F \models t$ *if* $t \in \mathsf{TYPE}$ *and* $\alpha(q_0) \sqsubseteq t$
- $F \models \pi{:}\phi$ *if* $F@\pi \models \phi$
- $F \models \pi_1 \doteq \pi_2$ *if* $\delta(\pi_1, q_0) = \delta(\pi_2, q_0)$
- $F \models \phi \wedge \psi$ *if* $F \models \phi$ *and* $F \models \psi$

Note that the type $\top$, which is already in DESC by definition, behaves as the constant *true* for this logic. It is satisfied by any feature structure $F \models \top$, because for all $F \in \mathcal{F}$, $\alpha(q_0) \sqsubseteq \top$.

Recall as well the following usual logical definitions,

Definition 15 *Consider the set of all feature structures that satisfy a certain description* ϕ, *that is* $Sat(\phi) = \{F \in \mathcal{F} | F \models \phi\}$.

If ϕ *is a formula in* DESC, *say that* ϕ *is* satisfiable *if there exists a feature structure* F *that satisfies it, that is the set* $Sat(\phi)$ *is not empty.*

We have the traditional result:

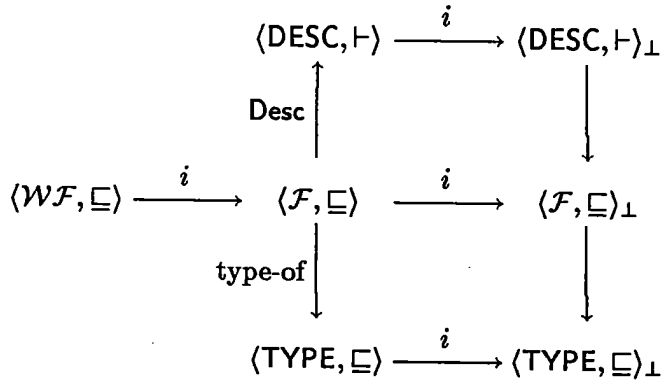
Proposition 2 *If $F_1 \models \phi$ and $F_1 \sqsubseteq F_2$ then $F_2 \models \phi$ (monotonicity).*

For every satisfiable formula ϕ there is a minimal feature structure $MinSat(\phi)$ that satisfies it.

For any feature structure in $\mathcal{F}$ there is a description $\mathsf{Desc}(F)$ *such that*

$$F \cong MinSat(\mathsf{Desc}(F))$$

And computing satisfiability of descriptions is complexity-O.K. Of course the results above are all in Kasper-Rounds, the only reason to recall them here is to remind the reader that these results are another way of expressing the existence of the internal logic. In other words within this small fragment of propositional logic, we have a notion of entailment '$\vdash$' and as entailment is reflexive and transitive, we can think of $\langle \mathsf{DESC}, \vdash \rangle$ as a pre-order, where meet $\sqcap$ is given by conjunction and $\top$ is the constant *true*. Clearly given any feature structure F we can write it as a big conjunction of descriptions, hence the function Desc in the proposition above and the diagram below:



Having established a minimum denominator one could extend the set of descriptions to accommodate several formalisms. If we decide for intuitionistic logic, we can have Moshier and Rounds or Dawar and Vijay-Shanker formalisms. If we decide for classical logic, we can have Kasper and Rounds system and Carpenter's system (both have classical disjunction). Still using classical logic but at right angles, we have G. Smolka's and M. Johnson's systems where DESC has variables and negation. M. Reape also adds variables, but he wants to consider the features in paths of the form $\langle f_1 : \phi \rangle$ as (possibility) modal[4] operators, thus getting a poly-modal logic.

Looking at the systems above we can have $\langle \mathsf{DESC}, \vdash \rangle_{MR}$, $\langle \mathsf{DESC}, \vdash \rangle_{DV}$, $\langle \mathsf{DESC}, \vdash \rangle_{KR}$, $\langle \mathsf{DESC}, \vdash \rangle_{C}$, $\langle \mathsf{DESC}, \vdash \rangle_{J}$, $\langle \mathsf{DESC}, \vdash \rangle_{R}$ etc..

9 Conclusions

We presented a rigorous *mathematical* definition of a system of well-formed typed features structures. Our system is very similar to Carpenter's system (1990), but we allow more expressive constraints to be made over types. In our system each type t_i is constrained by a whole, possibly re-entrant, constraint feature structure $C(t_i)$, while using Carpenter's appropriateness specifications each type is constrained by a list of features and types - a

[4]This also seems to be the case for Category Structures of Gazdar et al.

one-level only feature structure. On the other hand Carpenter allows (general) 'disjunctive constraints' which we do not handle at the moment. It should be noted that we could not use Carpenter's appropriateness specifications alone to give the sort of functionality for inheritance in the type system that we needed, thus the generalisation to well-formedness.

We also pointed out where somewhat different choices could be made in the formalisation of typed feature structures and presented two new operations that one could consider over feature structures looking at them from the mathematical viewpoint only. Finally we briefly discussed one of the reasons why there are many different logics of feature structures on the literature, but for a survey of these logics the reader is referred to the substantial works of Carpenter, Reape and Johnson. There are some other points that we would have liked to have investigated from the algebraic (or categorical) viewpoint. Feature structures 'look like' pointed sets with an action of the set of features FEAT on them, but there is more structure than that to them, as the rootedness condition has to be accounted for. Several of our definitions make essential use of the rootedness condition. A comparison between feature algebras and feature structures as well as the related issue of graph-unification versus term-unification is another point that deserves further investigation. On a different level, it would be nice to check how useful Carpenter's inequalities and extensionality specifications would be for a system like the LKB.

Acknowledgments

Many of the basic ideas and intuitions in this paper are due to Ann Copestake, Ted Briscoe and Antonio Sanfilippo. The way in which these ideas have been formalised is the responsibility of the author, as are any mistakes in the formalisation.

References

Carpenter, B. (1990). Typed feature structures: Inheritance, (in)equations and extensionality. In *Proceedings of the First International Workshop on Inheritance in Natural Language Processing*, pages 9–13, Tilburg, The Netherlands.

Carpenter, B. (1990). Typed feature structures: Inheritance, (In)equations and Extensionality. Notes for a course in the Summer School on Logic, Language and Information, Leuven - Belgium.

Carpenter, B. (to appear). The Logic of Typed Feature Structures. (in preparation) - typescript of December 1990 - To appear in Cambridge Tracts in Theoretical Computer Science.

Carpenter, B., Pollard, C., and Franz, A. (1991). The specification and implementation of constraint-based unification grammars. In *Proceedings of the Second International Workshop on Parsing Technology*, Cancun, Mexico.

Copestake, A., de Paiva, V., Sanfilippo, A. and Briscoe, T. (1991). Functionality of the LKB, Acquilex deliverable, March 1991.

Copestake, A. The LKB: A System fro Representing Lexical Information Extracted from Machine-Readable Dictionaries, this volume.

Davey, B.A. and Priestley, H. (1990). *Introduction to Lattices and Order.* Cambridge Mathematical Textbooks, CUP, Cambridge.

Dawar, A. and Vijay-Shanker, K. (1990). A three-valued interpretation of negation in feature structures descriptions. In *Proceedings of the 27th Annual conference of the Association for Computational Linguistics*, pg 18-24.

Dawar, A. and Vijay-Shanker, K. (1990). An interpretation of negation in feature structure descriptions. In *Computational Linguistics*, 16(1):11-21.

Evans, R. and Gazdar, G. (1990). The DATR papers. Cognitive Science Research Reports CSRP 139, University of Sussex, Sussex.

Gazdar, G., Klein, E., Pullum, G., Carpenter, R., Hukari, T. and Levine, R. (1988). Category Structures. *Computational Linguistics*, 14:1-19

Gazdar, G., Klein, E., Pullum, G., and Sag, I. (1985). *Generalized Phrase Structure Grammar.* Basil Blackwell, Oxford.

Gunther, C. and Scott, D. (1991). Semantic Domains. In *Handbook of Theoretical Computer Science.*

Johnson, M. (1988). *Attribute-Value Logic and the Theory of Grammar*, vol. 14 of *Lecture Notes* CSLI, Stanford.

Karttunen, L. (1984). Features and values. In *Proceedings of the 10th International Conference on Computational Linguistics.*

Kasper, R. T. and Rounds, W. C. (1986). A logical semantics for feature structures. In *Proceedings of the 24th Annual Conference of the Association for Computational Linguistics*, pages 235–242.

Kasper, R. T. and Rounds, W. C. (1990). The logic of unification in grammar. *Linguistics and Philosophy*, 13(1):35–58.

Kay, M. (1984). Functional unification grammar: a formalism for machine translation. In *Proceedings of the 10th International Conference on Computational Linguistics*, pages 75–78.

Moshier, D. (1988). *Extensions to Unification Grammar for the Description of Programming Languages.* PhD thesis, University of Michigan, Ann Arbor.

Moshier, D. and Rounds, W. (1987). A logic for partially specified data structures. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages.*

Pereira, F. C. N. and Shieber, S. M. (1984). The semantics of grammar formalisms seen as computer languages. In *Proceedings of the 10th International Conference on Computational Linguistics*, pages 123–129.

Pollard, C. J. and Moshier, M. D. (in press). Unifying partial descriptions of sets. In Hanson, P., editor, *Information, Language and Cognition*, volume 1 of *Vancouver Studies in Cognitive Science.* University of British Columbia Press, Vancouver.

Pollard, C. J. and Sag, I. A. (1987). *Information-Based Syntax and Semantics: Volume I – Fundamentals*, volume 13 of *CSLI Lecture Notes.* Chicago University Press, Chicago.

Reape, M. (1990). An Introduction to the Theory of Feature Structures. DYANA deliverable, Edinburgh.

Rounds, W. C. and Kasper, R. T. (1986). A complete logical calculus for record structures representing linguistic information. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science*, Cambridge, Massachusetts.

Sanfilippo, A. (1991). *LKB-Encoding of Lexical Knowledge from Machine-Readable Dictionaries*, this volume.

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

Smolka, G. (1988). A feature logic with subsorts. LILOG-REPORT 33, IBM – Deutschland GmbH, Stuttgart, FRG.

Smolka, G. (1989). Feature constraint logics for unification grammars. IWBS Report 93, IBM – Deutschland GmbH, Stuttgart, FRG. To appear in *Journal of Logic Programming.*

Zajac, R. (1990). Issues in the Design of a Language for Representing Linguistic Information Based on Inheritance and Feature Structures, this volume.

LKB Encoding of Lexical Knowledge from Machine-Readable Dictionaries

Antonio Sanfilippo
Computer Laboratory, University of Cambridge

Introduction

The research reported in this paper is concerned with the encoding of lexical information within a Knowledge Base which uses a typed graph unification formalism as representation language, and it was developed as part of a larger study of English verbs in the context of the ACQUILEX project[1]. The general aim of the work is to design a system of information structures which provide a suitable way of representing properties of word forms extracted from machine readable dictionaries within a lexical component for Natural Language Processing systems. In setting up such a lexical component, one finds that a considerable amount of information is often repeated across sets of word entries. To make the task of grammar writing more efficient, the information structures which are shared by related word entries can be expressed in the form of partially specified templates and related to the entries they characterize through inheritance. Shared information across sets of partially specified templates can be factored out and conveyed using the same technique. This makes it possible to avoid redefining the same information structures across lexical templates, thus reducing a great deal of redundancy in the specification of word forms. For example, general properties of intransitive verbs concerning subcategorization and argument structure can be simply stated once, and then inherited by lexical templates which provide word specific information (e.g. selectional restrictions, lexical aspect, orthography/phonology, predicate sense). Likewise, properties which are common to all verbs (e.g. part of speech, presence of a subject) or subsets of the verb class (presence of a direct object for transitive and ditransitive verbs) can be defined as templates which subsume all members of the verb class or some subset of it. This approach to word specification provides a highly structured organization of the lexicon according to which the properties of related word types as well as the relation between word types and specific word forms are expressed in terms of structure sharing and inheritance (Flickinger, Pollard & Wasow 1985; Flickinger 1987; Pollard & Sag 1987:191-209).

Following the general insights of this treatment, the goal of this paper is to describe a type system for English verbs which uses the unification-based representation language of the Lexical Knowledge Base designed as part of ACQUILEX (Copestake (this volume) and de Paiva (this volume)) to express inheritance and structure sharing in the lexicon. In §1, a general characterization of the information structures used to encode properties of word forms is given along with the specification of a grammar formalism which allows to test the appropriatedness of lexical representations in a parsing context. In §2 and §3, more specific information structures are described which provide a characterization of syntactic and semantic properties for a large subset of English verb types. In §4, the syntactic and semantic templates described in §2 and §3 are integrated to yield sign-based representations of verb types. The paper concludes with a study of psychological verbs where it is shown how word-sense specific information derived from machine readable dictionaries can be related to the system of verb types developed.

[1] *The Acquisition of Lexical Knowledge for Natural Language Processing Systems*, Esprit BRA-3030.

1 A Sign-Based Approach to Lexical Specification

While our description of English verb types is meant to be compatible with several theoretical approaches, it is obvious that reference to a specific grammar framework is necessary if we wish to test the fragment built. Our choice of grammar formalism was thus dictated by the need to parse illustrative examples while remaining as theory-neutral as possible. One way to do so is to make lexical descriptions rich enough so that the number of grammar rules needed for parsing can be reduced to a minimum. For example, by providing detailed information about subcategorization properties of verbs we were able to reduce to three the number of rules needed to parse sentences relative to some thirty verb types.[2]

In keeping with a sign-based approach to linguistic analysis (Pollard & Sag 1987, Zeevat *et al.* 1987), words and phrases are represented as (typed) feature structure where orthographic, categorial (e.g. syntactic) and semantic information is simultaneously represented as a conjunction of attribute-value pairs forming a **sign**:

(1)
```
Parents = top
[sign
 ORTH:[orth ]
 CAT: [cat ]
 SEM: [sem ]]
```

Lexical signs include a further attribute, **sense-id**, which provides dictionary information about word senses.

(2)
```
Parents = sign
[lex-sign
 ORTH:[orth ]
 CAT: [cat ]
 SEM: [sem ]
 SENSE-ID: [sense-id
            FS-ID: string
            LANGUAGE:language
            DICTIONARY:string
            LDB-ENTRY-NO:string
            HOMONYM-NO:string
            WORD: string
            SENSE-NO: string ]]
```

Syntactic properties of signs concerning part of speech and subcategorization are expressed using a Categorial Grammar approach to category specification (Zeevat *et al.* 1987). The category attribute of a sign can be either basic of complex. Basic categories are binary feature structures consisting of a category type, and a series of attribute value pairs encoding morphosyntactic information:[3]

(3)
```
Parents = cat
[basic-cat
 CAT-TYPE:cat-type
 M-FEATS:[m-feats ]]
```

Three types of basic categories are defined according to whether category type is **n** (noun), **np** (noun phrase) or **sent** (sentence); each category type is related to a specific group of morphosyntactic features (**nominal-feats** for **n** and **np**, and **sent-feats** for **sent**):

[2]The complete implementation of the grammar fragment developed with reference to the verb types discussed in the paper is included in a recent release of the LKB software developed by the ACQUILEX group in Cambridge.

[3]A box enclosing a type label, as in the case of **m-feats** in (3), signals that the attribute value pairs associated with the type have been omitted. Occasionally, attribute-value pairs which are not directly relevant to the description at hand will also be omitted.

(4)

```
Parents = basic-cat
[noun-cat
 CAT-TYPE:n
 M-FEATS:[nominal-m-feats
          REG-MORPH:boolean
          AGR: [agr
                PERS: person
                NUM: number
                GENDER: gender ]
          NOMINAL-FORM:nominal-form
          CASE: case
          COUNT:boolean ]]
```

```
Parents = basic-cat
[np-cat
 CAT-TYPE:np
 M-FEATS[nominal-m-feats ]]
```

```
Parents = basic-cat
[sent-cat
 CAT-TYPE:sent
 M-FEATS:[sent-m-feats
          VFORM:vform
          COMP-FORM:comp-form
          REG-MORPH: boolean
          DIATHESIS:  alternation]]
```

Complex categories are recursively defined by letting the type **cat** instantiate a feature structure with attributes RESULT, DIRECTION and ACTIVE. RESULT can take as value either a basic or complex category, ACTIVE is of type SIGN, and the direction attribute encodes order of combination relative to the active part of the sign (e.g. **forward** or **backward**).

(5)

```
Parents = cat
[complex-cat
 RESULT:[cat ]
 DIRECTION: direction
 ACTIVE:[sign ]]
```

The semantics of a sign is a formula. A formula consist of an index, a predicate and at least one argument:

(6)

```
Parents = sem
[formula
 IND: entity
 PRED:logical-pred
 ARG1:[sem ]]
```

The index of a formula is an **entity** which provides partial information about the ontological type denoted by the formula (in the fragment described it corresponds to existential quantification). At present, it will suffice to consider two basic semantic entities — eventualities (**eve**) and individual objects (**obj**) — plus a contentless entity, **dummy**, employed in the semantic characterization of pleonastic noun phrases (see footnote 6). (Additional types of object entities will be introduced in §5 to give a semantic specification of argument variables in psychological predicates.) Both **formula** and **entity** are subtypes of **sem** (semantics). The predicate of a formula is an atomic type which can instantiate either a logical constant (e.g. **and**) or a lexical predicate (e.g. *sleep*). Various subtypes of **formula** can be defined according to predicate arity as shown in (7).

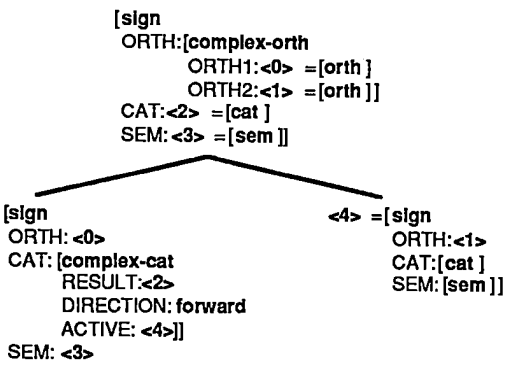
(7)

```
Parents = formula
[unary-formula
 IND: entity
 PRED:logical-pred
 ARG1:[sem ]]
```

```
Parents = formula
[binary-formula
 IND: entity
 PRED:logical-pred
 ARG1:[sem ]
 ARG2:[sem ]]
```

Lexical and phrasal signs are combined to form phrasal signs through rules of forward and backward functional application. Functional application allows a functor sign to combine with an adjacent argument sign just in case the information contained in the active sign of the functor is compatible with the information encoded in the argument sign. The result is a sign whose orthography is a binary feature structure encoding the functor and argument orthographic values, semantics correspond to the semantics of the functor, and category is equal to the category of the functor with its active sign removed. This is shown below with reference to the forward version of the rule where the argument sign occurs to the right of the functor.

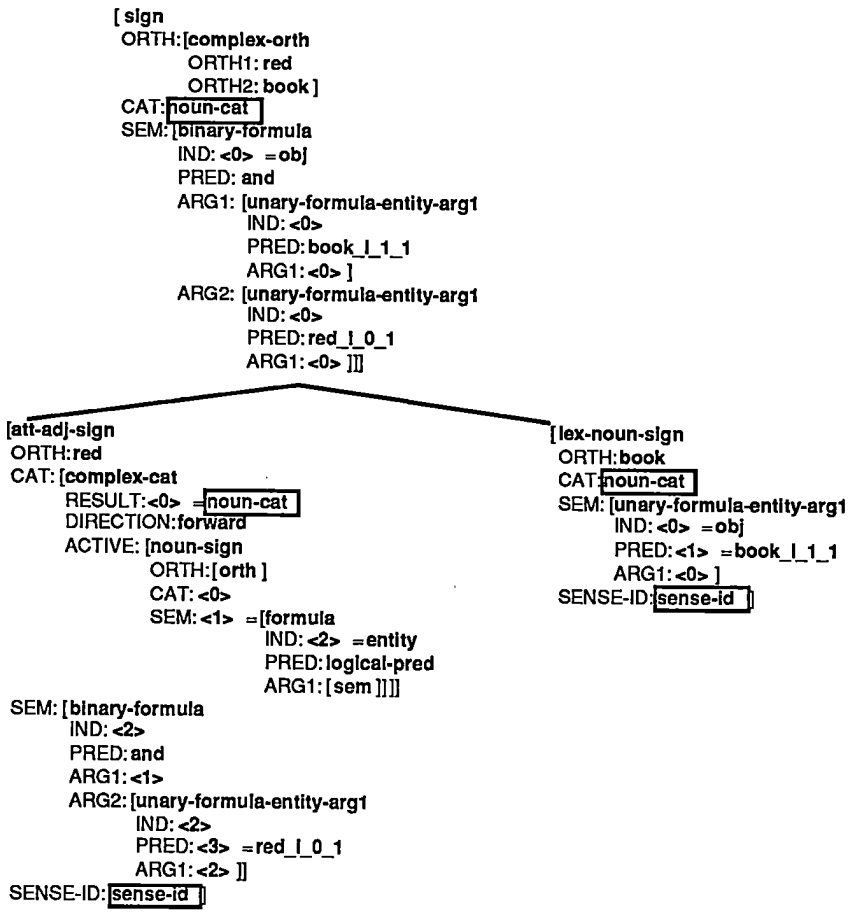
(8)



Forward Application Rule

The derivation for the phrasal sign *red book* in (9) provides a concrete example of how the rule operates.

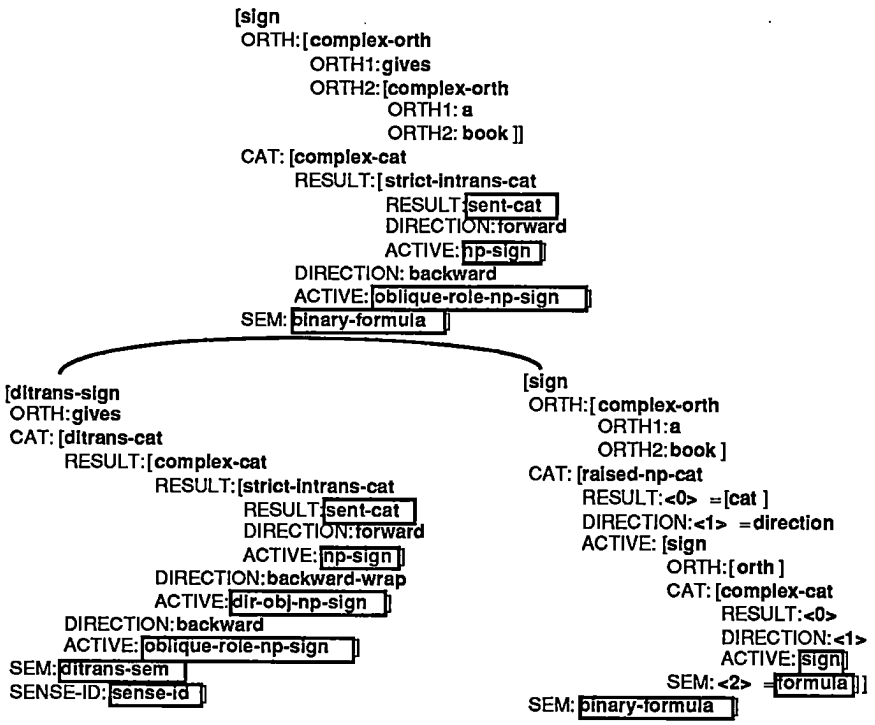
(9)



A third rule type, **backward-wrapping**, allows the next to last active sign of a functor to be consumed before the outermost active sign is. This rule is needed for transitive verbs which take a clausal or oblique complement (see §3 and §4); a sample application relative to the derivation of the phrasal sign *gives a book* is given below.[4]

[4]Following Zeevat *et al.* (1987), noun phrases are treated as polymorphic type-raised arguments.

(10)



2 Verb Semantics

One of the crucial issues in providing a semantic characterization of verbs regards the specification of arguments in terms of thematic roles. It is now often recognized that the elaboration of Davidson's approach to verb semantics proposed by Parsons (1980, 1990) provides a natural way of dealing with this issue. The naturalness of Parson's solution results from assigning thematic roles a central function in the association of a verb with its arguments during sentence formation. According to Parsons, verbs denote properties of eventualities, and thematic roles are relations between eventualities and individuals. The logical form of a sentence involves event quantification over these two types of eventuality-denoting expressions, as indicated in (11) where the semantics of the sentence *John sleeps* is characterized as 'some eventuality of sleeping in which John functions as the agent participant'.

(11) $\exists e[sleep(e) \wedge agent(e, john)]$

Thematic relations thus provide an indispensable layer of semantic interpretation to combine verb and noun phrase meanings into sentence meanings. A specification of this approach to verb semantics and predicate-argument association can be easily expressed in the formalism adopted here, as indicated in (12) where the reentrancies relative to the event variable encode the event binding which in (11) is represented through existential quantification.[5]

[5]In the grammar fragment assumed, proper names are treated as properties of individuals rather than simplex individual as in (11).

(12)

```
Parents = unary-formula-entity-arg1
[verb-formula
 IND: <0> = eve
 PRED: logical-pred
 ARG1: <0> ]

[sign
 ORTH:[complex-orth
        ORTH1:john
        ORTH2:sleeps ]
 CAT:[sent-cat ]
 SEM:[binary-formula
       IND: <0> =eve
       PRED: and
       ARG1: [unary-formula-entity-arg1
              IND: <1> =obj
              PRED: john_1
              ARG1: <1> ]
       ARG2: [strict-intrans-sem
              IND: <0>
              PRED: and
              ARG1: [verb-formula
                     IND: <0>
                     PRED: sleeps_1
                     ARG1: <0> ]
              ARG2: [p-agt-formula
                     IND: <0>
                     PRED: p-agt
                     ARG1: <0>
                     ARG2: <1> ]]]]
```

```
Parents = binary-formula
[theta-formula
 IND: <0> =eve
 PRED: theta-relation
 ARG1: <0>
 ARG2: dummy-or-obj ]
```

The semantic content of thematic relations is computed on the basis of entailments of verb meanings and lexical semantic defaults which qualify the agentivity potential of argument roles for each choice of predicate. This specification reproduces Dowty's account of thematic information (Dowty 1987) within a neo-Davidsonian treatment of verb semantics (Sanfilippo 1990). The basic insights of this approach consist in choosing two sets of properties which contribute to the definition of prototypical agent and patient roles, e.g. (13)-(14), and then defining specific thematic relations for each choice of predicate according to the generalizations in (15).

(13) CONTRIBUTING PROPERTIES FOR THE PROTO-AGENT ROLE
volition, sentience (and/or perception), causes event, movement

(14) CONTRIBUTING PROPERTIES FOR THE PROTO-PATIENT ROLE
change of state, causally affected by event, stationary

(15) a The prototypical agent of a verb (**p-agt**) is the thematic relation associated with the argument having the highest number of proto-agent properties entailed by the meaning of the verb and inherited by default

b The prototypical patient (**p-pat**) is the thematic relation associated with the argument of a transitive verb to which the highest number of proto-patient properties can be ascribed (inherently via entailment relations, and by default).

c All other verbal arguments are associated with an oblique role (**prep**) which subsumes members of the set of contentful prepositions, e.g. *to, from, with.*

(In addition to the proto-agent, proto-patient and prepositional roles, the predicate **no-theta** is introduced to characterize the relation between a pleonastic NP to its governing verb; such a relation is obviously contentless.) In light of this approach to thematic specification, the semantic content of proto-roles is ultimately computed by taking into account verb-sense information. More precisely, entailed properties of verb meanings which qualify argument roles — and at the same time provide a characterization of the semantic class to which a verb belongs — are used to sort proto-role predicates (see §5 for an illustrative example).

The semantics of all verbs signs is a conjunctive formula of type **verb-sem**. The first argument in **verb-sem** is a unary formula (**verb-formula**) which characterizes the verbal predicate as a property of eventualities. The second argument is a formula which encodes the semantics of subcategorized arguments.

(16)
```
Parents = binary-formula
[verb-sem
 IND: <0> = eve
 PRED: and
 ARG1: [verb-formula
        IND: <0>
        PRED: logical-pred
        ARG1: <0> ]
 ARG2: [formula
        IND: <0>
        PRED: logical-pred
        ARG1: [sem ]]]
```

A primary semantic classification of verbs is obtained by specifying how many verbal arguments are encoded in the second argument formula of **verb-sem**. Further distinctions are made according to what kind of verbal arguments are encoded:

- proto-agent, e.g. *John* in *John sleeps* and the sentences below
- proto-patient, e.g. *a book* in *John read a book*
- prepositional, e.g. *to Mary* in *John gave a book to Mary*
- non-thematic, e.g. *Bill* in *Bill seems to be sad*
- pleonastic, e.g. *It* in *It bothers Bill that Mary left*
- predicative (XCOMP), e.g. *to leave* in *John wishes to leave*
- sentential (COMP), e.g. *that Mary left* in *John said that Mary left*

In the case of strict intransitive verbs, e.g. *John sleeps*, the second argument in **verb-sem** is an atomic formula encoding a proto-agent role; the index of the proto-agent formula is equated to the index of the verb formula to indicate that the eventuality described by the verb and proto-agent formulae coincide.

(17)
```
Parents = verb-sem
[strict-intrans-sem
 IND: <0> = eve
 PRED: and
 ARG1: [verb-formula
        IND: <0>
        PRED: logical-pred
        ARG1: <0> ]
 ARG2: [p-agt-formula
        IND: <0>
        PRED: p-agt
        ARG1: <0>
        ARG2: obj ]]
```

Other verbs have semantics of type **trans/intrans-sem** where the second argument is a conjunctive formula whose ARG1 is a proto-agent/non-thematic formula and ARG2 can instantiate one or more argument roles. As for strict intransitives, the index or the proto-agent/non-thematic formula is equal to the eventuality argument of the verbal predicate.

(18)

```
Parents = verb-sem
[trans/intrans-sem
 IND: <0> = eve
 PRED: and
 ARG1: [verb-formula
        IND:<0>]
 ARG2: [formula
        IND: <0>
        PRED: and
        ARG1: [p-agt-or-no-theta
               IND:<0>]
        ARG2: [sem]]]
```

The presence of a single argument in addition to the proto-agent, defines three major subtypes of **trans/intrans-sem**. Verbs which take a single non-clausal complement other than the proto-agent have semantic type **trans/intrans-no-comp/xcomp-sem**. Subtypes of this type (e.g. strict transitives and intransitive which take an oblique object, cf. (20)) have a thematic subject. The index of the thematic formula following the proto-agent role is equal to the eventuality argument variable of the verbal predicate; this would not be so with clausal complements where the event described by the complement need not be equal to that described by the matrix verb, e.g. *Bill thought that Mary would come* (see (21)).

(19)

```
Parents = trans/intrans-sem
[trans/intrans-no-comp/xcomp-sem
 IND: <0> = eve
 PRED: and
 ARG1: verb-formula
 ARG2: [binary-formula
        IND: <0>
        PRED: and
        ARG1: p-agt-formula
        ARG2: [formula
               IND: <0>
               PRED: logical-pred
               ARG1: [sem ]]]]
```

Verbs of this semantic type which encode a proto-patient correspond to strict transitives (e.g. *John read a book*, while those which have a prepositional role correspond to intransitives which subcategorize for an oblique object (*John talked to Bill*).

(20)

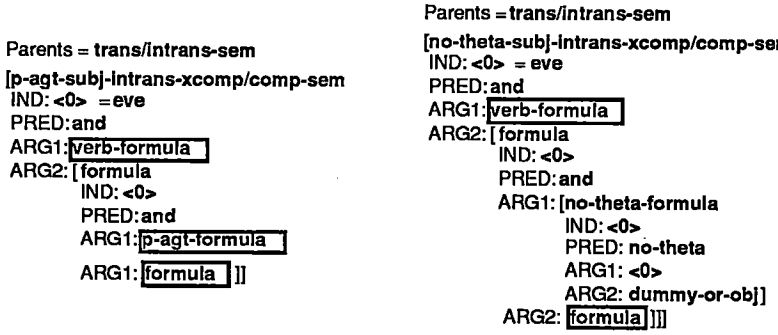
```
Parents = trans/intrans-no-comp/xcomp-sem
[strict-trans-sem
 IND: <0> = eve
 PRED: and
 ARG1: verb-formula
 ARG2: [binary-formula
        IND: <0>
        PRED: and
        ARG1: p-agt-formula
        ARG2: [p-pat-formula
               IND:<0>
               PRED: p-pat
               ARG1: <0>
               ARG2: obj ]]]
```

```
Parents = trans/intrans-no-comp/xcomp-sem
[intrans-obl-sem
 IND: <0> = eve
 PRED: and
 ARG1: verb-formula
 ARG2: [binary-formula
        IND: <0>
        PRED: and
        ARG1: p-agt-formula
        ARG2: [prep-formula
               IND:<0>
               PRED: prep
               ARG1: <0>
               ARG2: obj ]]]
```

The remaining two subtypes of **trans/intrans-sem** are for intransitive verbs which take a clausal complement represented by the last formula of the feature structures shown in (21). These are distinguished according to whether the subject is thematic (type **p-agt-subj-intrans-xcomp-comp-sem**, e.g. *John intended to come, Bill tought that John would come*) or non-thematic (type **no-theta-subj-intrans-xcomp/comp-sem** with subject raising verbs, e.g. *John seems to have solved the problem*, and intransitives which take a pleonastic subject, e.g. *It seems that John might not come, It won't hurt to remind him*):[6]

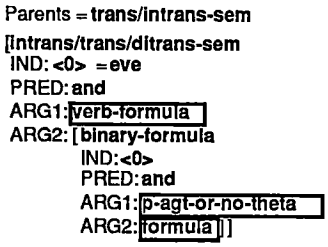
[6]Both raising and pleonastic argument NPs have thematic relation **no-theta**. With raising NPs the second argument of **no-theta** is an object variable, while with pleonastic NPs it is the contentless entity **dummy** (cf. §1). (Either [illegible]instantiation is possible for **no-theta-subj-intrans-xcomp/comp-sem** in (20) where **dummy-**

(21)



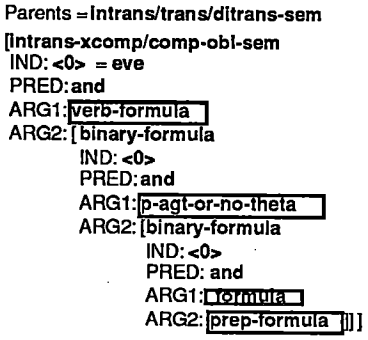
Verbs which take more than two arguments[7] have semantic type **intrans/trans/ditrans-sem**:

(22)



These are further classified according to whether or not there is a direct object role (i.e. a proto-patient or a non-thematic role (**p-pat-or-no-theta**) other than the subject argument). A sizeable group of those which do not encode a direct object correspond to verbs which take a clausal complement and an oblique role represented by the types **formula** and **prep-formula** in (23).

(23)



Two subtypes of **intrans-xcomp/comp-obl-sem** can be distinguished according to whether the subject is thematic (type **p-agt-subj-intrans-xcomp/comp-obl-sem**, e.g. *Jon agreed with Mary to go fishing, Jon promised to me that his car would not break down*) or non-thematic and possibly pleonastic (type **no-theta-subj-intrans-xcomp/comp-obl-sem**, e.g. *John seems to me to have solved the problem, It is hard for John to win the race, It seems to Mary that Bill will not come*):

---

or-obj is the join of **dummy** and **obj**.) This characterization is meant to capture the fact that raising NPs are non-thematic but make reference to some object entity which is assigned a participant role elsewhere in the sentence, while pleonastic NPs are neither thematic nor denoting.

[7] Here, only verbs which take a maximum of three arguments will be discussed.

(24)

```
Parents = intrans-xcomp/comp-obl-sem
[p-agt-subj-intrans-xcomp/comp-obl-sem
 IND: <0> = eve
 PRED: and
 ARG1: verb-formula
 ARG2: [ binary-formula
         IND: <0>
         PRED: and
         ARG1: p-agt-formula
         ARG2: [binary-formula
                IND: <0>
                PRED: and
                ARG1: formula
                ARG2: prep-formula ]]]
```

```
Parents = intrans-xcomp/comp-obl-sem
[no-theta-subj-intrans-xcomp/comp-obl-sem
 IND: <0> = eve
 PRED: and
 ARG1: verb-formula
 ARG2: [ binary-formula
         IND: <0>
         PRED: and
         ARG1: no-theta-formula
         ARG2: [binary-formula
                IND: <0>
                PRED: and
                ARG1: formula
                ARG2: prep-formula ]]]
```

Transitives which take a clausal argument and ditransitives are jointly characterized by the presence of a direct object role, i.e. **p-pat-or-no-theta** in (25).

(25)

```
Parents = intrans/trans/ditrans-sem
[trans/ditrans-sem
 IND: <0> = eve
 PRED: and
 ARG1: verb-formula
 ARG2: [ binary-formula
         IND: <0>
         PRED: and
         ARG1: p-agt-or-no-theta
         ARG2: [formula
                IND: <0>
                PRED: and
                ARG1: p-pat-or-no-theta ]]]
```

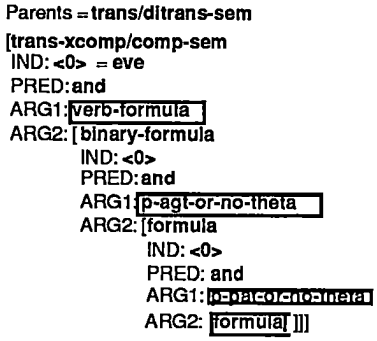
Ditransitives encode a prepositional role in addition to the proto-agent and proto-patient roles as indicated in (26). The index of all argument roles is equal to the eventuality argument variable of the verbal predicate so that the individuals to which they make reference can be understood as participants of the eventuality described by the verb.

(26)

```
Parents = trans/ditrans-sem
[ditrans-sem
 IND: <0> = eve
 PRED: and
 ARG1: [verb-formula
        ARG1: <0> ]
 ARG2: [ binary-formula
         IND: <0>
         PRED: and
         ARG1: [p-agt-formula
                ARG1: <0>]
         ARG2: [binary-formula
                IND: <0>
                PRED: and
                ARG1: [p-pat-formula
                       ARG1: <0>]
                ARG2: [prep-formula
                       ARG1: <0>]]]]
```

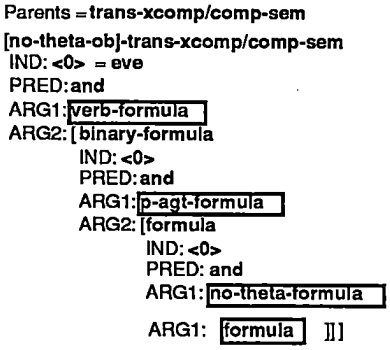
Transitives which take a clausal complement (encoded by the innermost formula in (27)) have semantic type **trans-xcomp/comp-sem**:

(27)



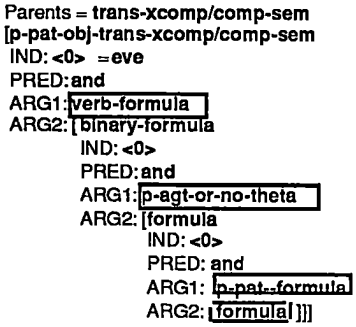
They can be further classified as to whether the subject or object argument is thematic or not. Those which have a non-thematic object (i.e. object raising verbs, e.g. *Bill believes Mary to have left*) have semantic type **no-theta-obj-trans-xcomp/comp-sem**:

(28)



Those which have a thematic object have semantic type **p-pat-obj-trans-xcomp/comp-sem**:

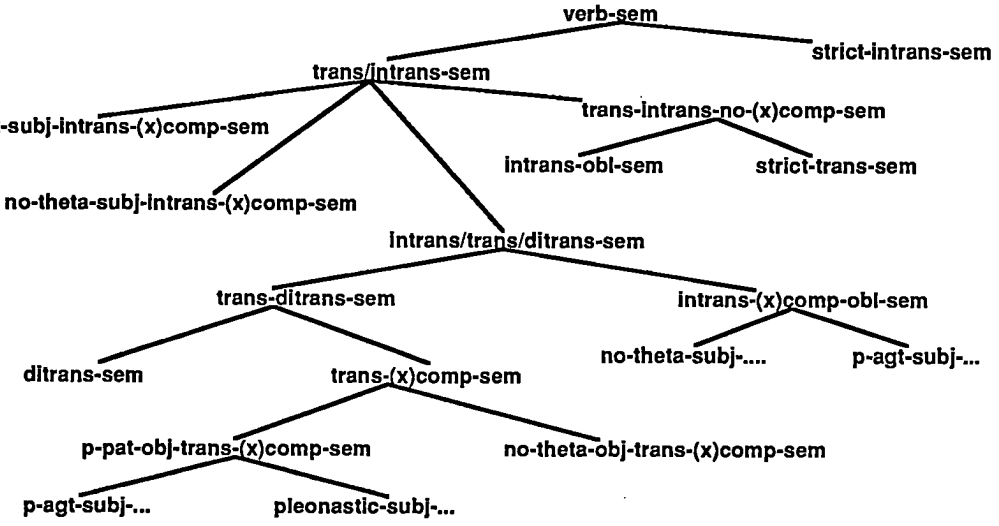
(29)



Verbs of this semantic type can be further classified according to whether the subject is thematic (type **p-agt-subj-p-pat-obj-trans-xcomp/comp-sem**, e.g. *Jon persuaded Mary that the car was worth buying, Bill persuades Mary to leave*) or pleonastic (type **pleonastic-subj-trans-xcomp/comp-sem**, e.g. *It bothers Bill that Mary left, It bothers Bill to leave*).

Here, our discussion of verb semantics comes to a close. A summary of the types described is given in the lattice fragment below.

(30)



3 Verb Syntax

Insofar as all verbs minimally subcategorize for a subject argument, the category type for all members of the verb class can be regarded as a subtype of **complex-cat**:

(31)
```
Parents = complex-cat
[verb-cat
 RESULT:[cat ]
 DIRECTION: direction
 ACTIVE:[sign ]]
```

The top-most partition of verbal category types is established according to whether there is one subcategorized argument or more. The first choice defines the category type of strict intransitive verbs shown in (32).[8]

[8] As shown in (10) and pointed out in footnote 4, verbs are treated as arguments which combine with polymorphic type-raised NPs. Consequently, VP and NP signs are combined into sentences though forward functional application:

```
[sign
 ORTH:[complex-orth
       ORTH1: John
       ORTH2 sleeps ]
 CAT: [sent-ca ]]
```

```
[sign                                    [strict-intrans-sign
 ORTH: John                               ORTH: sleeps
 CAT: [raised-np-cat                      CAT:[strict-intrans-cat
       RESULT: <0> = [cat]                      RESULT: [sent-cat ]
       DIRECTION: <1> = direction               DIRECTION: forward
       ACTIVE: [sign                            ACTIVE: [np-sign ]]
                ORTH: [prth]
                CAT: [complex-cat
                      RESULT: <0>
                      DIRECTION: <1>
                      ACTIVE: [np-sign ]]]]
```

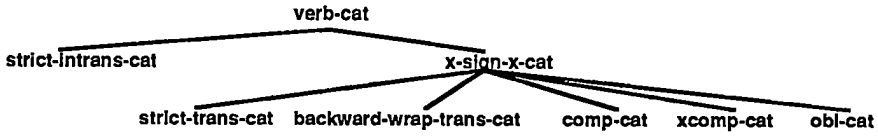
Insofar as the raised NP inherits directionality relative to its active sign from the argument VP sign, the direction value in verb categories of type **strict-intrans-cat** must be of type **forward**.

(32)

```
Parents = verb-cat
[strict-intrans-cat
 RESULT:[sent-cat
          CAT-TYPE:sent
          M-FEATS:[sent-m-feats
                    VFORM: vform
                    COMP-FORM:comp-form
                    REG-MORPH:boolean ]]
 DIRECTION:forward
 ACTIVE: [np-sign
          ORTH:[orth ]
          CAT:[np-cat
               CAT-TYPE:np
               M-FEATS:[nominal-m-feats
                        NOMINAL-FORM:nominal-form
                        CASE: case
                        COUNT:boolean
                        REG-MORPH:boolean
                        AGR:  agr  ]]
          SEM:[theta-formula ]]
```

Verbs which subcategorize for more than one argument have category of type **x-sign-x-cat**. As shown in (33), these can be further classified according to whether the outermost subcategorized sign is a direct object NP (**strict-trans-cat**), an oblique NP (**obl-cat**), a predicative complement (**xcomp-cat**), or a sentential complement (**comp-cat**). A further classificatory parameter is obtained taking into account presence of a direct object for verbs whose active sign is an oblique, sentential or predicative complement (**backwardwrap-trans-cat**).

(33)



Verbs whose outermost subcategorized sign is a direct object NP — i.e. an active noun phrase which is assigned accusative case and either a proto-patient or non-thematic role (see below) — correspond to strict transitives:[9]

(34)

```
Parents = x-sign-x-cat
[strict-trans-cat
 RESULT:[strict-intrans-cat ]
 DIRECTION:backward
 ACTIVE: [dir-obj-np-sign
          ORTH:[orth ]
          CAT:[np-cat
               CAT-TYPE:np
               M-FEATS:[nominal-m-feats
                        NOMINAL-FORM:nominal-form
                        CASE: acc
                        COUNT:boolean
                        REG-MORPH:boolean
                        AGR:[agr ]]
          SEM:[p-pat-or-no-theta ]]
```

Transitive verbs (i.e. verbs which subcategorize for a direct object) whose active sign is either an oblique, sentential or predicative complement have category type **backward-wrap-trans-cat**. As indicated in (35), the direction attribute relative to the direct object NP is specified to be of type **backward-wrap**. Such a specification allows the direct object NP to be satisfied through the 'backward-wrapping' rule (cf. (10)). The oblique/clausal complement is the outermost subcategorized sign according to Dowty's *obliqueness hierarchy* (Dowty 1982a, 1982b), although it is the direct object which follows the verb, e.g. *Bill gave a book to Mary, Mary persuaded Bill to*

[9]Directionality relative to the direct object NP is specified to be **backward** because verbs combine with polymorphic type raised NPs in the grammar implementation used to test the type system (see (10) and footnotes 4,8).

leave, Mary persuaded Bill that Jon is right. This encoding is motivated with respect to a variety of linguistic generalizations concerning control of predicative complements, binding of pronouns and reflexives, the functioning of lexical rules and parameters which govern the distribution of relative clauses across languages (see Pollard & Sag (1987:117-121) and references therein). For example, the subcategorization order imposed by the obliqueness hierarchy makes it possible to provide a generalized treatment of subject and object control for verbs which take a predicative complement (see discussion of category types for control verbs below).

(35)
```
Parents = x-sign-x-cat
[backward-wrap-trans-cat
 RESULT:[complex-cat
          RESULT:strict-intrans-cat
          DIRECTION: backward-wrap
          ACTIVE:dir-ob]-np-sign ]
 DIRECTION:direction
 ACTIVE:sign]
```

Verbs whose outermost subcategorized complement is a sentential sign have category of type **comp-cat**

(36)
```
Parents = x-sign-x-cat
[comp-cat
 RESULT:complex-cat
 DIRECTION:forward
 ACTIVE:[sign
          ORTH:[orth ]
          CAT:sent-cat
          SEM:[sem ]]]
```

Various subtypes of **comp-cat** can be defined according to whether the subcategorized sentence is

- finite with or without complementizer *that*, e.g. *He wished (that) she had called, It bothers Bill that Mary sleeps, Bill tells John that Mary sleeps*
- a *wh*-sentence, e.g. *He wondered whether she would come, He asked Bill whether she would come*
- in base form with complementizer *that*, e.g. *She desires that he leave, they petitioned the government that the law be reconsidered*
- infinitive preceded by a preposition, e.g. *They would prefer for John to do it*

The subtypes of **comp-cat** shown below characterize the first three of these subcategorization patterns.

(37)

Parents = comp-cat
[sfin-comp-cat
RESULT: complex-cat
DIRECTION: forward
ACTIVE: [sign
ORTH: [orth]
CAT: [sent-cat
CAT-TYPE: sent
M-FEATS: [sent-m-feats
VFORM: fin
COMP-FORM: no-comp-or-that-comp
REG-MORPH: boolean]]
SEM: [sem]]]

Parents = comp-cat
[swh-comp-cat
RESULT: complex-cat
DIRECTION: forward
ACTIVE: [sign
ORTH: [orth]
CAT: [sent-cat
CAT-TYPE: sent
M-FEATS: [sent-m-feats
VFORM: fin
COMP-FORM: wh-comp
REG-MORPH: boolean]]
SEM: [sem]]]

Parents = comp-cat
[sbase-comp-cat
RESULT: complex-cat
DIRECTION: forward
ACTIVE: [sign
ORTH: [orth]
CAT: [sent-cat
CAT-TYPE: sent
M-FEATS: [sent-m-feats
VFORM: base
COMP-FORM: that-comp
REG-MORPH: boolean]]
SEM: [sem]]]

Each subtype in turn can be further subdivided according to whether the verb it characterizes is transitive (e.g. *He asked Bill whether she would come*) or intransitive (e.g. *He wondered whether she would come*). The two types **trans-comp-cat** and **intrans-comp-cat** below define these two possibilities.

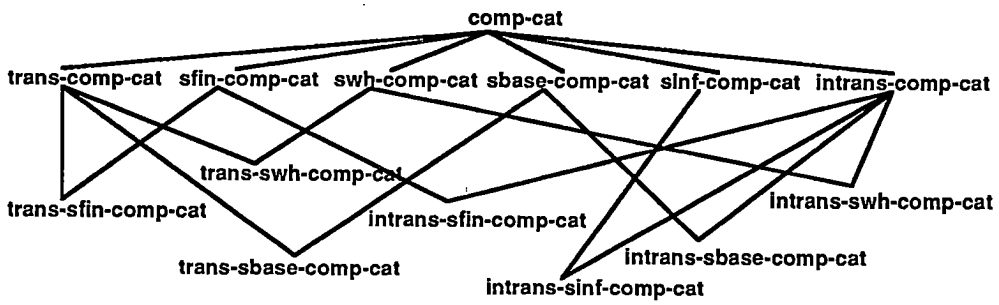
(38)

Parents = backward-wrap-trans-cat comp-cat
[trans-comp-cat
RESULT: [complex-cat
RESULT: strict-intrans-cat
DIRECTION: backward-wrap
ACTIVE: dir-obj-np-sign]
DIRECTION: forward
ACTIVE: [sign
ORTH: [orth]
CAT: sent-cat
SEM: [sem]]]

Parents = comp-cat
[intrans-comp-cat
RESULT: strict-intrans-cat
DIRECTION: forward
ACTIVE: [sign
ORTH: [orth]
CAT: sent-cat
SEM: [sem]]]

Types for transitive and intransitives verbs which subcategorize for a sentential complement which is 'sfin', 'swh' or 'sbase' are defined by intersecting the constraints of each of the types in (37) with those in (38). A summary of category types which inherit from **comp-cat** is given in the lattice fragment below.[10]

(39)



[10]The category of verbs which take an 'sinf' complement (e.g. *They would prefer for John to do it*) is defined as a subtype of **intrans-comp-cat** (cf. **intrans-sfin-comp-cat**) since there are no transitive verbs which exhibit this subcategorization pattern.

Verbs whose outermost active sign is a predicative complement — i.e. a sign whose active category is of type **complex-cat** — have category type **xcomp-cat**.

(40)

```
Parents = x-sign-x-cat
[xcomp-cat
 RESULT: complex-cat
 DIRECTION: forward
 ACTIVE: [sign
          ORTH: [orth ]
          CAT: complex-cat
          SEM: [sem ]]]
```

These can be distinguished as to whether the predicative complement is controlled (e.g. *John wants to leave*) or can have arbitrary reference (*It won't hurt to remind him*); the types **control-cat** and **intrans-vpinf-cat** below express these two possibilities. The term 'control' here is used to describe both equi and raising verbs, e.g. *John wants to leave, John seems to be sad.* No special category types which express the equi-raising distinction are necessary since the contrast is represented semantically: with equi verbs the controlling NP is thematic (i.e. its thematic relation is either **p-agt, p-pat** or **prep**), while with raising verbs the controlling NP is non-thematic (its thematic relation is of type **no-theta**). Control is encoded by equating the individual argument variable of the subcategorized NP which immediately precedes the predicative sign with the individual argument variable associated with the subject of the predicative sign. This encoding provides a specification of both subject and object control (see (42)).

(41)

```
Parents = xcomp-cat
[control-cat
 RESULT: [complex-cat
          ACTIVE: [sign
                   SEM: [binary-formula
                         ARG2: <0> = [sem]]]]
 ACTIVE: [sign
          CAT: [complex-cat
                ACTIVE: [sign
                         SEM: [binary-formula
                               ARG2: <0> ]]]]]
```

```
Parents = xcomp-cat
[intrans-vpinf-cat
 RESULT: [strict-intrans-cat ]
 ACTIVE: [np-sign
          CAT: [strict-intrans-cat
                RESULT: [sent-cat
                         M-FEATS: [sent-m-feats-cat
                                   VFORM: inf ]]]]]
```

As in the case of verbs taking sentential complements, verbs which subcategorize for a predicative complement can be classified taking into account presence of a direct object and morphosyntactic (as well as categorial) properties of the complement. The types **intrans-control-cat** and **trans-control-cat** in (42) provide a categorial characterization of transitive and intransitive control verbs.

(42)

```
Parents = control-cat
[intrans-control-cat
 RESULT: [strict-intrans-cat
          ACTIVE: [np-sign
                   SEM: [theta-formula
                         ARG2: <0> = [sem]]]]
 DIRECTION: forward
 ACTIVE: [sign
          CAT: [complex-cat
                ACTIVE: [np-sign
                         SEM: [theta-formula
                               ARG2: <0> ]]]]]
```

```
Parents = backward-wrap-trans-cat    control-cat
[trans-control-cat
 RESULT: [complex-cat
          RESULT: strict-intrans-cat
          DIRECTION: backward-wrap
          ACTIVE: [dir-obj-np-sign
                   SEM: [p-pat-or-no-theta
                         ARG2: <0> = dummy-or-obj ]]]
 DIRECTION: forward
 ACTIVE: [sign
          CAT: [complex-cat
                ACTIVE: [np-sign
                         SEM: [theta-formula
                               ARG2: <0> ]]]]]
```

According to the contraints inherited from **control-cat**, the controlling NP is the one which immediately precedes the predicative complement in either case. Because of the subcategorization order imposed by the obliqueness hierarchy (i.e. the subject is the innermost complement, a predicative argument the outermost and the direct object stands in between), it follows that the controlling NP corresponds to the subject with intransitive control verbs and to the object with transitive control verbs. The data in (43) provide strong empirical evidence in favour of this

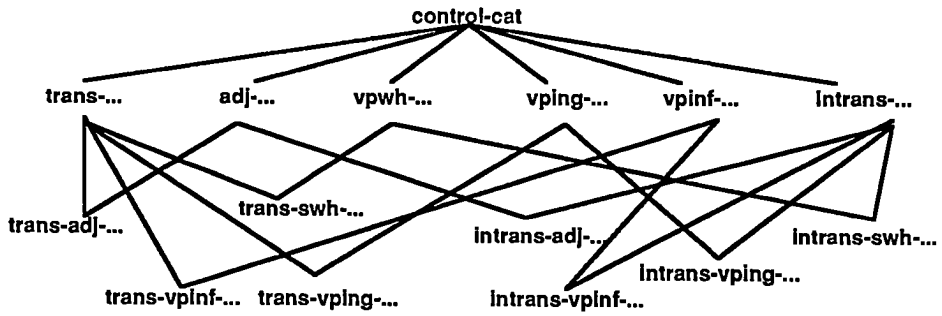
characterization of control patterns.

(43) a Jon wants to leave
Bill loves drinking
Brad sensed what to say to her
Mary feels better

b Mary persuaded Bill to leave
I tried to shame her into voting in the election
Mary believes Jon to be intelligent
He hates people asking him for money
I want this letter (to be) opened right now!
I want the letter ready by tomorrow

A full classification of control verbs is obtained by intersecting the types **intrans-control-cat** and **trans-control-cat** with the category types **vpinf-control-cat, vping-control-cat, vpwh-control-cat, adj-control-cat** which characterize control verbs whose predicative complement is a plain infinitive VP, a gerundive VP, a *wh*-infinitive VP or an adjectival phrase (see (43) for illustrative examples). The lattice fragment below illustrates the resulting hierarchy of control verb types.

(44)



Verbs which subcategorize for an oblique complement (i.e. an NP which is associated with a prepositional role and has case of type **p-case**) have category type **obl-cat**; in the hierarchy for verb categories, this type is only used to define the category of ditransitive verbs:

(45)

```
Parents = x-sign-x-cat
[obl-cat
 RESULT:[complex-cat ]
 DIRECTION:backward
 ACTIVE: [oblique-np-sign
          ORTH:[orth ]
          CAT:[np-cat
               CAT-TYPE:np
               M-FEATS:[nominal-m-feats
                        NOMINAL-FORM:nominal-form
                        CASE: p-case
                        COUNT:boolean
                        REG-MORPH:boolean
                        AGR:agr ]]
          SEM:[prep-formula ]]
```

```
Parents = obl-cat    backward-wrap-trans-cat
[ditrans-cat
 RESULT:[complex-cat
         RESULT:[strict-intrans-cat ]
         DIRECTION:backward-wrap
         ACTIVE:[dir-obj-np-sign ]]
 DIRECTION: backward
 ACTIVE:[oblique-np-sign ]]
```

4 Verb Signs

Verb signs are formed by integrating the semantic and category types described in the previous two sections, and adding orthographic information. The integration of syntactic and semantic information is carried out by coindexing the semantics of subcategorized arguments in the category types with the argument roles in the semantic types. For example, the type for strict intransitive

verb signs is defined by setting the semantic value of the active sign in **strict-intrans-cat** equal to the second argument formula of **strict-intrans-sem**:

(46)
```
Parents = verb-sign
[strict-intrans-sign
 CAT: [strict-intrans-cat
       ACTIVE: [np-sign
                SEM: <0> = [theta-formula ]]]
 SEM: [strict-intrans-sem
       ARG2: <0> ]]
```

With respect the subset of English verbs considered here, there are two additional ways of relating semantic and category types according to whether a verb subcategorizes for two or three arguments:

(47)
```
Parents = verb-sign
[2-complements-verb-sign
 CAT: [complex-cat
       RESULT: [complex-cat
                ACTIVE: [sign
                         SEM: <1> = [sem ]]]
       ACTIVE: [sign
                SEM: <0> = [sem ]]]
 SEM: [verb-sem
       IND: <2> = eve
       PRED: and
       ARG1: [verb-formula ]
       ARG2: [binary-formula
              IND: <2>
              PRED: and
              ARG1: <0>
              ARG2: <1> ]]]
```

```
Parents = verb-sign
[3-complements-verb-sign
 CAT: [complex-cat
       RESULT: [complex-cat
                RESULT: [complex-cat
                         ACTIVE: [sign
                                  SEM: <2> = [sem ]]]
                ACTIVE: [sign
                         SEM: <1> = [sem ]]]
       ACTIVE: [sign
                SEM: <0> = [p-agt-or-no-theta ]]]
 SEM: [intrans/trans/ditrans-sem
       IND: <3> = eve
       PRED: and
       ARG1: [verb-formula ]
       ARG2: [binary-formula
              IND: <3>
              PRED: and
              ARG1: <0>
              ARG2: [binary-formula
                     IND: <3>
                     PRED: and
                     ARG1: <1>
                     ARG2: <2> ]]]]
```

Specific verbs types are defined by either adding constraints to the category and/or semantic attributes of the two types in (47), or merging them with other types. For example, a strict transitive is defined as a **2-complements-verb-sign** whose category and semantics are of type **strict-trans-cat** and **strict-cat-sem** respectively:

(48)
```
Parents = 2-complements-verb-sign
[strict-trans-sign
 CAT: [strict-trans-cat ]
 SEM: [strict-trans-sem ]]
```

By contrast, control verbs which take two arguments (a subject and predicative phrase) are defined as the meet of **2-complements-verb-sign** and **control-verb-sign** which defines a verb whose category is of type **control-cat**:

(49)
```
Parents = verb-sign
[control-verb-sign
 CAT: [control-cat ]]

subj-control-intrans-sign
Parents = control-verb-sign    2-complements-verb-sign
```

Subtypes for the class of control verbs with type **subj-control-intrans-sign** are defined according to whether the subject is thematic or not. Those which have a thematic subject correspond to subject equi verbs, e.g. *Jon wants to leave, Bill loves drinking, Brad sensed what to say to her*, and have semantics of type **p-agt-subj-intrans-xcomp/comp-sem**:

(50)
```
Parents = subj-control-intrans-sign
[subj-equi-intrans-sign
 SEM: [p-agt-subj-intrans-xcomp/comp-sem]]
```

Those which have a non-thematic subject belong to the class of subject raising verbs, e.g. *John seems to sleep, Bill seems sad*, and have semantics of type **no-theta-subj-intrans-xcomp/comp-sem**.

(51)
```
Parents = subj-control-intrans-sign
[subj-raising-intrans-sign
 SEM: [no-theta-subj-intrans-xcomp/comp-sem ] ]
```

A further classification of subject equi and raising intransitives is derived by taking into account morphosyntactic and categorial features of the predicative complement (see description of category types for control verbs in §3, and the list of types for verb signs at the end of this section).

Other verbs whose argument and subcategorization structures are characterized by the type **2-complements-verb-sign** include intransitives which take a sentential complement or oblique object. Intransitives which subcategorize for a sentential complement — e.g. *He wished she had called, He wondered whether she would come, They would prefer for John to do it, She desires that you come at once, It seems that Mary sleeps* — inherit also from **comp-verb-sign** which define the class of verb signs with category type **comp-cat**:

(52)
```
Parents = comp-verb-sign  2-complements-verb-sign
[comp-intrans-sign
 CAT: [comp-cat
       RESULT: [strict-intrans-cat ]
       ACTIVE: [sign
                CAT: sent-cat]]]
```

As in the case of control intransitives, various subtypes of **comp-intrans-sign** can be obtained according to whether the subject is thematic or non-thematic (i.e. pleonastic) taking into account morphosyntactic features of the sentential complement (see description of category types for verbs taking sentential complements in section §3, and the list of types for verb signs at the end of this section).

Intransitives which subcategorize for an oblique object (type **obl-intrans-sign**, e.g. *Bill talks to Mary*) inherit from **2-complements-verb-sign** and **obl-sign** which characterizes verb signs with category type **obl-cat**:

(53)
```
Parents = verb-sign
[obl-sign
 CAT: [obl-cat ]]

Parents = obl-sign   2-complements-verb-sign
[obl-intrans-sign
 CAT: [obl-cat
       RESULT: [strict-intrans-cat ]
       ACTIVE: [np-sign
                CAT: [np-cat
                      M-FEATS: [nominal-m-feats
                                CASE: p-case]]
                SEM: [prep-formula ]]
 SEM: [intrans-obl-sem]]
```

Verbs which takes three arguments inherit the appropriate coindexing relations between their subcategorization and predicate-argument structures from the type **3-complements-verb-sign** shown above. In most cases, their subcategorization frame consists of a subject argument and either

- an oblique object followed by a clausal complement (e.g. *John promised to Mary that Bill will come*),
- a direct object followed by an oblique object (*John gave a book to Mary*), or

- a direct object followed by a clausal complement (*John persuaded Mary to leave*)

Those which take an oblique object inherit also from **obl-sign:**

(54)
obl-xcomp/comp-intrans-or-ditrans-sign
Parents = **obl-sign** **3-complements-verb-sign**

Verbs of this type include intransitive as well as ditransitive verbs. The intransitive ones inherit from the type **obl-xcomp/comp-intrans-sign** which states the presence of a clausal complement:

(55)
Parents = **obl-xcomp/comp-intrans-or-ditrans-sign**
[**obl-xcomp/comp-intrans-sign**
CAT: [**obl-cat**
RESULT:[**complex-cat**
RESULT:[**strict-intrans-cat**]]]]

They can be further classified according to whether the clausal complement is predicative or sentential, taking into account thematic properties of the subject:

- thematic subject with predicative complement, e.g. *John promised (to) Mary to read a book*
- non-thematic subject with predicative complement, e.g. *John seems to me to have solved the problem*
- thematic subject with sentential complement, e.g. *Jon promised to Mary that Bill sleeps*
- pleonastic subject with sentential complement, e.g. *It seems to Bill that John sleeps*

Where a predicative complement occurs, subject control is involved; the type **subj-control-obl-intrans-vpinf-sign** provide a specification for intransitives of this kind which subcategorize for an infinitive VP by stating that the result category is of type **intrans-vpinf-control-cat.**

(56)
Parents = **obl-xcomp/comp-intrans-sign**
[**subj-control-obl-intrans-vpinf-sign**
CAT: [**obl-cat**
RESULT:[**intrans-vpinf-control-cat**]]]

Further specifications concerning thematic properties of the subject argument provide two subtypes of **subj-control-obl-intrans-vpinf-sign** for equi and raising intransitives which subcategorize for an oblique object (e.g. *John promised (to) Mary to read a book, John seems to me to have solved the problem*); the equi type inherits also from **p-agt-subj-obl-xcomp/comp-intrans-sign**, a subtype of **obl-xcomp/comp-intrans-sign** which encodes a thematic subject:

(57)
Parents = **subj-control-obl-intrans-vpinf-sign**
[**subj-raising-obl-intrans-vpinf-sign**
SEM: [**no-theta-subj-intrans-xcomp/comp-obl-sem**]]

Parents = **obl-xcomp/comp-intrans-sign**
[**p-agt-subj-obl-xcomp/comp-intrans-sign**
SEM: [**p-agt-subj-intrans-xcomp/comp-obl-sem**]]

subj-equi-obl-intrans-vpinf-sign
Parents = **p-agt-subj-obl-xcomp/comp-intrans-sign** **subj-control-obl-intrans-vpinf-sign**

Intransitive verbs which take a sentential and oblique complement are defined as subtypes of either **p-agt-subj-obl-xcomp/comp-intrans-sign** or simply **obl-xcomp/comp-intrans-sign** according to whether the subject is pleonastic (type **extrap-obl-comp-intrans-Sfin-sign**, e.g. *It seems to Bill that John sleeps*) or thematic (type **obl-comp-intrans-Sfin-sign**, e.g. *John promised to Mary that Bill sleeps*).

(58)
Parents = **obl-xcomp/comp-intrans-sign**
[**extrap-obl-comp-intrans-sfin-sign**
CAT: [**obl-cat**
RESULT:[**sfin-comp-cat**
RESULT: sent-cat
ACTIVE: dummy-np-sign]]
SEM: no-theta-subj-intrans-xcomp/comp-obl-sem]

Parents = **p-agt-subj-obl-xcomp/comp-intrans-sign**
[**obl-comp-intrans-sfin-sign**
CAT: [**obl-cat**
RESULT:[**sfin-comp-cat**
RESULT: sent-cat
ACTIVE: [**np-sign**
SEM: p-agt-formula]]]]

Verbs which subcategorize for a direct object and either a clausal or oblique complement have category type **backward-wrap-trans-cat**:

(59)
```
Parents = 3-complements-verb-sign
[trans-xcomp/comp/obl-sign
 CAT: [backward-wrap-trans-cat ]]
```

As discussed in §1 and §3, such a category specification allows the direct object to be the first argument to combine with the verb through 'backward-wrapping' even though the oblique/clausal complement is the outermost subcategorized sign according to Dowty's obliqueness hierarchy. Verbs of this type include ditransitives and transitives which take a clausal complement. Ditransitives inherit also from **obl-xcomp/comp-intrans-or-ditrans-sign** (cf. (54)) which provides appropriate specifications for the oblique argument:

(60)
```
Parents = obl-xcomp/comp-intrans-or-ditrans-sign    trans-xcomp/comp/obl-sign
[ditrans-sign
 SEM: [ditrans-sem ]]
```

Transitives which subcategorize for a clausal complement can be classified according to whether the clausal complement is predicative or sentential, taking into account thematic properties of the subject and direct object arguments. Those which take a predicative complement inherit from both **trans-xcomp/comp/obl-sign** and **control-verb-sign**:

(61)
```
control-trans-sign
Parents = trans-xcomp/comp/obl-sign    control-verb-sign
```

They are further classified according to whether the object is thematic (type **equi-trans-sign**, e.g. *Mary persuades Bill to read a book, I tried to shame her into voting in the election*) or non-thematic (type **raising-trans-sign**, e.g. *Mary believes Jon to be intelligent, He hates people asking him for money*):

(62)
```
Parents = control-trans-sign                 Parents = control-trans-sign
[equi-trans-sign                             [raising-trans-sign
 SEM: [p-pat-obj-trans-xcomp/comp-sem ]]      SEM: [no-theta-obj-trans-xcomp/comp-sem ]]
```

A first classification of transitive equi verbs is made according to whether the subject is pleonastic (type **extrap-equi-trans-vpinf-sign**, e.g. *It pleases Bill to read a book*) or thematic (type **p-agt-subj-equi-trans-sign**, e.g. *Mary persuades Bill to read a book*).

(63)
```
Parents = equi-trans-sign
[extrap-equi-trans-vpinf-sign
 CAT: [trans-vpinf-control-cat
       RESULT: [complex-cat
                RESULT: [sent-cat
                         ACTIVE: [dummy-np-sign ]]]]
 SEM: [pleonastic-subj-trans-xcomp/comp-sem]]

Parents = control-trans-sign
[p-agt-subj-equi-trans-sign
 SEM: [p-agt-subj-p-pat-obj-trans-xcomp/comp-sem]]
```

Various subtypes of 'p-agt-subj' equi and raising transitives are defined according to morphosyntactic and categorial features of the clausal complement (see description of category types for control verbs in §3, and the list of types for verb signs below).

Transitives which subcategorize for a (finite) sentential complement inherit both from **comp-verb-sign** and **trans-xcomp/comp/obl-sign**:

(64)
```
Parents = comp-verb-sign    trans-xcomp/comp/obl-sign
[comp-trans-sign
 CAT: [sfin-comp-cat ]]
```

Two subtypes can be distinguished according to whether the subject is thematic (type **reg-comp-trans-sign**, e.g. *Bill tells John that Mary sleeps*), or pleonastic (type **extrap-comp-trans-sign**, e.g. *It bothers Bill that Mary sleeps*:

(65)

```
Parents = comp-trans-sign
[reg-comp-trans-sign
 SEM: [p-agt-subj-p-pat-obj-trans-xcomp/comp-sem]]
```

```
Parents = comp-trans-sign
[extrap-comp-trans-sign
 CAT: [trans-sfin-comp-cat
       RESULT: [complex-cat
                RESULT: [sent-cat
                         ACTIVE: [dummy-np-sign ]]]
       ACTIVE: [dir-obj-np-sign
                CAT: [np-cat
                      M-FEATS: [nominal-m-feats
                                COMP-FORM: that-comp ]]]]
 SEM: [pleonastic-subj-trans-xcomp/comp-sem ] ]
```

To conclude the description of verb signs, here follows a list of all types defined along with illustrative examples.

- **strict-intrans-sign**, e.g. *John sleeps*
- **strict-trans-sign**, e.g. *John reads a book*
- **subj-equi-intrans-vpinf-sign**, e.g. *John wants to sleep*
- **subj-equi-intrans-vping-sign**, e.g. *John loves sleeping*
- **subj-equi-intrans-adj-sign**, e.g. *John feels sad*
- **subj-raising-intrans-vpinf-sign**, e.g. *John seems to sleep*
- **subj-raising-intrans-adj-sign**, e.g. *John seems sad*
- **pleonastic-subj-intrans-vpinf-sign**, e.g. *It hurts to feel sad*
- **equi-trans-vpinf-sign**, e.g. *John persuades Mary to sleep*
- **equi-trans-vping-sign**, e.g. *John hates people asking him for money*
- **equi-trans-adj-sign**, e.g. *John found Mary sad*
- **raising-trans-vpinf-sign**, e.g. *John believes Mary to sleep*
- **raising-trans-vping-sign**, e.g. *John hates Mary sleeping*
- **raising-trans-adj-sign**, e.g. *John wants the book ready*
- **extrap-equi-trans-vpinf-sign**, e.g. *It pleases Mary to sleep*
- **reg-comp-intrans-sfin-sign**, e.g. *John thinks (that) Mary sleeps*
- **reg-comp-intrans-swh-sign**, e.g. *Mary wonders whether John sleeps*
- **reg-comp-intrans-sinf-sign**, e.g. *Mary prefers for John to sleep*
- **reg-comp-intrans-sbase-sign**, e.g. *Mary desires that John sleep*
- **extrap-comp-intrans-sign**, e.g. *It seems that Mary sleeps*
- **extrap-comp-trans-sign**, e.g. *It bothers Bill that Mary sleeps*
- **reg-comp-trans-sign**, e.g. *Bill tells John that Mary sleeps*
- **obl-intrans-sign**, e.g. *John talks to Bill*
- **ditrans-sign**, e.g. *John gives a book to Mary*
- **subj-equi-obl-intrans-vpinf-sign**, e.g. *John promises Mary to sleep*
- **subj-raising-obl-intrans-vpinf-sign**, e.g. *John seems to Mary to sleep*
- **obl-comp-intrans-sfin-sign**, e.g. *John promises Mary that Bill sleeps*
- **extrap-obl-comp-intrans-Sfin-sign**, e.g. *It seems to Bill that John sleeps*
- **subj-equi-intrans-vpwh-sign**, e.g. *Brad sensed what to say to her*

5 Relating Verb-Sign Types to Word-Sense Templates

The verb types described in §4 can be related to lexical templates encoding word-sense specific information to provide a specification of subcategorization and argument structure properties for a large number of English verbs. To assess the feasibility of such an enterprise, a case study was run with reference to word-sense templates for psychological verbs. These were derived automatically using as sources tape versions of the Longman Dictionary of Contemporary English (LDOCE) and Longman Lexicon (LLex) mounted on the Lexical Data Base (LDB) developed by the ACQUILEX group in Cambridge (ACQUILEX Deliv. No. 2.3.3(a)). First, word senses relative to psychological verbs were chosen and classified into semantic subtypes using the set codes in LLex which index verbs expressing feelings, emotions attitudes and sensations (i.e. sets with letter code 'F'). Psychological predicates were classified according to the following parameters:

- affect is positive (*admire, delight*), neutral (*experience, interest*) or negative (*fear, scare*)
- stimulus argument is realized as object and experiencer as subject, e.g. *admire, experience, fear*
- stimulus argument is realized as subject and experiencer as object, e.g. *delight , interest, scare*

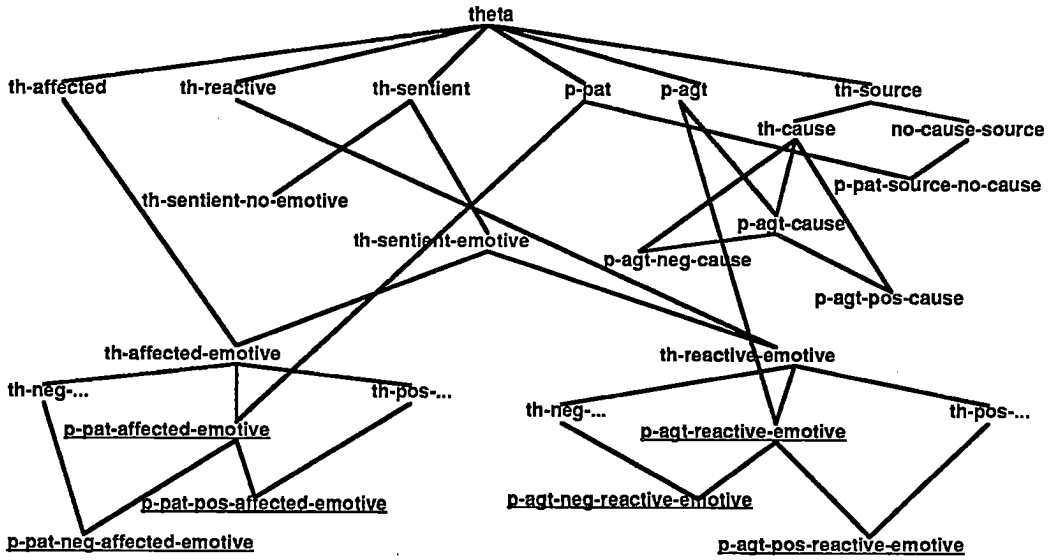
Psychological verbs with experiencer subjects were classified as 'non-causative'; the stimulus of these verbs was considered to be a 'source' to which the experiencer 'reacts emotively'. Psychological verbs with stimulus subjects were instead classified as involving 'causation'; accordingly, the stimulus argument was consided as a 'causative source' by which the experiencer participant is 'emotively affected'. This analysis combines insights from several approaches to the classification of psychological predicates (see Jackendoff (1990), Levin (1989), and references therein). Six subtypes of psychological verbs were thus defined according to semantic properties of the stimulus and experiencer arguments:

(66)

STIMULUS ARGUMENT	EXPERIENCER ARGUMENT	ILLUSTRATIVE EXAMPLE
non-causative source	neutral, reactive, emotive	*experience*
non-causative source	positive, reactive, emotive	*admire*
non-causative source	negative, reactive, emotive	*fear*
neutral causative source	neutral, affected, emotive	*interest*
positive causative source	positive, affected, emotive	*delight*
negative causative source	negative, affected, emotive	*scare*

The properties used in this classification were then used to define a hierarchy of thematic sorts, as shown in the lattice fragment below where the underlined types correspond to the proto-agent and proto-patient role types used to distinguish the six semantic varieties of psychological predicates.

(67)



To double-check the results of this classification and add information concerning subcategorization and further semantic restrictions on argument roles, a derived dictionary was created where the relevant LLex verb senses were linked to corresponding verb senses in LDOCE using the Dictionary Correlation Kit developed by the ACQUILEX group in Cambridge (Poznanski 1991; Sanfilippo & Poznanski). With such a derived dictionary, LDB queries could be run which combined information from LLex, LDOCE and two other dictionaries derived from LDOCE: LDOCE_Inter and LDOCE_Sem. LDOCE_Inter was derived by a translation program which mapped the grammar codes of LDOCE entries into a theoretically uncommitted intermediate representation (Boguraev & Briscoe 1989; Carroll & Grover 1989). LDOCE_Sem was derived by extracting genus information from dictionary definitions in LDOCE (Alshawi 1989; Vossen 1990). The availability of combined dictionary sources made it possible to create word-sense templates from LDB queries which in addition to providing thematic restrictions on the stimulus and experiencer roles encoded

- detailed information about subcategorization patterns (LDOCE_Inter)
- information about diathesis alternations (LDOCE_Inter)
- selectional restrictions (*box codes* in LDOCE)

The recovery of detailed subcategorization information form LDOCE_Inter was highly instrumental in assigning verb types to verb-sense templates. For example, whenever a verb sense in LDOCE_Inter was associated with the information in (68) a verb template of type **strict-trans-sign** was automatically created.

(68) ((Cat V) (Takes NP NP) ...)

Usign this technique, verb types were assigned to some 200 verb senses; this assignment yielded a verb lexicon of 431 entries. The templates below provide illustrative examples for the six semantic varieties of psychological verbs taken into account.[11]

```
fear L_2_1_3
STRICT-TRANS-SIGN
< cat : result : result : m-feats : diathesis > = NO-INFO
< cat : result : active : sem : arg2 > = (E-ANIMAL E-HUMAN)
```

[11]The alternation type **unaccusative** marks verbs which may undergo the *unspecified object deletion rule* (e.g. *a book which is certain to delight them* vs. *a book which is certain to delight*), while **ergative** marks verbs which are amenable to the *causative-inchoative* or *middle-inchoative* alternation (e.g. *a man who scares Bill* vs. *a man who doesn't scare (easily)*).

```
< cat : result : active : sem : pred > = P-AGT-NEG-REACTIVE-EMOTIVE
< cat : active : sem : pred > = P-PAT-SOURCE-NO-CAUSE
< lex-sign sense-id : sense-id dictionary > = "LDOCE"
< lex-sign sense-id : sense-id ldb-entry-no > = "12872"
< lex-sign sense-id : sense-id sense-no > = "1".
experience L_2_0
STRICT-TRANS-SIGN
< cat : result : result : m-feats : diathesis > = NO-INFO
< cat : result : active : sem : arg2 > = (E-ANIMAL E-HUMAN)
< cat : active : sem : arg2 > = E-ABSTRACT
< cat : result : active : sem : pred > = P-AGT-REACTIVE-EMOTIVE
< cat : active : sem : pred > = P-PAT-SOURCE-NO-CAUSE
< lex-sign sense-id : sense-id dictionary > = "LDOCE"
< lex-sign sense-id : sense-id ldb-entry-no > = "12364"
< lex-sign sense-id : sense-id sense-no > = "0".
admire L_0_1_1
STRICT-TRANS-SIGN
< cat : result : result : m-feats : diathesis > = NO-INFO
< cat : result : active : sem : arg2 > = E-HUMAN
< cat : active : sem : arg2 > = OBJ
< cat : result : active : sem : pred > = P-AGT-POS-REACTIVE-EMOTIVE
< cat : active : sem : pred > = P-PAT-SOURCE-NO-CAUSE
< lex-sign sense-id : sense-id dictionary > = "LDOCE"
< lex-sign sense-id : sense-id ldb-entry-no > = "355"
< lex-sign sense-id : sense-id sense-no > = "1".
scare L_1_1
STRICT-TRANS-SIGN
< cat : result : result : m-feats : diathesis > = ERGATIVE
< cat : result : active : sem : arg2 > = OBJ
< cat : active : sem : arg2 > = E-HUMAN
< cat : result : active : sem : pred > = P-AGT-NEG-CAUSE
< cat : active : sem : pred > = P-PAT-NEG-AFFECTED-EMOTIVE
< lex-sign sense-id : sense-id dictionary > = "LDOCE"
< lex-sign sense-id : sense-id ldb-entry-no > = "31745"
< lex-sign sense-id : sense-id sense-no > = "1".
interest L_2_1_1
STRICT-TRANS-SIGN
< cat : result : result : m-feats : diathesis > = NO-INFO
< cat : result : active : sem : arg2 > = OBJ
< cat : active : sem : arg2 > = E-HUMAN
< cat : result : active : sem : pred > = P-AGT-CAUSE
< cat : active : sem : pred > = P-PAT-AFFECTED-EMOTIVE
< lex-sign sense-id : sense-id dictionary > = "LDOCE"
< lex-sign sense-id : sense-id ldb-entry-no > = "18589"
< lex-sign sense-id : sense-id sense-no > = "1".
delight L_2_1
STRICT-TRANS-SIGN
< cat : result : result : m-feats : diathesis > = UNACCUSATIVE
< cat : result : active : sem : arg2 > = OBJ
< cat : active : sem : arg2 > = E-HUMAN
< cat : result : active : sem : pred > = P-AGT-POS-CAUSE
< cat : active : sem : pred > = P-PAT-POS-AFFECTED-EMOTIVE
< lex-sign sense-id : sense-id dictionary > = "LDOCE"
< lex-sign sense-id : sense-id ldb-entry-no > = "9335"
< lex-sign sense-id : sense-id sense-no > = "1".
```

When loaded into the LKB, the templates above will exspand into sign-based representations of verb senses as shown in (69) for the transitive verb *experience*; such representations will arise from integrating word-specific information provided by verb-sense templates with the information encoded by the verb types (e.g. STRICT-TRANS-SIGN in the examples above).

(69)

```
[strict-trans-sign
 ORTH:experience
 CAT:[strict-trans-cat]
 SEM:[strict-trans-sem
      IND:<0> =eve
      PRED:and
      ARG1:[verb-formula]
      ARG2:[binary-formula
            IND:<0>
            PRED:and
            ARG1:<1> =[p-agt-formula
                       IND:<0>
                       PRED:p-agt-reactive-emotive
                       ARG1:<0>
                       ARG2:(e-animal e-human) ]
            ARG2:<2> =[p-pat-formula
                       IND:<0>
                       PRED:p-pat-source-no-cause
                       ARG1:<0>
                       ARG2:e-abstract ]]]
 SENSE-ID:[sense-id]]
```

6 Conclusion

Over the last few years, the utilization of machine readable dictionaries in compiling large scale lexicons for Natural Language Processing systems has awakened the interest of an increasing number of researchers. As a result of this trend, the need has arisen to develop large lexical knowledge bases where the information extracted from machine readable dictionaries can be suitably stored to maximize portability and reusability. Current research in this area has shown that the use of inheritance systems based on typed unification provides a knowledge representation language which satisfy this need with computational efficiecy and formal adequacy. In keeping with these recent developments, the work described in this paper gives a concrete example of how properties of verb forms extracted from machine readable dictionaries can be encoded within a Lexical Knowledge Base which uses a typed system of unification as representation language. The system described consists of a network of information structures with links defined in terms of inheritance. These information structures provide efficient means to represent detailed information about syntactic and semantic properties of verb forms in a format which can be tested for appropriatedness in a parsing context and can be easily tailored to suit requirements of specific NLP systems.

Acknowledgements

In developing the study described in this paper, I have benefitted from discussions with Ted Briscoe, John Carroll, Ann Copestake and Valeria de Paiva.

References

ACQUILEX Deliv. NO. 2.3.3(a) (1990) *Lexical Database System: User Manual*, ESPRIT BRA-3030

Alshawi, H. (1989) Analysing the Dictionary Definitions. In Boguraev, B. & Briscoe, T. (eds.) *Computational Lexicography for Natural Language Processing.* Longman, London.

Boguraev, B. & Briscoe, T. (1989) Utilising the LDOCE Grammar Codes. In Boguraev, B. & Briscoe, T. (eds.) *Computational Lexicography for Natural Language Processing.* Longman, London.

Carroll, J. & Grover, C. (1989) The Derivation of a Large Computational Lexicon for English from LDOCE. In Boguraev, B. & Briscoe, T. (eds.) *Computational Lexicography for Natural Language Processing.* Longman, London.

Copestake, A. (1991) The LKB: a System for Representing Lexical Information Extracted from Machine-Readable Dictionaries. This volume.

Dowty, D. (1982) Grammatical Relations and Montague Grammar. In Jacobson, P. and Pullum, G. K. (eds.) *The Nature of Syntactic Representation*, pp. 79-130. D. Reidel, Dordrecht.

Dowty, D. (1982) More on the categorial analysis of grammatical relations. In Zaenen, A. (ed.) *Subjects and Other Subjects: Proceedings of the Harvard Conference on Grammatical Relations*, Bloomington, Indiana, 1982. Also in Ohio State University Working Papers in Linguistics 26 102-133.

Dowty, David (1987) *Thematic Proto Roles, Subject Selection, and Lexical Semantic Defaults.* LSA Colloquium paper.

Flickinger, D. (1987) *Lexical Rules in the Hierarchical Lexicon* PhD Thesis, Stanford University.

Flickinger, D., Pollard, C. and Wasow, T. (1985) Structure Sharing in Lexical Representation. In Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics.

Jackendoff, R. (1990) *Semantic Structures.* MIT Press, Cambridge, Mass.

Levin, B. (1989) *Towards a Lexical Organization of English Verbs.* Ms., Dept. of Linguistics, Northwestern University.

de Paiva, V. (1991) Types and Constraints in the LKB. This volume.

Parsons, Terence (1980) Modifiers and Quantifiers in Natural Language. *Canadian Journal of Philosophy*, supplementary Volume VI, 29-60.

Parsons, Terence (1990) *Events in the Semantics of English: a Study in Subatomic Semantics.* MIT press, Cambridge, Mass.

Pollard, Carl and Sag, Ivan (1987) *An Information-Based Approach to Syntax and Semantics: Volume 1 Fundamenals.* CSLI Lecture Notes 13, Stanford CA.

Poznanski, V. (1991) *DCK: a Dictionary Correlation Kit*, ms., Computer Laboratory, University of Cambridge.

Sanfilippo, A. (1990) *Grammatical Relations, Thematic Roles and Verb Semantics.* PhD thesis, Centre for Cognitive Science, University of Edinburgh, Scotland.

Sanfilippo, A. & Poznanski, V. (1991) The Acquisition of Lexical Knowledge from Combined Machine-Readable Dictionary Sources, ms., Computer Laboratory, University of Cambridge.

Vossen, P. (1990) *A Parser-Grammar for the Meaning Descriptions of LDOCE*, Links Project Technical Report 300-169-007, Amsterdam University.

Zeevat, Henk, Klein, Ewan and Calder, Jo (1987) An Introduction to Unification Categorial Grammar. In Haddock, J. Nick, Klein, Ewan and Morrill, Glyn (eds.) *Edinburgh Working Papers in Cognitive Science, Volume 1: Categorial Grammar, Unification Grammar, and Parsing.*

Defaults in the LRL

Ann Copestake
Computer Laboratory, University of Cambridge

The typed feature structure formalism described in the previous papers is augmented with a default inheritance system. In this paper we first introduce this informally and illustrate the sort of taxonomic data that our system was designed to represent. We then discuss the formal issues involved in introducing defaults into our representation language (LRL). As will become clear, there are several difficulties with defining a usable default system, and this work should be regarded as considerably more speculative than that on the type system.

1 Default inheritance in the LRL

Feature structures may be specified as inheriting by default from one or more other (well-formed) feature structures which we refer to in this context as *psorts*. Psorts may correspond to (parts of) lexical entries or be specially defined. Since psorts may themselves inherit information, default inheritance (notated by $<$, "inherits from") in effect operates over a hierarchy of psorts. We prohibit cycles in the inheritance ordering. Inheritance order must correspond to the type hierarchy order.

if p_1 and p_2 are psorts $p_1 < p_2 \Rightarrow type-of(p_1) \sqsubseteq type-of(p_2)$

The typing system thus restricts default inheritance essentially to the filling in of values for features which are defined by the type system. Default inheritance is implemented by a version of default unification, which is discussed in detail below.

The ordering on the psort hierarchy gives us an ordering on defaults. So for example, assume that the following is part of the lexical entry for BOOK_L_1_1 (ie BOOK_L_1_1 is the name of the psort, where the code L_1_1 refers to the sense number in the machine readable dictionary).

BOOK_L_1_1

$$\left[\begin{array}{l}\textbf{lex-noun-sign} \\ \text{RQS} = \left[\begin{array}{l}\textbf{artifact\_physical} \\ \text{TELIC} = \left[\begin{array}{l}\textbf{verb-sem} \\ \text{PRED} = \textbf{read\_L\_1\_1}\end{array}\right] \\ \text{PHYSICAL-STATE} = \textbf{solid\_a}\end{array}\right]\end{array}\right]$$

This feature structure shows part of the relatively rich semantic structure which we encode for nouns based on the notion of *qualia structure*, described, for example, in Pustejovsky(1989). We refer to this as the *relativised qualia structure* (RQS). The feature TELIC is used to provide a slot for the semantics of the verb sense which is associated with the purpose of an entity ("reading" in this case). (The way in which such a representation may be used in the treatment of logical metonymy was described in Briscoe et al (1990).) Other features (for example PHYSICAL-STATE) are used to encode information which is useful for applications such as sense-disambiguation.

The following path specifications make the lexical entries defined inherit from BOOK_L_1_1:

```
autobiography_L_0_0      <rqs> < book_L_1_1 <rqs>
dictionary_L_0_1         <rqs> < book_L_1_1 <rqs>
                         <rqs : telic : pred > = refer_to_L_0_2
lexicon_L_0_0            <rqs> < dictionary <rqs>
```

AUTOBIOGRAPHY_L_0_1 would thus have the same values as BOOK_L_1_1 for both TELIC and PHYSICAL-STATE. DICTIONARY_L_0_1 will inherit the value **solid_a** for the feature PHYSICAL-STATE but the value of TELIC (**refer_to_L_0_2**) overrides that which would be inherited from BOOK_L_1_ 1 (**read_L_1_1**). LEXICON_L_0_0 inherits its value for the telic role from DICTIONARY_L_0_1 rather than from BOOK_L_1_1.

LEXICON_L_0_0

$$\left[\begin{array}{l}\textbf{lex-noun-sign} \\ \text{RQS} = \left[\begin{array}{l}\textbf{artifact\_physical} \\ \text{TELIC} = \left[\begin{array}{l}\textbf{verb-sem} \\ \text{PRED} = \textbf{refer\_to\_L\_0\_2}\end{array}\right] \\ \text{PHYSICAL-STATE} = \textbf{solid\_a}\end{array}\right]\end{array}\right]$$

Multiple default inheritance is allowed but is restricted to the case where the information from the parent psorts does not conflict. This is enforced by unifying all (fully-expanded) immediate parent psorts before default unifying the result with the daughter psort. The type restriction on default inheritance means that all the psorts must have compatible types and the type of the daughter must be the meet of those types. We define inheritance to operate top-down; that is a psort will be fully expanded with inherited information before it is used for default inheritance.

We also allow non-default inheritance from psorts, implemented by ordinary unification. This is a relatively recent addition to the LRL, prompted partly by issues in the representation of the multilingual translation links, which are not discussed here. It also seemed to be desirable in the representation of qualia structure, in order to allow the telic role of a noun to be specified directly in terms of a verb sense, without allowing other information in that lexical entry to conflict. Thus the entry for dictionary above would actually specify:

```
<rqs : telic > == refer_to_L_0_2 < sem >
```

where == indicates non-default inheritance.

2 Default inheritance and taxonomies

Although introducing psorts as well as types may seem unnecessarily complex, there seem to be compelling reasons for doing so for this application, where we wish to use taxonomic information extracted from MRDs to structure the lexicon. For example, the inheritance relationship defined above is the result of analysing the following definitions of *autobiography*, *dictionary* and *lexicon* in LDOCE.

autobiography 1 a book written by oneself about one's own life.

dictionary 1 a book that gives a list of words in alphabetical order with their pronunciations and meanings.

lexicon a dictionary esp. of an ancient language

The taxonomies are essentially the network produced by linking the word sense defined with the word sense used as the *genus term* in its definition, when the relationship involved is essentially an "IS-A" relationship. The issues of identifying the genus sense, and the meaning of "IS-A" relationship in this context, are discussed in Copestake(1990).

The type hierarchy is not a suitable way for representing taxonomic inheritance for several reasons. Perhaps the most important is that taxonomically inherited information is defeasible, but typing and defaults are incompatible notions. Types are needed to enforce an organisation on the lexicon — if this can be overridden it is useless. Furthermore the type system is taken to be complete, and various conditions are imposed on it, such as the greatest lower bound condition, which ensure that deterministic classification is possible. Taxonomies extracted from dictionaries will not be complete in this sense, and will not meet these conditions. Intuitively we would expect to be able to classify lexical entries into categories such as **person**, **artifact** and so on, and to be able to state that all **creatures** are either **persons** or **animals**, since in effect this is how we define those types. But we would not expect to be able to use the finer-grained, automatically acquired information in this way; we will never extract all possible categories of horse for example. (Cf Brachman et al's(1985) distinction between *terminological* and *assertional* knowledge.)

In implementational terms, using the type hierarchy to provide the fine-grain of inheritance possible with taxonomic information would be very difficult. A type scheme should be relatively static; any alterations may affect a large amount of data and checking that the scheme as a whole is still consistent is a non-trivial process. Because the inheritance hierarchies are derived from taxonomies and thus are derived semi-automatically from MRDs, they will contain errors and it is important that these can be corrected easily.

In practice, deciding whether to make use of the type mechanism or the psort mechanism has been relatively straightforward. If we wish to define a feature which is particular to some group of lexical entries we have to introduce a type, if we wish to specify the value of a feature, especially if the information might be defeasible, we use a psort.

Several of the decisions involved in designing the default inheritance system were thus influenced by the application. The condition that the default inheritance ordering reflects the type ordering was partly motivated by the desire to be able to provide an RQS type for lexical entries on the basis of taxonomic data alone. (However it also seems intuitively reasonable as a way of restricting default inheritance; without some such restriction it is difficult to make any substantive claims when default inheritance is used to model some linguistic phenomenon.)

Since we have to cope with errors in extraction of information from MRDs, and with the lexicographers' original mistakes, we adopted the conservative condition

that information inherited from multiple parents has to be consistent. This is discussed in more detail below. However our consistency condition seems to be met fairly naturally by the data. Taxonomies extracted from MRDs are in general tree-structured (once sense-disambiguation has been performed); there do not tend to be many examples of genuine conjunction in the genus term, for example. Multiple inheritance is mainly needed for cross-classification; artifacts for example may be defined principally in terms of their form or in terms of their function, but here different sets of features are specified, corresponding to different parts of the qualia structure, so the information is consistent.

3 Default unification of untyped feature structures

Default unification is defined so that when a non-default feature structure is unified with a default feature structure only values in the default structure which do not conflict with values in the non-default structure are incorporated. In our case the default feature structure will be the feature structure associated with the psort and the non-default feature structure will be that associated with the inheritor. (The result of this may later be treated as the default in another default unification operation.) We will use $A \overset{<}{\sqcap} B$ to indicate default unification where A is non-default. In this section we consider the definition of default unification of untyped feature structures; typed feature structures are considered in the following section.

We would like default unification to have the following properties.

1. $A \overset{<}{\sqcap} B \sqsubseteq A$
   Default unification adds information monotonically to the non-default.
   Clearly it should not be possible to remove non-default information, and all definitions of default unification of which we are aware do meet this criterion.

2. if $A \sqcap B \neq \bot$ then $A \sqcap B = A \overset{<}{\sqcap} B$
   Default unification behaves like ordinary unification in the cases where ordinary unification would succeed.
   Intuitively ordinary unification should correspond to the case where the default feature structure is totally compatible with the non-default structure.

3. $A \overset{<}{\sqcap} B \neq \bot$
   Default unification never fails
   This seems highly desirable as a distinguishing feature between default unification and ordinary unification.

4. Default unification returns a single result, deterministically.
   We obviously do not want to introduce non-determinism into the system. Multiple results are awkward both from the implementation and the usability viewpoint.

It is in general necessary that default unification be implementable with reasonable efficiency and it is highly desirable that it give results which are intuitively plausible for our users.

The examples of default unification given in section ?? are unproblematic. However there are cases where there are conflicts between parts of the default information, because of reentrancy in the default or in the non-default feature structure. This is discussed in detail in Carpenter(1991, this volume); here we wish to extend that discussion slightly in order to illustrate the varieties of default unification which have been proposed and to describe our own variant. In many ways we regard Carpenter's definition of skeptical default unification as the paradigm; it meets all the conditions enumerated above and has a definition which can be simply paraphrased as "incorporate the maximal consistent information from the default". Unfortunately it appears that it cannot be implemented efficiently (Carpenter, personal communication, see also below).

The following examples illustrate the differences in behaviour between the definitions that have been proposed, by Bouma(1990), Carpenter, Calder(1991), Russell et al (1991, this volume) and by ourselves.

(1) $$\begin{bmatrix} F = a \\ G = \top \end{bmatrix} \overset{\leq}{\sqcap} \begin{bmatrix} F = b \\ G = c \end{bmatrix} = \begin{bmatrix} F = a \\ G = c \end{bmatrix}$$

This is the simplest case of default unification; the conflicting information in the default is ignored, but the non-conflicting is incorporated. (Although we are discussing untyped feature structures, we use $\top$ to indicate an unspecified value.)

(2) $$\begin{bmatrix} F = \boxed{1} \\ G = \boxed{1} \end{bmatrix} \overset{\leq}{\sqcap} \begin{bmatrix} F = a \\ G = \top \end{bmatrix} = \begin{bmatrix} F = \boxed{1} \\ G = \boxed{1} \end{bmatrix}$$ Bouma

$$= \begin{bmatrix} F = \boxed{1}\ a \\ G = \boxed{1} \end{bmatrix}$$ other definitions

We include this example to illustrate that Bouma's definition of default unification does not meet our second criterion, since it gives a different result from ordinary unification. (In practice this behaviour is presumably not apparent since Bouma suggests that ordinary unification is attempted before default unification.)

(3) $$\begin{bmatrix} F = \boxed{1} \\ G = \boxed{1} \end{bmatrix} \overset{\leq}{\sqcap} \begin{bmatrix} F = a \\ G = b \end{bmatrix} = \left(\begin{bmatrix} F = \boxed{1}\ a \\ G = \boxed{1} \end{bmatrix}, \begin{bmatrix} F = \boxed{1}\ b \\ G = \boxed{1} \end{bmatrix} \right)$$ Calder Carpenter (credulous)

$$= \begin{bmatrix} F = \boxed{1} \\ G = \boxed{1} \end{bmatrix}$$ Bouma Carpenter (skeptical) LRL

$$= \bot$$ Russell et al

Here the presence of reentrancy in the non-default means that the two default values are in conflict. A credulous definition will return multiple values; skeptical definitions return only that information which is common to all the credulous results. The difficulty in defining default unification is to exclude the possibility of the result de-

pending on the order in which individual parts of the default feature structure are unified with the non-default feature structure. All these definitions do exclude such order dependence.

(4)

$$\begin{bmatrix} F = a \\ G = \top \end{bmatrix} \lessdot \begin{bmatrix} F = \boxed{1}\ b \\ G = \boxed{1} \end{bmatrix} = \left(\begin{bmatrix} F = \boxed{1}\ a \\ G = \boxed{1} \end{bmatrix}, \begin{bmatrix} F = a \\ G = b \end{bmatrix} \right) \quad \text{Calder, Carpenter (credulous)}$$

$$= \begin{bmatrix} F = a \\ G = \top \end{bmatrix} \quad \text{Carpenter (skeptical)}$$

$$= \begin{bmatrix} F = \boxed{1}\ a \\ G = \boxed{1} \end{bmatrix} \quad \text{LRL}$$

$$= \begin{bmatrix} F = a \\ G = b \end{bmatrix} \quad \text{Bouma}$$

$$= \bot \quad \text{Russell et al}$$

In example 3 there is no basis for deciding which of the conflicting information in the default structure should be incorporated. However in example 4 it is possible to claim that there is a basis for distinguishing between the two pieces of default information which could be potentially incorporated but which are in conflict with each other, since one involves a path equivalence specification and the other a path value specification;

```
< f > = < g >
< g > = b
```

As discussed below, in the LRL we made a decision to prefer specifications of equivalence to specifications of values. Bouma's result arises because he normalises default structures, in a way which gives

```
< f > = b
< g > = b
```

for the default structure in example 4 (on the basis that this gives a structure which is equivalent with respect to unification).

(5)

$$\begin{bmatrix} F = \top \\ G = a \\ H = b \\ J = \top \end{bmatrix} \lessdot \begin{bmatrix} F = \boxed{1} \\ G = \boxed{1} \\ H = \boxed{1} \\ J = \boxed{1} \end{bmatrix} = (4 \text{ possibilities}) \quad \text{Calder, Carpenter (credulous)}$$

$$= \begin{bmatrix} F = \top \\ G = a \\ H = b \\ J = \top \end{bmatrix} \quad \text{Carpenter (skeptical), LRL (current)}$$

$$= \begin{bmatrix} F = \boxed{1} \\ G = a \\ H = b \\ J = \boxed{1} \end{bmatrix} \quad \text{Bouma}$$

$$= \bot \quad \text{Russell et al, LRL (proposed)}$$

Here the conflict in the default information is entirely between specifications of reentrancy.

In order to compare some of the varieties of default unification we will formalise them in terms of successively unifying pieces of information carried by the default into the non-default feature structure while taking account of possible conflicts (cf Russell et al). A critical notion here is that of "pieces of information"; we can define a general notion of decomposition of a feature structure $Decomp(F)$ into component pieces of information, which must meet the following criterion (if default unification is to have the property of being equivalent to ordinary unification in the cases where that would succeed):

$\sqcap(Decomp(F)) = F$
F is equal to the unification of all the information in its decomposition. (As illustrated above Bouma's normalisation of the default does not meet this criterion.)

By considering the case where decomposition is into the minimal atomic units of information $At(F)$ (Carpenter p. 9), we can give a definition which is equivalent to Carpenter's skeptical default unification:

$F_1 \overset{<}{\sqcap} F_2 = F_1 \sqcap \bigsqcap \{F \in At(F_2) \mid F_1 \sqcap F \neq \bot$ and there is no F' such that $F_2 \sqsubseteq F'$ and $F' \sqcap F_1 \neq \bot$ and$F' \sqcap F_1 \sqcap F = \bot\}$

The intuitive basis for this definition is to consider successively adding the minimal (atomic) units of information from the default into the non-default. In the cases where there are conflicts, such as examples 3, 4 and 5 above, this would give different results depending on the order in which the atomic feature structures were added in. To produce the equivalent of credulous default unification we would do the addition once for each possible ordering of default atomic feature structures (and remove duplicates). The definition above is equivalent to skeptical default inheritance because only information which is consistent with all possible orderings is added. It is thus obvious that the complexity of the algorithm as described is unacceptable (worse than exponential) since checking for all possible F' would involve creating the unification of each member of the power-set of $At(F_2)$. Reducing the complexity of this definition to acceptable levels seems unlikely to be possible, given the behaviour of examples such as 5, above.

The option taken by Russell et al(1991, this volume) is to keep the tractable (near-linear) behaviour of ordinary unification at the cost of saying that default unification fails under the circumstances where there are conflicts in the default information. In terms of our definition above we can split $At(F)$ into $PE(F)$, the set of path equivalence specifications, and $PV(F)$ the set of path value specifications. If the reentrant part of the default unifies with the non-default, and the reentrant part of the non-default unifies with the default, no conflicts can arise in the default information. Thus Russell et al have:

$$F_1 \overleftarrow{\sqcap} F_2 = \bot \text{ if either } \sqcap\mathrm{PE}(F_2) \sqcap F_1 = \bot \text{ or } \sqcap\mathrm{PE}(F_1) \sqcap F_2 = \bot$$
$$= F_1 \sqcap \bigsqcap\{F \in At(F_2) \mid F_1 \sqcap F \neq \bot\} \text{ otherwise}$$

The LKB's current default unification algorithm also makes use of a distinction between reentrant and non-reentrant atomic feature structures. The definition used is:

> $F_1 \overleftarrow{\sqcap} F_2 = F_1 \sqcap \bigsqcap\{F \in At(F_2) \mid F_1 \sqcap F \neq \bot$ and for all conflicting F' such that $F_2 \sqsubseteq F'$and$F' \sqcap F_1 \neq \bot$ and$F' \sqcap F_1 \sqcap F = \bot$, F "takes precedence over" $F'\}$
> Where F takes precedence over F' iff F is a specification of path equivalence ($F \in \mathrm{PE}(F_1)$) and F' contains at least one path value specification.

Thus we introduced a precedence order between path equivalence and path value specifications. This was actually done because the linguists involved in designing the LRL expressed a preference for a behaviour where reentrancy took precedence over values; for our application in particular this seems desirable because reentrancy is usually set up specifically, by the linguist, whereas values are more likely to be acquired automatically. (However, in our use of the LKB so far, this means that reentrancy tends to be set up in the type system, and is thus, in effect, non-default. This is considered further below.)

This definition can, in practice, be implemented considerably more efficiently than Carpenter's, although its worst case behaviour is still worse than exponential. Initially the re-entrant parts of the default feature structure can be unified individually with the non-default, and it is only necessary to consider conflicts that arise in the reentrant set. Thus the exponential term involves only the path equivalence specifications and since typically $|\mathrm{PE}(F)| \ll |At(F)|$ the implementation is not unreasonably slow. (Furthermore this is the worst case behaviour; it is usually possible to split $\mathrm{PE}(F)$ into sets which are guaranteed not to interact). The procedure then reduces to one of default unifying a tree-structured feature structure with a non-default reentrant structure. There are still possible conflicts, of the type in example 3 above (which would cause unification failure by Russell et al's definition). However it is possible to allow for these with a linear algorithm by storing the original non-default value in the feature structure representation at reentrant points as unification proceeds, and reverting to it if a conflict arises. In effect, what we are relying on is that if the non-default feature structure is the only reentrant one, all conflicts are localised.

But this definition still seems unsatisfactory, even though it meets all the criteria we enumerated at the beginning of this section. The worst case complexity is terrible, the implementation is awkward, and the behaviour can be obscure. A better compromise seems to be to specify that inheritance of information about reentrancy is non-defeasible, and that default unification fails in the case where the non-default feature structure and the reentrant part of the default feature structure do not unify.

> $F_1 \overleftarrow{\sqcap} F_2 = F_3 \sqcap \bigsqcap\{F \mid F \in PV(F_2)$ and there is no $F' \in PV(F_2)$ such that

$F' \sqcap F_3 \neq \bot$ and $F' \sqcap F_3 \sqcap F = \bot\}$
where $F_3 = F_1 \sqcap \bigsqcap PE(F_2)$

Such a definition, where default unification involves filling in values, and expanding, rather than modifying, existing feature structure skeletons, is relatively simple to understand. (It also avoids the rather complex behaviour of Carpenter's definition with respect to the difference between specified and unspecified paths.) In practice changing the definition seems unlikely to cause any significant problems with our use of the LKB, because reentrancy tends to be specified in the type system, and is in effect not defeasible.

4 Default unification of typed feature structures

Consideration of typed feature structures further multiplies the possible definitions of default inheritance. Rather than attempting to even approximate to a definition which corresponds to incorporating the maximal amount of information carried by the types in a feature structure, we have chosen to use a definition in which the type system constrains default inheritance. Information which is carried by any feature structure which is part of the default is incorporated if it is fully compatible with the non-default (ie unifies with the relevant part), but is only ever partially incorporated (ie default unified) if its type is the same as, or more general than, the non-default.

For example, given:

$$F_1 = \begin{bmatrix} \mathbf{t1} \\ \mathtt{F} = \begin{bmatrix} \mathbf{t2} \\ \mathtt{G} = \mathbf{t5} \\ \mathtt{H} = \top \end{bmatrix} \\ \mathtt{J} = \begin{bmatrix} \mathbf{t3} \\ \mathtt{G} = \mathbf{t5} \\ \mathtt{H} = \top \end{bmatrix} \end{bmatrix}; F_2 = \begin{bmatrix} \mathbf{t1} \\ \mathtt{F} = \begin{bmatrix} \mathbf{t2} \\ \mathtt{G} = \mathbf{t6} \\ \mathtt{H} = \mathbf{t7} \end{bmatrix} \\ \mathtt{J} = \begin{bmatrix} \mathbf{t4} \\ \mathtt{G} = \mathbf{t6} \\ \mathtt{H} = \mathbf{t7} \end{bmatrix} \end{bmatrix};$$

$\mathbf{t5} \sqcap \mathbf{t6} = \bot$ and $\mathbf{t3} \not\sqsubseteq \mathbf{t4}$

then

$$F_1 \overleftarrow{\sqcap} F_2 = \begin{bmatrix} \mathbf{t1} \\ \mathtt{F} = \begin{bmatrix} \mathbf{t2} \\ \mathtt{G} = \mathbf{t5} \\ \mathtt{H} = \mathbf{t7} \end{bmatrix} \\ \mathtt{J} = \begin{bmatrix} \mathbf{t3} \\ \mathtt{G} = \mathbf{t5} \\ \mathtt{H} = \top \end{bmatrix} \end{bmatrix}$$

We can formalise this in terms of a decomposition function (TypeDecomp) which differs from the atomic decomposition function in that it does not split up the feature structures completely. Only parts of the feature structure which are fully type

compatible with the non-default structure are split; TypeDecomp is thus defined relative to the non-default structure. For example:

$$\text{TypeDecomp}(F_1, F_2) =$$

$$\left\{ \begin{bmatrix} \texttt{t1} \\ \texttt{F} = \begin{bmatrix} \texttt{t2} \\ \texttt{G} = \texttt{t6} \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \texttt{t1} \\ \texttt{F} = \begin{bmatrix} \texttt{t2} \\ \texttt{H} = \texttt{t7} \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \texttt{t1} \\ \texttt{J} = \begin{bmatrix} \texttt{t4} \\ \texttt{G} = \texttt{t6} \\ \texttt{H} = \texttt{t7} \end{bmatrix} \end{bmatrix} \right\}$$

5 Inheritance hierarchies and multiple inheritance

One way of allowing inheritance to operate over a hierarchy would be to modify the definition of default unification, to order the information units in a way which corresponded to the inheritance hierarchy (cf the way in which we defined preference of path equivalence specifications to path value specifications). Clearly this is not any more computationally feasible than the formulation of default unification which we gave at the beginning. Conflicts can arise, not just from reentrancy, but also from multiple inheritance conflicts of the Nixon diamond type, where there is no ordering between defaults to allow resolution. Again we could produce variant definitions; if reentrancy is regarded as non-defeasible for example, all the reentrant atomic feature structures could be unified first and if that succeeded the non-reentrant structures could be considered in groups according to their priority. Essentially definitions along these lines give a skeptical, "bottom-up", inheritance scheme.

We did not attempt to implement such a scheme in the LKB. We define inheritance to operate top-down over whole feature structures; that is a psort will be fully expanded with inherited information before it is used for default inheritance. As Carpenter explains this can give different results from a bottom-up definition since default unification is non-associative. (The particular example that Carpenter uses does not have non-associative behaviour under our definition of default unification, but there are other cases which do.) If we view default inheritance in terms of individual units of information being asserted at various points in an inheritance hierarchy, top-down inheritance can result in information which is asserted at a higher level being preferred over information asserted at the lower level.

However we want default inheritance to be a relationship between coherent parts of fully formed lexical entries. Thus we believe that the top-down behaviour is justifiable. It is also far more efficient than bottom-up inheritance would be, for this application, since the expanded psort can be cached. And again in practice, the top-down, bottom-up distinctions in behaviour arise with very low frequency.

Our decision to restrict multiple default inheritance to the cases where the information inherited is consistent was determined by our use of semi-automatically acquired data. A fundamental point is that we cannot decide on an appropriate way of resolving conflicts in multiple inheritance without knowing what type of conflicts actually arise. Given that automatic extraction of information from MRDs is inevitably error prone, and that lexicographers' definitions are frequently not mutually consistent, we expected that most conflicts would be due to errors in the extraction

process, or to inadequate definitions. Thus disallowing multiple inheritance conflicts seemed reasonable as an initial position. This at least gives the user the option of manually editing the lexical entries in order to get the desired behaviour, whereas any approach which did not signal the presence of conflicts would not. We will review our approach after making further use of the LKB to represent further lexical semantic information.

References

Bouma G(1990) 'Defaults in Unification Grammar', *Proceedings of the 28th ACL,* Pittsburg, pp.165–173

Brachman, R.J., Gilbert, V.P. and Levesque, H.J.(1985) 'An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of Krypton', *Proceedings of the 9th IJCAI,* Los Angeles, pp.532–539

Briscoe E J, Copestake A A and Boguraev B K(1990) 'Enjoy the paper: Lexical semantics via lexicology', *Proceedings of the 13th Coling,* Helsinki, pp.42–47

Calder J(1991) *Some notes on Priority Union,* Paper presented at the ACQUILEX Workshop on Default Inheritance in the Lexicon, Cambridge

Carpenter R(1990) 'Typed feature structures: Inheritance, (In)equality and Extensionality', *Proceedings of the First International Workshop on Inheritance in Natural Language Processing,* Tilburg, The Netherlands, pp.9–18

Carpenter R(1991) 'Skeptical and Credulous Default Unification with Applications to Templates and Inheritance', *Proceedings of the ACQUILEX Workshop on Default Inheritance in the Lexicon,* Cambridge

Carroll J and Grover C(1989) 'The derivation of a large computational lexicon for English from LDOCE' in Boguraev B and Briscoe E J (eds.), *Computational lexicography for natural language processing,* Longman, London, pp.117–134

Carroll J(1990) *Lexical Database System: User Manual,* Esprit BRA-3030 ACQUILEX deliverable no. 2.3.3(c)

Copestake A(1990) 'An approach to building the hierarchical element of a lexical knowledge base from a machine readable dictionary', *Proceedings of the First International Workshop on Inheritance in Natural Language Processing, The Netherlands,* Tilburg, pp.19–29

Copestake A A and Briscoe E J(1991) 'Lexical Operations in a Unification Based Framework', *Proceedings of the ACL SIGLEX Workshop on Lexical Semantics and Knowledge Representation,* Berkeley, California, pp.88-101

Copestake A and Jones B(1991) *Support for multi-lingual lexicons in the LKB system,* ms University of Cambridge, Computer Laboratory

Daelemans W(1990) 'Inheritance in Object-Oriented Natural Language Processing', *Proceedings of the First International Workshop on Inheritance in Natural Language Processing, The Netherlands,* Tilburg, pp.30–39

Evans R and Gazdar G (eds)(1990) *The DATR papers,* Cognitive Science Research Paper CSRP 139, School of Cognitive and Computing Sciences, University of Sussex

Krieger H and Nerbonne J(1991) 'Feature-Based Inheritance Networks for Computational Lexicons', *Proceedings of the ACQUILEX Workshop on Default Inheritance in the Lexicon,* Cambridge

Moshier M D and Rounds W C(1987) 'A logic for partially specified data structures', *Proceedings of the 14th ACM Symposium on the Principles of Programming Languages,* , pp.156–167

Pustejovsky J(1989) 'Current issues in computational lexical semantics', *Proceedings of the 4th European ACL,* Manchester, pp.xvii–xxv

Rodriguez H et al(1991) *Guide to the extraction and conversion of taxonomies,* ACQUILEX project draft user manual, Universidad Politechnica de Catalunya, Barcelona

Russell G, Ballim A, Carroll J and Warwick-Armstrong S(1991) 'A Practical Approach to Multiple Default Inheritance for Unification-Based Lexicons', *Proceedings of the ACQUILEX Workshop on Default Inheritance in the Lexicon,* Cambridge

Shieber S(1986) *An Introduction to Unification-based Approaches to Grammar,* University of Chicago Press, Chicago

Using the LKB

Ann Copestake
Computer Laboratory, University of Cambridge

1 The LKB implementation

The LKB as described in the previous papers is fully implemented in Procyon Common Lisp running on Apple Macintoshes. It is in use by all the groups involved in the ACQUILEX project. In total there are currently about 20 users on five sites in different countries. Interaction with the LKB is entirely menu-driven. Besides the obvious functions to load and view types, lexical entries, lexical rules and so on, there are various other facilities which are necessary for the application. A very simple (and inefficient) parser is included, to aid development of types and lexical entries. There are tools for supporting multilingual linked lexicons, described in Copestake and Jones (1991), which we will not discuss here. The LKB is integrated with our lexical database system (LDB, Carroll 1990) so that information extracted from dictionary entries stored in the LDB can be used to build LKB lexicons.

The type system which has been developed for use on the ACQUILEX project is fairly large (about 450 types and 80 features). Since expanded feature structures for lexical entries are complex, the display facilities allow selective viewing of feature structures, either individually or according to type. For example, a user who is not interested in the verb syntactic structure can "shrink" that part of the feature structure for the type **verb-sign** (by selecting **Shrink** on the menu attached to the node in the display) and all subsequently expanded noun lexical entries will show only their CAT type rather than the entire feature structure.

Currently nearly 10,000 lexical entries containing syntactic and semantic information have been stored in the LKB. The bulk of these entries are currently made up of English nouns derived from LDOCE for which the main semantic information is inherited down semi-automatically derived taxonomies (Copestake 1990). Work has begun on deriving multi-lingual linked lexicons.

Given the complexity of the feature structures for lexical entries, and the size of the lexicons to be supported by the LKB, it is clearly not possible to store lexicons in main memory. Lexical entries are thus stored on disk, to be expanded as required. Entries may be indexed by type of feature structure at the end of user-defined paths, and also by the psort(s) which they inherit information from, although producing such indices for large lexicons is time consuming. Checking lexical entries (for well-formedness, default inheritance conflicts and presence of cycles) can be carried out

at the same time as indexing or acquisition.

2 Typing and automatic acquisition of large lexicons

Our notion of typing of feature structures can be regarded as a way of getting the inheritance functionality of templates in untyped feature structure formalisms, with the added advantages of type checking and type inference. These advantages are discussed in Carpenter(1990); here we consider the utility of typing feature structures for our application.

As a method of lexical organisation, types have significant advantages over templates, for a large scale collaborative project. Once an agreed type system is adopted, the compatability of the data collected by each site is guaranteed (there may of course be problems of differing interpretation of types and features but this applies to any representation). In an untyped feature system, typographical errors and so on may go undetected, and debugging a large template system can be extremely difficult; a type system makes error detection much simpler. Since a given feature structure has a type permanently associated with it, is also much more obvious how information has come to be inherited than if templates are used.

Essentially the same advantages of safety and clarity apply to strict typing of feature structures as to strict typing in programming languages. Of course a reduction in flexibility of representation has to be accepted, once a particular type system is adopted for a project. In practise however, encoding the agreed representation in terms of a type system, rather than by means of templates, makes global alterations much easier because of the localisation of the information and because of the error checking. (For example, if a change is made in a feature name, it is easy to find all types which will be affected, since a feature can only be introduced at one point in the hierarchy. Any residual occurrences of the old feature name in lexical entries will be picked up as errors by the type system.) The type system determines the path structure of a lexical entry; for inheritance of values more flexibility may be needed, but this is provided by the default mechanism.

The type system can be integrated with tools for semi-automatic analysis of dictionary definitions, and initial work on this is described in Rodriguez et al(1991). Types are correlated with the templates used in a robust pattern matching parser, and user interaction can be controlled by the type system. For example, manual association of feature values with psorts can be a highly efficient method of acquiring information, since hundreds of entries will inherit from psorts such as drink_L_2_1. The user is only allowed to introduce information appropriate for a particular type, and a menu-based interface can both inform the user of the possibilities and prevent errors.

The utility of typing for error checking when representing automatically acquired data can be seen in the following simple example. The machine readable version of LDOCE associates semantic codes with senses. Examples of such codes are P for plant, H for human, M for male human, K for male human or animal, and so on.

When automatically acquiring information about nouns from LDOCE, we specify a value for the feature SEX, where this is possible according to the semantic codes. Thus the automatically created lexical entry for bull 1 (1) contains the line:

```
< rqs : sex > = male
```

In the current type system the feature SEX is introduced at type **creature**. A few entries have incorrect semantic codes; **Irish stew** for example has code K. Since **Irish stew** is under FOOD_L_1_1 in the taxonomy its RQS type is **c_object**, which is not consistent with **creature**, and therefore SEX was detected as an inappropriate feature. Attempts at expansion of the automatically generated lexical entry caused an error message to be output, and the user had the opportunity to correct the mistake. If the LKB were not a typed system, errors such as this would not be detected automatically in this way.

In contrast, automatic classification of lexical entries by type, according to feature information, can be used to force specification of appropriate information. A lexical entry which has not been located in a taxonomy will be given the most general possible type for its RQS. However if a value for the feature SEX has been specified this forces an RQS type of **creature**. This would also force the value of ANIMATE to be **true**, for example. Whether it is acceptable to do this without checking with the user depends on the observed reliability of the assignment of the attribute. To continue with the current example, the error rate found where sex denoting semantic codes were assigned to lexical entries of inappropriate type was under 0.5%.[1] This is sufficiently reliable that automatic type inference is appropriate. However other automatic feature assignments will be the result of parsing definitions, where the reliability is much lower, especially given the use of automatic techniques for ambiguity resolution.

One limitation of the type system as described is that it is not possible in general to enforce cooccurrence restrictions, even of a quite limited sort. For example Sanfillipo's representation of verb semantics in the LKB involves using thematic roles and encoding restrictions on arguments of a predicate by sorting the variables. In order to do this a type **theta-formula** is defined to have the following constraint:

$$\left[\begin{array}{l} \textbf{theta-formula} \\ \text{IND} = \boxed{1} \quad \textbf{eve} \\ \text{PRED} = \textbf{theta-relation} \\ \text{ARG1} = \boxed{1} \\ \text{ARG2} = \textbf{obj} \end{array}\right]$$

To classify psychological predicates thematic predicates such as **theta-sentient** are used; in this case the second argument to any formula whose predicate is **theta-sentient** should denote a sentient entity; ie if the value of PRED is **theta-sentient** then the value of ARG2 is **e-sentient**. But the nearest we could get to achieving this would be to define a subtype of **theta-sentient**, **theta-sentient-formula**, with constraint:

[1] This is not the same as the overall error rate; for example a lexicographer might have used a code M, where H would have been more appropriate, and this would not be detected by the system.

$$\begin{bmatrix} \textbf{theta-sentient-formula} \\ \text{IND} = \boxed{1} \quad \textbf{eve} \\ \text{PRED} = \textbf{theta-sentient} \\ \text{ARG1} = \boxed{1} \\ \text{ARG2} = \textbf{e-sentient} \end{bmatrix}$$

and to define other subtypes for the other possible theta relations. This does not really achieve the desired result however. For example:

$$\begin{bmatrix} \textbf{theta-formula} \\ \text{IND} = \boxed{1} \quad \textbf{eve} \\ \text{PRED} = \textbf{theta-sentient} \\ \text{ARG1} = \boxed{1} \\ \text{ARG2} = \textbf{e-plant} \end{bmatrix}$$

is still a well-formed feature structure, despite the fact that it cannot be extended to be a well-formed structure with a type corresponding to that of any leaf node in the type hierarchy (assuming that **e-plant** $\sqcap$ **e-sentient** $= \bot$). This seems undesirable; the type system is supposed to be complete, so intuitively we might expect such a feature structure to be ill-formed in some sense. It seems clear that we cannot check for such cases efficiently in general, because to do so would, in the worst case, involve attempting to unify the feature structure with the constraints of all leaf types which were subtypes of its type.

We refer to a feature structure which can be extended to a well-formed structure where every type is a leaf type as "ultimately well-formed", and we can enforce such cooccurrence restrictions when automatically acquiring lexical entries from the MRDs by checking for ultimate well-formedness. This does not impose an unreasonable overhead in practice.

Efficiency gains arising directly from the use of types were not a major factor in our decision to use a typed system. Although unification of typed feature structures will be somewhat more efficient than untyped ones, as unification will fail as soon as type conflict occurs, this is not particularly important in the LKB, since most unifications will be performed while expanding lexical entries when the vast majority of unifications would be expected to succeed. In fact, because there is some overhead in typing the feature structures, the use of types probably decreases efficiency slightly, although the unifications involved will be comparable to those needed if the same information were conveyed by templates. But the LKB has to cope with large lexicons, with thousands of complex lexical entries, and thus space efficiency rather than speed is the major consideration. The most important factor in space efficiency is the use of inheritance, both in the type system and the psort system, which allows unexpanded lexical entries to be very compact.

We expect the efficiency advantages of typing in the LKB to come more indirectly, in that the type system provides the main method of constraining the application of lexical rules and translation links. If generalisations are encoded in a type system in an appropriate fashion, such that lexical rules can be defined in such a way that they apply to succinctly characterisable parts of the lexicon, then indexing lexical entries by type, should allow the efficient application of lexical rules (see Copestake and Briscoe 1991).

3 Conclusion

The aim of the ACQUILEX project is to demonstrate that large-scale lexical information can be acquired from MRDs and represented in a way that makes it usable by a range of NLP systems. The first essential for this is a well-defined representation language, which is efficiently implementable. We have chosen to use an LRL which is relatively "theory-neutral" in the same sense as PATR-II; it could be used to implement different linguistic theories. It is, of course, not possible to represent information in the LRL in a theory-neutral way; the second essential requirement is to have some theory of the data to be represented which can be encoded in the LRL. Sanfilippo's paper shows how this can be done for one group of verbs; this also illustrates one advantage that our LRL has over PATR-II, in that the type system makes the encoding of the theory more explicit. It is however possible to make use of the information encoded even if a different treatment is adopted; for example deriving a verb lexicon for a system which did not rely on theta roles to express semantic argument structure would be straightforward, because information simply has to be lost rather than added. In the final paper we have explained why the type system is particularly important for automatic data acquisition.

Clearly linguistic theories, their encoding in the LRL, and even the LRL itself, may have to be modified in response to the data. It is not currently possible to construct a large lexicon which incorporates lexical semantic information without doing some work on the linguistic theory, since formal lexical semantics is a relatively undeveloped field. And in our discussion of the LRL we have shown that there are problematic areas, in the treatment of defaults in particular, where modifications will be required. We are using default inheritance as a means of structuring the lexicon with semi-automatically acquired data (whereas Sanfilippo's work involves instantiating structures with semi-automatically acquired information), and although there is some theoretical motivation behind this use (for which see Copestake 1990), this is far from being completely worked out. However, even if further changes are made to the LRL, most, if not all, of the existing data will be reusable, because the current language has been explicitly specified.

Acknowledgements

This work was supported by Esprit BRA-3030, ACQUILEX 'The Acquisition of lexical knowledge for Natural Language Processing systems'. Several people contributed in various ways to the design, implementation and development of the LKB, especially Valeria de Paiva, Antonio Sanfilippo, Ted Briscoe, John Carroll, John Bowler and Horacio Rodriguez. We are very grateful to Bob Carpenter for his detailed comments on our use of typed feature structures and default unification.

References

Bouma G(1990) 'Defaults in Unification Grammar', *Proceedings of the 28th ACL,* Pittsburg, pp.165–173

Brachman, R.J., Gilbert, V.P. and Levesque, H.J.(1985) 'An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of Krypton', *Proceedings of the 9th IJCAI,* Los Angeles, pp.532–539

Briscoe E J, Copestake A A and Boguraev B K(1990) 'Enjoy the paper: Lexical semantics via lexicology', *Proceedings of the 13th Coling,* Helsinki, pp.42–47

Calder J(1991) *Some notes on Priority Union,* Paper presented at the ACQUILEX Workshop on Default Inheritance in the Lexicon, Cambridge

Carpenter R(1990) 'Typed feature structures: Inheritance, (In)equality and Extensionality', *Proceedings of the First International Workshop on Inheritance in Natural Language Processing,* Tilburg, The Netherlands, pp.9–18

Carpenter R(1991) 'Skeptical and Credulous Default Unification with Applications to Templates and Inheritance', *Proceedings of the ACQUILEX Workshop on Default Inheritance in the Lexicon,* Cambridge

Carroll J and Grover C(1989) 'The derivation of a large computational lexicon for English from LDOCE' in Boguraev B and Briscoe E J (eds.), *Computational lexicography for natural language processing,* Longman, London, pp.117–134

Carroll J(1990) *Lexical Database System: User Manual,* Esprit BRA-3030 ACQUILEX deliverable no. 2.3.3(c)

Copestake A(1990) 'An approach to building the hierarchical element of a lexical knowledge base from a machine readable dictionary', *Proceedings of the First International Workshop on Inheritance in Natural Language Processing, The Netherlands,* Tilburg, pp.19–29

Copestake A A and Briscoe E J(1991) 'Lexical Operations in a Unification Based Framework', *Proceedings of the ACL SIGLEX Workshop on Lexical Semantics and Knowledge Representation,* Berkeley, California, pp.88-101

Copestake A and Jones B(1991) *Support for multi-lingual lexicons in the LKB system,* ms University of Cambridge, Computer Laboratory

Daelemans W(1990) 'Inheritance in Object-Oriented Natural Language Processing', *Proceedings of the First International Workshop on Inheritance in Natural Language Processing, The Netherlands,* Tilburg, pp.30–39

Evans R and Gazdar G (eds)(1990) *The DATR papers,* Cognitive Science Research Paper CSRP 139, School of Cognitive and Computing Sciences, University of Sussex

Krieger H and Nerbonne J(1991) 'Feature-Based Inheritance Networks for Computational Lexicons', *Proceedings of the ACQUILEX Workshop on Default Inheritance in the Lexicon,* Cambridge

Moshier M D and Rounds W C(1987) 'A logic for partially specified data structures', *Proceedings of the 14th ACM Symposium on the Principles of Programming Languages,* , pp.156–167

Pustejovsky J(1989) 'Current issues in computational lexical semantics', *Proceedings of the 4th European ACL,* Manchester, pp.xvii–xxv

Rodriguez H et al(1991) *Guide to the extraction and conversion of taxonomies,* ACQUILEX project draft user manual, Universidad Politechnica de Catalunya, Barcelona

Russell G, Ballim A, Carroll J and Warwick-Armstrong S(1991) 'A Practical Approach to Multiple Default Inheritance for Unification-Based Lexicons', *Proceedings of the ACQUILEX Workshop on Default Inheritance in the Lexicon,* Cambridge

Shieber S(1986) *An Introduction to Unification-based Approaches to Grammar,* University of Chicago Press, Chicago