A Practical Approach to Multiple Default Inheritance for Unification-Based Lexicons

Graham Russell*
ISSCO

Afzal Ballim*
ISSCO

John Carroll†
Cambridge University Computer Laboratory

Susan Warwick-Armstrong*
ISSCO

This paper describes a unification-based lexicon system for NLP applications that incorporates mechanisms for multiple default inheritance. Such systems are intractable in the general case—the approach adopted here places a number of restrictions on the inheritance hierarchy in order to remove some of the sources of complexity while retaining more desirable properties. Implications of the design choices are discussed, comparisons are drawn with related work in computational linguistics and AI, and illustrative examples from the lexicons of German and English are given.

1. Introduction

In essence, the primary task of a computational lexicon is to associate character strings representing word forms with various types of information, able to account for their distribution within a sentence and for their contribution to the meaning of a text. Economical lexical organization appears to require the ability, on the one hand, to make general statements about classes of words, and, on the other, to express exceptions to such statements affecting both individual words and subclasses of words. These considerations have provoked interest in applying to the domain of the lexicon AI knowledge representation techniques involving the notions of inheritance and defaults.[1] Unfortunately, many of the schemes that have been proposed are highly complex; departing from simple tree-form taxonomies dramatically increases the problems of dealing in a systematic fashion with default inheritance. Not only are the intuitions underlying the behavior of systems such as those of Touretzky (1986) and Sandewall (1986) unstable (as Touretzky et al. [1987] show), Selman and Levesque (1989) demonstrate that certain of them are NP-hard and thus effectively intractable.

In a well-defined domain such as the lexicon there remains the possibility of mitigating these problems by adopting a principled compromise; it may be advantageous to surrender some of the generality of an inheritance system, if the more restricted version that results retains sufficient expressive power for the domain in question. We expand on this below and propose a number of simplifications that are motivated by the particular task that the system is intended to perform.

* ISSCO, 54 route des Acacias, 1227 Geneva, Switzerland.
† Cambridge University Computer Laboratory, New Museums Site, Pembroke Street, Cambridge CB2 3QG, UK.

1 See, e.g., Evans and Gazdar 1990, Gazdar 1990 (especially the References), much of the material in Daelemans and Gazdar 1990, and the contributions to Briscoe et al. 1991. We discuss some of this work below.

The system described here has been implemented as part of the ELU[2] unification grammar development environment for research in machine translation, made up of parser, generator, lexicon, and transfer mechanism. The user language resembles that of PATR-II (Shieber 1986a), but provides a larger range of data types and more powerful means of stating relations between them. Among the requirements imposed by the context within which this system is used are (i) the ability to both analyze and generate complex word forms, (ii) full integration with existing parts of the ELU environment, and (iii) the ability to accommodate a relatively large number of words. In particular, an important objective is to preserve as far as possible the flavor of this type of environment: a specialized programming language for linguistic descriptions, suitable for interpretation by a variety of programs performing tasks such as analysis and generation of sentences and phrases, lexical lookup, etc., which associates with natural language expressions representations defined by the writer of a linguistic description and employs unification as the method of combining information and enforcing constraints. The intention is that the ELU language should be as general as possible, in the sense of imposing minimal restrictions on the form of linguistic analyses and descriptions.

Unification is an attractive choice as the basic computational mechanism of such a system, for well-known reasons: its algebraic properties permit a declarative, monotonic semantics for the language that is independent of whatever programs interpret it. However, it does not always meet the requirements of practical linguistic descriptions. The treatment of exceptions is a case in point; negation and conditional statements of the kind excluded by pure unification frequently play a large part in accounts of natural language phenomena, and the best-motivated area for the application of these ideas appears to be the lexicon. It is this problem that the current work seeks to address. We would hope that the interest of the system presented here extends beyond the particular environment for which it has been developed; as an attempt to reconcile two apparently disparate paradigms, the combination of inheritance and unification may be of interest in quite different contexts.

The remainder of the paper is organized as follows. Section 2 discusses general issues concerning the organization of the system: the basic formalism is introduced, together with the notions of multiple inheritance, default inheritance, and class precedence. Section 3 describes the system in greater detail, showing how the form of the inheritance hierarchy determines how information in lexical specifications is combined, and Section 4 provides some comparisons with related work. Some more extended linguistic examples appear in Section 5; these serve to illustrate the various mechanisms described earlier and suggest methods that might be adopted in formulating other analyses. Finally, in Section 6, the current implementation is discussed.

2. The Lexicon as a Hierarchy

2.1 An Overview of the Formalism

An ELU lexicon consists of a number of 'classes,' each of which is a structured collection of constraint equations encoding information common to a set of words, together with links to other more general 'superclasses.' For example, if an 'intransitive' class is used to express the common syntactic properties shared by all intransitive verbs, then particular instances of intransitive verbs can be made to inherit this informa-

2 "Environnement Linguistique d'Unification" (Estival 1990). See also Johnson and Rosner (1989) for a description of the earlier UD system on which ELU is based.

tion by specifying the 'intransitive' class as one of their superclasses—it then becomes unnecessary to specify the relevant properties individually for each such verb. Similarly, the 'intransitive' class need not express any of the more general properties of the 'verb' class. The lexicon may be thought of as a tangled hierarchy of classes linked by inheritance paths, with, at the most specific level, lexical classes and, at the most general, classes for which no superclasses have been defined; these therefore inherit no information from elsewhere. Basic lexical entries are themselves classes,[3] and any information they contain is standardly specific to an individual word; lexical and nonlexical classes differ in that analysis and generation take only the former as entry points to the lexicon.

It is possible to define certain properties of a class as defeasible. Such a property will be inherited from a superclass only if it is consistent with all properties contributed by more specific classes; if it is not consistent, then it is ignored. Defeasible constraints imposed on a class of words may thus be overridden by information associated with individual members or subclasses of that class; this is the notion of default that underlies the system. It is possible, for example, to place in the class expressing general properties of verbs an equation such as '`<aux> = no`' (i.e. "typical verbs are not auxiliaries"), while placing the contradictory specification '`<aux> = yes`' in a subclass from which only auxiliaries inherit. The ability to encode exceptional properties of lexical items is extremely attractive from the linguistic point of view; the lower the position in the hierarchy at which the property appears, the more exceptional it may be considered.

Like other parts of an ELU description, a lexicon consists of a set of statements that are compiled into an internal format for use in analysis and generation of linguistic expressions. Section 2.6 describes some relevant aspects of the lexical compilation process.

2.1.1 Class Definition. A class definition consists of the compiler directive '`#Class`' (for a nonlexical class) or '`#Word`' (for a lexical class),[4] 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 (possibly empty) 'variant' equation sets.

2.1.2 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 2.5); 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. Precedence of direct superclasses is

3 Thus no distinction is made between classes and 'instances,' as in, e.g., KL-ONE (Schmolze and Lipkis 1983) and much more recent work. One reason for this is that instances of the kind appealed to by some KR researchers do not exist within a unification-based linguistic description. What would count as an instance in this context is an occurrence of a word in the analysis of some text; in general, information associated with one particular occurrence of a word will originate partly in its lexical specification and partly through unifications with other items.

A second, related reason for not enforcing a class/instance distinction is that lexical classes may themselves usefully be inherited from; examples of this behavior are discussed in Section 5.2.

4 Both lexical and nonlexical classes have the same form and interpretation. The distinction becomes relevant for lexical access; only lexical classes are taken as entry points to the lexicon.

determined by the order in which they appear in the list—more specific classes appear to the left of more general ones. Precedence of nondirect superclasses is derived by topologically sorting the hierarchy in a manner described in Section 2.6.

2.1.3 Main Equation Set. Following the superclass declaration are zero or more equations 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 that will be clarified in Section 3 below.

2.1.4 Variant Equation Sets. Following the (possibly empty) main equation set are zero or more sets of equations 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 3.2). 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 '|'.

2.1.5 String Concatenation. Construction and decomposition of complex words are carried out by the string concatenation operator '&&.' An equation of the form

```
String = Prefix && Suffix
```

unifies `String` with the result of concatenating `Prefix` and `Suffix`. The unification is nondeterministic, in the sense that if `String` is `abc`, and `Prefix` and `Suffix` are otherwise uninstantiated unification variables, the equation has the four solutions shown below, where '`""`' indicates the empty string:

(i) `Prefix : ""` `Suffix : abc`

(ii) `Prefix : a` `Suffix : bc`

(iii) `Prefix : ab` `Suffix : c`

(iv) `Prefix : abc` `Suffix : ""`

2.2 Other Aspects of the ELU Language

We introduce here a variety of notational conventions used in the examples below.

2.2.1 Lists. Values of attributes may take the form of lists of feature structures, notated in grammar rules and lexical entries as '[$F_1 \ldots, F_n$]'.

2.2.2 Disjunction. Disjunction is defined over atomic feature structures. A value defined as 'a/b/c' represents a set of atoms $D = \{a, b, c\}$, which unifies with: (i) any $d \in D$ (result: d); (ii) any other disjunction representing a set D' such that $D \cap D' = I$ (result: I if nonempty, otherwise failure); and (iii) the negation (see below) of a disjunction representing a set D' (result: $D - D'$ if nonempty, otherwise failure).

2.2.3 Negation. Negation is defined over atomic feature structures and disjunctions of atomic feature structures; a value defined as '~a' unifies with any feature structure F that does not unify with a, the result being F.

2.2.4 Path Specifications. Attribute value paths are notated in the form '<$a_1 \ldots, a_n$>.'

2.2.5 Variables. Unification variables have an initial uppercase letter; quoted expressions beginning with uppercase letters are interpreted as constants. Thus, `Var` is a variable, and both `atom` and `'Var'` are atomic FSs.

2.3 An Example

The following example illustrates the form and interaction of class definitions.

```
#Word  walk (Intransitive Verb)
    <stem> = walk

#Class Intransitive ()
    <subcat> = [Subj]       <Subj cat> = np

#Class Verb ()
    <aux> = no              <cat> = v
    |
    <tense> = past          <form> = <stem> && ed
    |
    <agr> = sg3             <tense> = present
    <form> = <stem> && s
    |
    <agr> = non_sg3         <tense> = present
    <form> = <stem>
```

The lexical class `walk` is declared as having two direct superclasses, `Verb` and `Intransitive`; its main set contains just one equation, which sets the value of the feature `<stem>` to be `walk`. `Intransitive` has no direct superclasses, and its main equation set assigns to the value of `<subcat>` a list with one element, a feature structure in which the value of `<cat>` is `np`. Neither `walk` nor `Intransitive` has any variant equation sets. `Verb`, by contrast, has three, in addition to two main set equations assigning default values for `<cat>` and `<aux>`. The three variants accounted for by this example are:

- the past tense verb, in which the value of `<form>` unifies with the result of concatenating the value of `<stem>` with the string 'ed,'
- the third person singular form of the present tense, in which the suffix string is 's,' and
- the form representing other combinations of person and number in the present tense; in the last case, the `<form>` value is simply identical to the `<stem>` value.

2.4 Multiple Inheritance

As we have just seen, a class may inherit from more than one direct superclass. Instead of a simple, tree-form taxonomic structure, therefore, the lexical hierarchy takes the form of a directed (acyclic) graph. In general, multiple inheritance of this kind necessitates more complex methods of searching a hierarchy; in the worst case, the number of paths to be traversed grows exponentially with the depth of the hierarchy. Moreover, since it then becomes possible to reach a node by more than one path, the presence of exceptions (via default assignments or negative links) permits incompatible

conclusions to be drawn—forks leading to such inconsistencies must be detected and suitable records kept. Much of the complexity of inheritance reasoners lies in finding and determining what to do in these cases of ambiguity.[5] 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 preferable to the simpler tree-structured hierarchy. If it proved possible to eliminate this source of complexity, the goal of a practical inheritance system would be nearer.

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. A French lexicon will distinguish three basic verb conjugations, exemplified by the transitive *aimer* ('love'), *haïr* ('hate'), and *voir* ('see'). Each of these three classes, not surprisingly, also contains intransitive verbs; in a tree-form hierarchy, nodes must be created for first conjugation transitive and first conjugation intransitive verbs, and similarly for the other cases, with information relevant to transitive verbs in general, or first conjugation verbs in general, repeated at each appropriate node. When the many subclasses and special cases of inflectional behavior are taken into account, together with a more adequate analysis of subcategorization patterns, the number of nodes required, and the redundancy of the specifications, increases considerably.

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 constraints relevant to different types of information are always disjoint, then the system as a whole is 'orthogonal' (Touretzky 1986: 73ff). The significance of this point lies in the fact that, if superclasses *B* and *C* of some class *A* are independent in this way, no conflict can arise when *A* inherits from both *B* and *C*, and the result of inheriting default information from the superclasses will not vary according to the order in which the defaults apply.

The organization of a lexicon must reflect two types of relation between classes of words. Clearly, certain classes stand in the subset/superset relation—'words' in general, 'verbs,' 'present tense verbs,' 'present tense verbs agreeing with a plural subject,' 'present tense verbs agreeing with a second person plural subject,' and so on. Equally clearly, within many of these classes there exist disjoint subsets—'present tense verbs,' 'past tense verbs,' 'infinitive verbs,' 'verbs agreeing with second person plural subject,' 'verbs agreeing with third person plural subject,' and so on. Word classes within the lexicon must encode both types of relation, and, in broad terms, it is the latter that one might wish to represent by means of unordered multiple inheritance.

The design of such a system presents a three-way choice: (i) to accept ambiguity of inheritance; (ii) to enforce orthogonality in the hierarchy; (iii) to eliminate unordered multiple inheritance. The present system takes the third option; ambiguity of inheritance is eliminated by enforcing a total ordering on the superclasses of any given class or word, and by making it 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 for unordered multiple inheritance, 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

5 Cf. examples of cascaded ambiguity and On-Path versus Off-Path preemption in Touretzky et al. (1987).

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 judiciously when required, but are responsible for determining when this should occur. In effect, we are giving away 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. This is no bad thing, since 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.

2.5 Default Inheritance

The use of default information in a unification-based system raises the question of how the two types of operation are to interact. If unification is employed in its conventional form, then combining conflicting information from different classes will produce failure and a null result. While this is reasonable behavior elsewhere, it prevents the desired treatment of exceptions to generalizations in word classes. One step toward a solution is to define default inheritance of constraints to be such that if A is a FS associated with a class C and B is a default FS associated with a superclass of C, then the result of applying B to A is $R = A \sqcup B$, if $R \neq \top$, and A otherwise.[6] This is not yet sufficient, since a failure of one subpart of B causes all other, possibly compatible, information in B to be discarded. What is needed is a scheme in which A unifies with as many as possible of the individual constraints embodied in B. We shall see in Section 3.2 how the casual "as many as possible" can be made precise.

2.6 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 is whether Nixon is a pacifist. In a conventional inheritance network, this problem might be represented by the configuration shown in Figure 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'?[7]

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

6 Throughout this paper, '$A \sqcup B$' denotes the unification of A and B, '$\top$' denotes the inconsistent feature structure equated with failure of unification, and '$\bot$' denotes the empty or most general feature structure, which subsumes all others.

7 If the links in question are strict rather than defeasible, a system may resolve the ambiguity by 'skeptical' or 'credulous' reasoning, yielding no answer or both answers, respectively. A mixture of strict and defeasible links (i.e., a 'heterogeneous' inheritance system) is argued by Ballim et al. (1988; 1990) to be more capable than a homogeneous system (one with only strict or only defeasible links) of expressing the intuitions underlying networks such as the "Nixon diamond." See also Section 4.3.

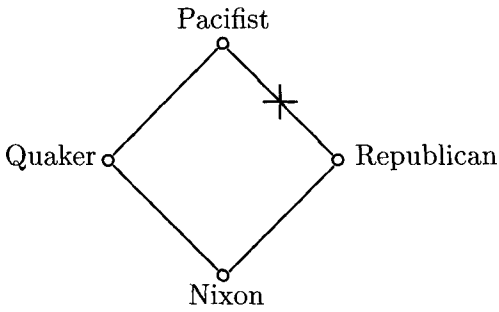

Figure 1
The Nixon diamond.

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 direct 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>` attribute 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 attributes `<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, i.e., in this simple case, on the order in which they appear in the superclass declaration; the declaration `(Quaker Republican)` states not only what the direct 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)`.

A frequent approach to such matters in the AI community is to arrange for a result where neither of the conflicting properties is inherited, so that, in the case of the present example, the value of `<pacifist>` is unspecified. A trivial implementation of this solution in the ELU lexicon would be to omit from the class definitions all mention of pacifisthood; the semantics of graph unification is such that an absence of information concerning some attribute is compatible with any possible value. If the

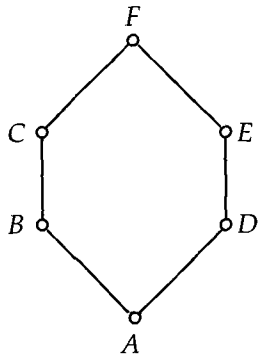

Figure 2
A partially ordered set of classes.

intention were to permit either of the values `yes` and `no`, but no other, then the value of `<pacifist>` could be supplied as a disjunction: `'yes/no.'`

Consider now a situation in which conflicting information occurs in nondirect superclasses, say *C* and *D* in the lexicon below. No superclass declaration contains both of these, and so the type of local ordering constraint imposed by the declarations in the 'Nixon' example is not available.

```
#Word  A (B D)     #Class B (C)     #Class C (F)
#Class D (E)       #Class E (F)     #Class F ()
```

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; we adopt some of the CLOS terminology here.[8] The 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. Informally, the CPL is created by performing a topological sort on the partial order: the effect can be seen most clearly in connection with a graphical representation such as that in Figure 2, which represents the partial order generated by the local superclass ordering constraints of the lexicon above.

Note that the left-to-right order of the two 'branches' in Figure 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 Figure 2 is therefore $\langle A, B, C, D, E, F \rangle$, with *A* being the most specific and *F* the most general.[9] 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.

A given CPL can be derived from several sets of class definitions; the lexicons shown below are some of those which, when compiled, will result in the same CPL

8 See Steele (1990: 782ff) for details of the algorithm, and Keene (1989: 118ff) for discussion.
9 The ordering relation of specificity on classes should not be confused with the subsumption relation on feature structures; the former is determined by superclass declarations and is independent of the information content of any constraints within the classes. Typically, more specific classes describe smaller sets of words.

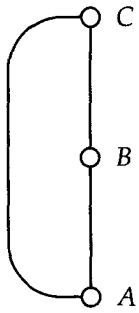

Figure 3
An impossible hierarchy.

$\langle A, B, C, D, E, F \rangle$, as the example above:

```
#Word  A (B)       #Class B (C)      #Class C (D)
#Class D (E)       #Class E (F)      #Class F ()

#Word  A (B)       #Class B (C)      #Class C (D)
#Class D (E F)     #Class E ()       #Class F ()

#Word  A (B)       #Class B (C)      #Class C (D E F)
#Class D (E)       #Class E (F)      #Class F ()

#Word  A (B C D)   #Class B (F)      #Class C ()
#Class D (E F)     #Class E ()       #Class F ()
```

The class precedence algorithm may thus be seen as defining an equivalence relation on lexical specifications.

Patterns of inheritance between superclasses of a lexical class are determined solely by the CPL. Note that this scheme excludes a number of configurations that have figured in the AI literature. Dual paths from one class to another, of the sort shown in Figure 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, $1 \leq i < j < k \leq n$. However, there is no prohibition on expressing the same information in two classes—this type of redundancy is harmless in that it does not complicate the process of searching the 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 and may introduce paradoxes.

Nor is there any means of expressing negative links of the kind shown in Figure 1.[10] The ELU lexicon is thus what Touretzky et al. (1987) term a *unipolar* system. 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 appears to enable exceptionality to be expressed without explicit negation of inheritance. The second reason is connected with the nature of graph 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

10 Negated inheritance links should not be confused with the negation introduced in Section 2.2, which is defined over the unification of atomic feature structures.

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. Finally, the defaulting behavior of the lexicon as a whole is determined by the CPLs of each lexical class, and these are in turn derived from the superclass declarations within class definitions. This imposition of a global organization by a collection of local constraints is attractively consistent with the spirit of unification-based formalisms.

3. Information in the Hierarchical Lexicon

Having discussed the form and organization of the lexicon, we continue with an account of the manner in which information distributed among its classes combines to characterize a set of lexical items.

3.1 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. The purpose of this condition is to preserve declarativeness; a set of equations that satisfy it may be applied without regard to order.[11] 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 that cannot be overridden, and multiple variant sets 'multiply' FSs, e.g., to produce different members of an inflectional paradigm.

The lexicon below provides a simple nonlinguistic illustration of how defeasible inheritance and nondefeasible inheritance interact in the lexicon.

```
#Word A (B)
    <p 1> = a
    |     <q 1> = r
    |     <q 1> = s

#Class B (C)
    <p 1> = b
    <p 2> = b
    |     <q 1> = s
```

11 See Section 4.2 for an example of the interaction of reentrancy and default unification.

```
#Class C ()
    <p 1> = c
    <p 2> = c
    <p 3> = c
    |    <q 2> = t
    |    <q 2> = u
```

The CPL of A is $\langle A, B, C \rangle$. Applying the single default equation of class A to the empty FS $\perp$ produces the FS shown as (1) below; the result of applying the two variant sets to this is the pair of FSs (2) and (3).

$$(1)\ \begin{bmatrix} \text{P} & [\,1\ \text{a}\,] \end{bmatrix} \qquad (2)\ \begin{bmatrix} \text{P} & [\,1\ \text{a}\,] \\ \text{Q} & [\,1\ \text{r}\,] \end{bmatrix} \qquad (3)\ \begin{bmatrix} \text{P} & [\,1\ \text{a}\,] \\ \text{Q} & [\,1\ \text{s}\,] \end{bmatrix}$$

Since there is no conflict between $\perp$ and the information in class A the situation is straightforward; two FSs are created, one for each of A's variant sets. Class B then applies to each of these, but in this case conflicts do arise. The first is between the values of P:1 present in (2) and (3) and that specified in the default set of B. The default constraint is overridden by the information contributed by the more specific class A, while the equation providing the value of P:2 succeeds, producing the FSs (4) and (5).

$$(4)\ \begin{bmatrix} \text{P} & \begin{bmatrix} 1 & \text{a} \\ 2 & \text{b} \end{bmatrix} \\ \text{Q} & [\,1\ \text{r}\,] \end{bmatrix} \qquad (5)\ \begin{bmatrix} \text{P} & \begin{bmatrix} 1 & \text{a} \\ 2 & \text{b} \end{bmatrix} \\ \text{Q} & [\,1\ \text{s}\,] \end{bmatrix}$$

The second conflict in class B arises from the single variant set, which permits only the value **s** for Q:1. This is a strict constraint, and, while (5) satisfies it, (4) does not. Only (5), therefore, is considered by class C.

As before, the default set of C contains constraints (those involving P:1 and P:2) that are overridden by existing information; the third default equation succeeds, and yields (6). The two variant classes each add a value for Q:2, producing the FSs (7) and (8).

$$(6)\ \begin{bmatrix} \text{P} & \begin{bmatrix} 1 & \text{a} \\ 2 & \text{b} \\ 3 & \text{c} \end{bmatrix} \\ \text{Q} & [\,1\ \text{s}\,] \end{bmatrix} \qquad (7)\ \begin{bmatrix} \text{P} & \begin{bmatrix} 1 & \text{a} \\ 2 & \text{b} \\ 3 & \text{c} \end{bmatrix} \\ \text{Q} & \begin{bmatrix} 1 & \text{s} \\ 2 & \text{t} \end{bmatrix} \end{bmatrix} \qquad (8)\ \begin{bmatrix} \text{P} & \begin{bmatrix} 1 & \text{a} \\ 2 & \text{b} \\ 3 & \text{c} \end{bmatrix} \\ \text{Q} & \begin{bmatrix} 1 & \text{s} \\ 2 & \text{u} \end{bmatrix} \end{bmatrix}$$

Intuitively, (7) and (8) represent two alternative sets of information that the lexicon associates with the lexical class A.

It is useful to have some terminology for the various steps we have just followed. We shall refer to the result of applying to some FS F the set of default equations in a class C as the *default extension* of F with respect to C. (1) above is thus the default extension of $\perp$ with respect to A, and (5) the default extension of (2) with respect to B, for example. Similarly, we refer to the result of applying both default and variant set information in a class C to some FS F as the *superclass extension* of F with respect to C. The set containing (2) and (3) is then the superclass extension of (1) with respect to A, that consisting of just (5) is the superclass extension of (2) with respect to B, and the set containing (7) and (8) is the superclass extension of (5) with respect to C. Finally, since C is the most general superclass of A, (7) and (8) together comprise the *global extension* of the lexical class A.

Figure 4 shows the pattern of inheritance arising from this lexicon; classes are labeled 'A,' 'B,' and 'C,' and within each the default and variant sets are identified as 'd' and 'v,' respectively. Arcs are labeled with the names of FSs shown above.

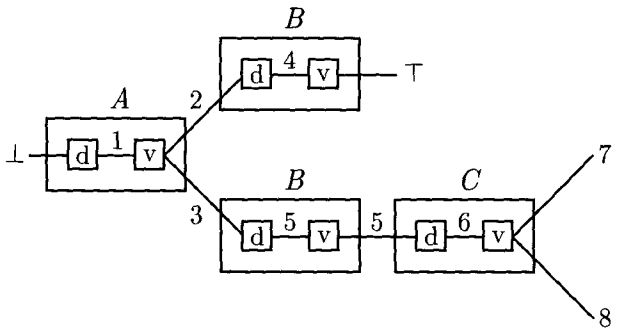

Figure 4
Feature structures characterized by the lexicon.

3.2 Definitions

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 the auxiliary notions *default extension* and *superclass extension.*

3.2.1 Default Extension. Let ϕ be a FS, and $R(\phi)$ the restriction of ϕ to reentrant paths, i.e., the most general FS such that $\forall p, q\,[\phi(p) \equiv \phi(q) \rightarrow R(\phi)(p) \equiv R(\phi)(q)]$.[12] Then the *default extension* of a FS ϕ with respect to a set of FSs Ψ is

$$\phi \sqcup \bigsqcup\{\psi \in \Psi \mid \phi \sqcup \psi \neq \top\}$$

if both $R(\phi) \sqcup \bigsqcup \Psi \neq \top$ and $\phi \sqcup R(\bigsqcup \Psi) \neq \top$, and $\top$ otherwise.

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 behavior of the lexicon.

The condition referring to reentrancy 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 is applicable, is unchanged, i.e., identical to ϕ; otherwise the lookup fails.[13]

3.2.2 Superclass Extension. The *superclass extension* S 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

$$\mathrm{S}(\phi, C) = \{\psi \mid 1 \leq i \leq n \wedge v_i' \sqcup \phi' = \psi \wedge \psi \neq \top\},$$

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.

$\mathrm{S}(\phi, C)$ 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.

12 Here, '$\phi(p)$' denotes the value of the attribute-value path p in the FS ϕ, and '$\equiv$' denotes token identity of its operands.

13 See Section 4.1 for a different approach to order sensitivity proposed by Carpenter (1991).

3.2.3 Global Extension. The *global extension* of a lexical class L having the CPL $C = \langle c_1, \ldots c_n \rangle$ is G_n, where $G_0 = \{\bot\}$, and

$$G_{i>0} = \bigcup \{\Psi \mid \forall \phi \in G_{i-1}, \Psi = S(\phi, c_i)\}.$$

To speak in procedural terms, $\bot$ is the empty FS that 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 $\bot$ the CPL of L.[14]

The set of lexical items admitted by a lexicon consists of the union of the global extensions of all lexical classes in the lexicon.

3.3 Variant Sets—Discussion

Variant sets may be thought of as representing a restricted form of disjunction over complex FSs. Kasper and Rounds (1986) show general disjunctive unification to be intractable, since it involves an exponentially complex step of expansion to disjunctive normal form. Note, however, that the alternation embodied in variant sets is guaranteed to be at the 'top level' of FSs only; that is to say, the FSs described by multiple variant sets are already in disjunctive normal form, apart from any atomic disjunctions they may contain. For this reason, multiple variant sets impose no mutual constraints; regardless of whether the information they contain is orthogonal, complementary, or conflicting, the crucial factor is the unifiability of each variant set individually with the single candidate FS.

The variant set mechanism tends to suggest a close connection between strict constraints and multiple realizations of class membership. Clearly, a class may have a single variant set, so nondefeasible information does not entail alternation. On the other hand, it may at first appear that the system forces a style of analysis in which, for example, all members of an inflectional paradigm have equal status. In many cases, this will be undesirable. Consider once more the class of regular English verbs; instead of encoding their inflectional behavior by means of class `Reg_Verb_Present` below, one might well wish to express the generalization that five out of the six present tense forms are identical, and treat the sixth as an exception.

```
#Class Reg_Verb_Present (Verb)
    |
    <agr> = sg1        <form> = <stem>
    |
    <agr> = sg2        <form> = <stem>
    |
    <agr> = sg3        <form> = <stem> && s
    |
    <agr> = pl1        <form> = <stem>
    |
    <agr> = pl2        <form> = <stem>
    |
    <agr> = pl3        <form> = <stem>
```

14 The fact that G_i is defined here recursively in terms of G_{i-1} should not be misinterpreted as reflecting a 'top-down' view of inheritance—the default unification performed in order to compute the default extension of a FS requires a bottom-up view, in which more specific classes are inherited from before more general ones.

A more attractive analysis would involve the use of a class that encodes in one variant set just the exceptional case, and in a second provides sufficient information to exclude that exception from a default stated in a more general class.

```
#Class Reg_Verb_Present_Exception (Verb)
    |
    <agr> = sg3          <form> = <stem> && s
    |
    <agr> = sg1/sg2/pl1/pl2/pl3

#Class Verb ()
    <form> = <stem>
```

The lexicon will generally be called upon to establish a pairing of morphosyntactic information (tense, case, agreement, etc.) and phonological or graphological information (the form taken by the variant of a word that expresses a particular morphosyntactic property). The two kinds of information will normally be expressed as the values of distinct attributes, in which case the semantics of default unification set out in Section 3.2 will not enforce the intended pairing. The strict unification applying within variant sets causes equations to be interpreted in the conventional way for unification-based linguistic descriptions, i.e., as conjunctive constraints on feature structures. In practice, it is tempting to think of the lexicon as operating in a directional manner. Looking up a word involves retrieving from the lexicon the collection of morphosyntactic properties expressed by a given word form, and here the natural informal interpretation will be that the existence of that wordform implies the presence of those properties. Generating a word, on the other hand, involves retrieving the word form that expresses a given set of morphosyntactic properties, and here the natural informal interpretation will be the reverse. Nevertheless, the system retains the nondirectionality of unification, and this informal view of its operation is no more than a convenient approximation to its underlying semantics.

Since strict constraints are able to filter out unwanted intermediate solutions, computing the global extension of a lexical class will in general involve creation of a larger number of FSs than appear in the result. If $v(c)$ is the number of variant sets in the class c, and f is the function in the natural numbers such that $f(0) = 1$ and $f(k) = k$ if $k > 0$,[15] the maximum number of FSs constructed for a lexical class with the CPL $\langle c_1, \ldots c_n \rangle$ is given by

$$\prod_{i=1}^{n} f(v(c_i)).$$

This maximum arises when no failures of strict unification occur.

Finally, while the variant set mechanism provides some of the functionality of the lexical rules proposed by Flickinger et al. (1985), Calder (1989), and others, the structure of the ELU lexicon does not admit the full range of capabilities of these more powerful devices; cyclic rule applications cannot be simulated, for example.

4. Some Comparisons

As we mentioned briefly in Section 1, the idea of applying inheritance to the lexicon, and indeed to linguistic descriptions in general, is not a new one. Here we consider a few of the proposals that have appeared in recent years.

15 Classes with one or no variant set yield at most a singleton superclass extension for each FS they apply to, while others multiply a FS by a factor of no more than their total number of variant sets.

4.1 Defaults and Reentrancy

The notion of 'default extension' employed here resembles the 'priority union' of Kaplan (1987: 180f) and the 'conservative addition' of Shieber (1986b). As Bouma (1990) points out, the result of defaulting under this approach may depend on order of application. To take Bouma's (p. 166) simple example, default unification of the FSs

$$(1)\ \begin{bmatrix} \text{F} & \text{a} \end{bmatrix} \qquad (2)\ \begin{bmatrix} \text{G} & \text{b} \end{bmatrix}$$

with a re-entrant structure

$$(3)\ \begin{bmatrix} \text{F} & \boxed{1} \\ \text{G} & \boxed{1} \end{bmatrix}$$

produces different, nonunifying, results:

$$(4)\ \begin{bmatrix} \text{F} & \boxed{1}\ \text{a} \\ \text{G} & \boxed{1} \end{bmatrix} \qquad (5)\ \begin{bmatrix} \text{F} & \boxed{1}\ \text{b} \\ \text{G} & \boxed{1} \end{bmatrix}.$$

(4) results when (1) applies before (2), and (5) when (2) applies before (1).

Similarly, defaulting the FSs

$$(6)\ \begin{bmatrix} \text{F} & \boxed{1} \\ \text{G} & \boxed{1} \end{bmatrix} \qquad (7)\ \begin{bmatrix} \text{G} & \text{b} \end{bmatrix}$$

onto a FS

$$(8)\ \begin{bmatrix} \text{F} & \text{a} \end{bmatrix}$$

produces one of

$$(9)\ \begin{bmatrix} \text{F} & \boxed{1}\ \text{a} \\ \text{G} & \boxed{1} \end{bmatrix} \qquad (10)\ \begin{bmatrix} \text{F} & \text{a} \\ \text{G} & \text{b} \end{bmatrix},$$

according to whether (6) applies before (7), in which case the result is (9), or (7) applies before (6), to produce (10).

Both situations are avoided in the ELU system by making applicability of defaults conditional on the unifiability of reentrant subsets of the default information and candidate structure (see Section 3). The first case, where the candidate FS (3) contains a reentrancy that has nonunifying extensions in the default information (1 and 2), is excluded by the requirement that $R(3) \sqcup \bigsqcup\{1,2\} \neq \top$, and the second case, where a reentrancy in a default FS (6) has nonunifying extensions in the candidate FS (8) and other default FSs (7), is excluded by the requirement that $(8) \sqcup R(\bigsqcup\{(6),(7)\}) \neq \top$.

Carpenter (1991) proposes a more permissive view of default unification in which conflicts of this type are resolved in one of two modes, 'credulous' and 'skeptical.' Credulous default unification preserves the maximum possible default information, and is defined so as to produce a (nonempty) set of solutions. For example, the result of credulously defaulting (11) (the unification of 6 and 7 above) onto (8) is the set of solutions (12) (i.e., 9 and 10).

$$(11)\ \begin{bmatrix} \text{F} & \boxed{1}\ \text{b} \\ \text{G} & \boxed{1} \end{bmatrix} \qquad (12)\ \left\{ \begin{bmatrix} \text{F} & \boxed{1}\ \text{a} \\ \text{G} & \boxed{1} \end{bmatrix}, \begin{bmatrix} \text{F} & \text{a} \\ \text{G} & \text{b} \end{bmatrix} \right\}$$

Skeptical default unification retains only consistent information, and is defined as the generalization of such a set of solutions.[16] Skeptically defaulting (11) onto (8) produces

16 The generalization of two FSs corresponds to their greatest lower bound in the subsumption lattice. See Karttunen (1984).

the FS (13), in which the only default information to have been preserved is that concerning the presence of an attribute G, whose value, since it conflicts with that in (8), is unspecified:

$$(13)\ \begin{bmatrix} \text{F} & \text{a} \\ \text{G} & \bot \end{bmatrix}$$

Neither of these proposals for the treatment of inconsistent default information is equivalent to ours. Of the various properties of skeptical and credulous default unification noted by Carpenter, the ELU lexicon shares the following.

(i) Strict information is preserved—i.e., default unification is monotonic (see Section 4.2).

(ii) Default unification reduces to standard unification if the two FSs involved are consistent.

(iii) Default unification is finite.

The fourth property, that skeptical and credulous default unification are both well defined in all cases, is not shared by the present system—as we have seen above, the definition of default extension (Section 3.2) produces no solution for unifications of the type illustrated here.

4.2 Monotonicity

The 'template' facility of PATR-II (Shieber 1986a: 55ff) allows inheritance within the lexicon. Lexical entries may be defined by means of templates that have other templates in their definitions. Thus, if the definition of a template T_1 mentions the templates T_2 and T_3, any entry in which T_1 appears also receives the contents of T_2 and T_3. This is a form of strict multiple inheritance, since, if T_2 and T_3 contain conflicting information, unification fails and no corresponding lexical item is created. Shieber (1986a: 59ff) also describes a type of default inheritance implemented in terms of a noncommutative 'overwriting' operation, in which constraint equations may be formulated so as to take precedence over existing structures. These devices permit the construction of heterogeneous systems like that of ELU; one difference between the two would be that, in Shieber's scheme, the statements exhibiting default behavior are ones that have the ability to override others over which they have precedence, whereas in the approach described above the default statements are ones that can be overridden by others that have precedence over them.

In the D-PATR environment presented by Karttunen (1986), all templates engage in defaulting: "... templates and other specifications that occur [in lexical entries] are processed sequentially from left to right. Each item is compiled to a directed graph and superimposed on the graph previously compiled" (p. 76). Here, then, defaulting occurs as standard; any sequence of statements in a lexical entry may be such that a later statement conceals the effect of an earlier one. Again, the fact that defaulting is interpreted as the overwriting of one structure by another means that structures may be modified in a nonmonotonic fashion.

The implementation of defaults in ELU contrasts with these approaches in that the system 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 $\langle c_{i+1}, \ldots c_m \rangle, i < m \leq n$.[17] This follows directly from the definition of

17 'Apply' here is used in the sense introduced in Section 3.2.

"default extension" given in Section 3.2. Each individual constraint in a default set contributes information to a FS F if and only if that information is consistent (i.e., unifies) with the existing contents of F. Inconsistent information is ignored, but since it is only inconsistent information that would be capable of destructively modifying F, by removing or replacing some of its content, the consequence is that F subsumes its default extension with respect to any set of constraints; moreover, since the other operations involved in determining superclass extension employ only conventional 'strict' unification, these are equally unable to introduce into the system any element of nonmonotonicity.

4.3 Other Issues

Inheritance hierarchies are frequently employed in object-oriented programming languages and environments. Some work in computational linguistics has been explicitly object-oriented in nature (Daelemans 1990; De Smedt 1984; De Smedt and de Graaf 1990) but, as Gazdar (1990) observes, this has tended to adapt existing general-purpose techniques and languages, rather than attempt to devise specialized linguistic formalisms. Daelemans (1990) suggests that inheritance can replace unification as a basic mechanism for linguistic description, but his proposals remain relatively undeveloped.

In some ways, the use of typed FSs and other data structures in ELU represents a step toward a rapprochement of unification and object-oriented techniques. However, object-oriented programming involves much more than the use of inheritance, and the lack of mechanisms in ELU implementing encapsulation of data or message-passing protocols means that it would be a mistake to classify the work presented here as a variety of object-oriented linguistic description.

DATR (Evans and Gazdar 1990) is a language designed for the definition of inheritance networks for use in lexical specification. Its syntax is similar to ELU's in that paths and their values are grouped together into compound statements, in DATR called 'nodes,' but at a less superficial level the two systems are quite different. An example will make some of the differences clear. The theory below defines two nodes, `A` and `B`:

```
A: <p1> == v1
   <p2a p2b> == v2
   <p3> == B.

B: <p3> == v3.
```

The values of `<p1>` and `<p2a p2b>` in `A` are set locally, while the value of `<p3>` is inherited from `B`. Inheritance of values is thus specialized to individual paths in DATR, rather than applying uniformly to entire classes as in ELU. Moreover, the defaulting mechanism of DATR is independent of the inheritance mechanism, and based on path extension; an equation of the form '$\langle a_1, \ldots a_i \rangle$ == v' allows v to be inferred as the value of all paths $\langle a_1, \ldots a_k \rangle$, $1 \leq i < k$, such that no initial subpath $\langle a_1, \ldots a_j \rangle$, $i < j \leq k$, has explicitly been assigned a different value. As a consequence, the objects manipulated by DATR are strictly linear, in contrast to the more general feature structures that form the basis of the ELU lexicon[18]—in practice, path-value equations in DATR tend to correspond to variant sets in ELU.

Flickinger et al. (1985) present a model of the lexicon based on multiple default inheritance in the frame language HPRL (Rosenberg 1983). Two inheritance modes

18 This is not to say that the two are not interconvertible—see Kilbury et al. (1991).

exist: 'normal,' a conventional shortest-path default method in which a slot is filled by information from the most specific accessible node in the hierarchy, and 'complete,' by which multiple values for a slot are obtained from a number of more general frames. Values for the FEATURES slot for the word *has* thus include (AUX PLUS) inherited from AUXILIARY, (AGREEMENT 3RD-SING) from THIRD-SING, and (CONTROL SSR) from SUBJECT-RAISE. The values in this example are compatible; the question of contradictions in complete-mode inheritance is raised, but it is not stated whether any that do occur are resolved, produce inconsistent results, or cause the search to fail. Flickinger (1987: 60) refers to constraining the hierarchy "in such a way that each single-valued attribute of some given class or member [is] assigned a value by at most one of the immediate superclasses, or its parents, so conflicting values [cannot] occur," but does not indicate whether this constraint is to be imposed automatically as part of a compilation or consistency checking process, or is simply to take the form of advice to users. He considers as an alternative a rule of inheritance "which for each attribute assigns priority to some one of the parent classes," and a convention similar to that of the ELU lexicon, whereby precedence of direct superclasses is encoded in the order of elements in a list.

The system described by Padgham (1988) resembles the ELU lexicon in taking the objects in the hierarchy as being sets of properties, partitioned into strict and defeasible subsets: "The *type core* includes those characteristics which we regard as always present in objects of this type .... The *type default* contains the information for typical objects of that type." Padgham's types may also be partially specified, and thus bear quite a close resemblance to ELU classes with single variant sets. Where this arrangement differs from the ELU lexicon is in permitting inheritance links between types to involve just these substructures; to represent the fact that instances of type T_1 are always typical instances of type T_2, a link connects the core of T_1 and the default of T_2, and so on. Negative relations between core and default substructures may also be expressed.

Inheritance hierarchies have chiefly been studied within two overlapping but conceptually distinguishable subareas of artificial intelligence: knowledge representation and commonsense reasoning. The latter of these domains is characterized by concerns that extend beyond those of the present work. For example, problems of reasoning with possibly incomplete knowledge do not arise in the context of lexical organization, since for any query the writer of a lexicon can be assumed to have envisaged the entire conclusion space and all factors required for computing it.

5. Example Analyses

This section contains some longer linguistic examples illustrating aspects of the ELU lexicon introduced above. While these analyses are correct in the sense that they characterize sets of feature structures that achieve the desired pairing of word forms and morphosyntactic properties, it should of course be borne in mind that each represents just one treatment among the many possible.

5.1 English Irregular Verbs

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.[19] There are exceptions to the general rule, however, and inheritance mechanisms must be sufficiently flexible to permit deviation from this pattern.

Consider the small class of English verbs including *dream, lean, learn,* and *burn;* these have, for many speakers, alternate past finite and past participle forms: e.g., *dreamed* and *dreamt*. The following fragment produces the correct analyses, in which the value of <morph> expresses inflectional information, and that of <form> is the corresponding word form.[20]

```
#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 ()
      <cat> = v
      <p_fin_form> = <stem> && ed
      <psp_form> = <stem> && ed
      |
      <morph> = present_sg3          <form> = <stem> && s
      |
      <morph> = present_nonsg3       <form> = <stem>
      |
      <morph> = pastfinite           <form> = <p_fin_form>
      |
      <morph> = pastnonfinite        <form> = <psp_form>
```

The lexical class of a regular verb such as walk contains a minimum of idiosyncratic information, the value of the feature <stem>. Its direct superclass, Verb, contributes the information that the value of <cat> is v, and that the values of both <p_fin_form> and <psp_form> are the result of concatenating the <stem> value and ed. In addition to this default information, Verb has four variant sets, corresponding to the four combinations of <morph> and <form> values admitted by the fragment. Finite and nonfinite past forms are both associated via the values of <p_fin_form> and <psp_form> with the string *walked*, the third person singular present tense variant is assigned the string *walks*, and the word form representing the other present tense variants is *walk*.

Irregular verbs differ in having the two past forms specified in their lexical class. The defeasible constraints providing values for <psp_form> and <p_fin_form> in Verb

19 Blocking is a consequence of the architecture proposed by Calder (1989) and of the DATR language.

20 The analysis sketched here is simplified in the sense that several variants within the Verb class have been omitted, all inflectional information is embodied as the value of the single feature <morph>, and no account is given of the various spelling alternations that accompany suffixation.

are overridden by this more specific information, so that while *sink* behaves like regular verbs in its present tense, the values of <form> for the past finite and past nonfinite variants are *sank* and *sunk*.

Verbs like *dream* have DualPast as a direct superclass, with higher precedence than Verb. DualPast has two variant sets, the first of which assigns as values of <psp_form> and <p_fin_form> the concatenation of the <stem> value and t, giving *dreamt*, etc. The second variant set is empty (recall that variant sets are preceded by the vertical bar '|') and the absence of contradictory specifications in this second variant permits the equations in the main set of Verb to apply. In addition to specifying exceptional properties, therefore, the definition of DualPast also permits inheritance of properties from more general classes; among these properties is that shared by verbs like *walk* of forming the two past forms by suffixing *ed* to the stem, which produces the regular (*dreamed*, etc.) past forms.

5.2 German Separable Verbs

Two large classes of German verbs are the separable- and inseparable-prefixed compound verbs; their analysis involves some quite intricate interactions of morphological, syntactic, and semantic information. Separable verbs are of interest syntactically because, as their name suggests, the prefix is a bound morpheme only in certain syntactic environments, namely when the verb is untensed or head of a verb-final clause.[21] Members of both classes share morphological, but not necessarily syntactic or semantic, properties of the verb that corresponds in form to their stem. The separable-prefix verb *weglaufen* ('run away') and inseparable *verlaufen* ('elapse') are two such verbs that the lexicon should be able to relate to their apparent stem *laufen* ('run'), assigning them similar morphological properties—*laufen* then becomes the model for a family of verbs. Morphosyntactic peculiarities of the separable verbs include the form of the past participle, in which the normal past participle prefix *ge-* is preceded by the separable prefix (e.g., *weggelaufen*), and the existence of an infinitive form, in which the separable prefix is followed by *zu* (e.g., *wegzulaufen*); the infinitive of other verbs is formed phrasally (from the perspective of written language, at least) with the particle *zu* as a separate word. Past participles of inseparable-prefix verbs omit the *ge-* prefix.

As in the example of Section 5.1, lexical classes encode just the idiosyncratic properties of verbs. Here, for compound verbs related to *laufen*, only their semantics is specified at this level; other properties are inherited from classes associated with the prefix and stem. The class laufen, in addition, specifies that the form of the past tense stem for this family of verbs is *lief*.

```
#Word  laufen (VStem Strong)
       <sem> = laufen
       <morph past_stem> = lief

#Word  weglaufen (Weg laufen)
       <sem> = weglaufen

#Word  verlaufen (Ver laufen)
       <sem> = verlaufen
```

The class Ver inherits from NonSeparable, where values for <morph psp prefix> (the form of prefix for the past participle) and <morph prefix> (the form of prefix for other

21 Within the syntactic analysis assumed here, their distribution is controlled by a binary feature <syn inv>, whose value in these contexts is no, and elsewhere is yes.

variants) are instantiated to the value of <prefix> provided in Ver—the past participle of *verlaufen* is thus *verlaufen*, etc.

```
#Class Ver (NonSeparable)
       <prefix> = ver

#Class NonSeparable ()
       <morph prefix> = <prefix>
       <morph psp prefix> = <prefix>
```

The behavior of separable verbs is more complex, and involves a number of variant sets in the Separable class. The first of these accounts for the prefix occurring in the past participle, which is the result of concatenating the value of <morph prefix> and ge—the past participle of *weglaufen* is thus *weggelaufen*. The second variant set accounts similarly for the infinitive form *wegzulaufen*. The third accounts for all other forms of a separable verb in which the prefix is attached; these are just the forms for which the sentence grammar assigns the value no to the <syn inv> attribute, with the exception of the past participle and infinitive, which receive special treatment. The fourth variant accounts for the other cases, i.e., those finite verbs in which the prefix is detached, and for which the sentence grammar assigns to <syn inv> the value yes. Here, the <syn prefix> attribute encodes the form of the detached prefix for use by the sentence grammar.

```
#Class Weg (Separable)
       <prefix> = weg

#Class Separable ()
       |
       <syn infl> = psp
       <morph psp prefix> = <prefix> && ge
       |
       <syn infl> = inf
       <morph prefix> = <prefix> && zu
       |
       <syn infl> = ~psp/inf
       <syn inv> = no
       <morph prefix> = <prefix>
       |
       <syn infl> = pres_sg_3/past_sg_3
       <syn inv> = yes
       <syn prefix> = <prefix>
```

The rôle of the class VStem is rather different; it serves to identify the substring of a verb's citation form that corresponds to its "base stem." For the majority of verbs, whose citation form ends in *-en*, this is what results when any prefix (with the empty string as default) and the final *-en* are detached. For the minority of verbs whose citation form ends in *-eln* or *-ern*, only the final *-n* is detached. Recall that the citation form of verbs in this lexicon coincides with the value of the <sem> attribute.

```
#Class VStem (Verb)
       <prefix> = ''
       |
       <sem> = <prefix> && <morph bse stem> && en
       |
       <sem> = <prefix> && <morph bse stem> && n
       <sem> = _ && ln/rn
```

The lexical class `laufen` inherits directly, and `weglaufen`, etc., indirectly, from the `Strong` class. Some strong verbs undergo a stem alternation known as 'Umlaut' in certain parts of their paradigm—the third person singular of the present tense for *laufen* is *läuft*, for example. This modification is performed here by the macro call in the main set of the `Strong` class; `Umlaut` is defined elsewhere as a macro that employs the string concatenation operator described in Section 2.1 to segment the string passed as its first argument, change the vowel, and reassemble the components to produce its second argument. Strong verbs form their past participle with the suffix *-en*, where weak verbs have *-t*. Their second and third person singular present tense forms may undergo vowel modification as described above, and the past tense forms are irregular, with a zero suffix for the first and third person singular.

```
#Class Strong (VInfl Verb)
       |
       <syn infl> = psp
       <morph psp suffix> = en
       |
       !Umlaut(<morph bse stem>,<morph stem>)
       <syn infl> = pres_sg_3
       |
       <syn infl> = past_sg_3
       <morph suffix> = ''
       <morph stem> = <morph past_stem>
       |
       <syn infl> = ~psp/pres_sg_3/past_sg_3
```

Other members of the paradigm are described by the fourth variant set of `Strong`, the function of which is not to add information, but rather to permit them to inherit from `VInfl` properties common to weak verbs. `VInfl` contains default statements concerning the past participle form: unless contradicted by more specific information, this is to be constructed from the prefix *ge-*, the suffix *-t*, and the value of `<morph bse stem>` provided in `VStem`. The weak verb *warten*, for example, has the past participle *gewartet*. The 'base' or 'bare infinitive' form is constructed similarly, and the suffix for the third person singular of the present tense is stated to be *-t*, to give *läuft*, etc. As before, the fourth variant set permits members of the paradigm not treated explicitly in this class to inherit from `Verb`.

```
#Class VInfl (Verb)
       <morph psp prefix> = ge
       <morph psp suffix> = t
       <morph psp stem> = <morph bse stem>
       <syn infl> = psp/bse/pres_sg_3/past_sg_3
       |
       <syn infl> = psp
       <form> = <morph psp prefix> &&
                <morph psp stem> && <morph psp suffix>
       |
       <syn infl> = bse
       <form> = <morph prefix> &&
                <morph bse stem> && en
       |
       <syn infl> = pres_sg_3
       <morph suffix> = t
       |
       <syn infl> = ~psp/bse/pres_sg_3
```

The most general class, Verb, establishes default values for prefix, stem, and suffix attributes, and states that the default <form> value is the result of concatenating whatever values these attributes have been assigned, either in this class or in more specific ones. It also states that, by default, a verb has no detached separable prefix; tensed separable verbs are exceptions to this, and receive their <syn prefix> value in the Separable class. The two variant sets in Verb serve to restrict the syntactically relevant <syn inv> attribute to a no value for the base, past participle, and infinitive forms, while leaving tensed verbs able to take either value.

```
#Class Verb ()
       <cat> = v
       <syn prefix> = none
       <morph stem> = <morph bse stem>
       <morph prefix> = ''
       <morph suffix> = en
       <form> =
         <morph prefix> && <morph stem> && <morph suffix>
       |
       <syn infl> = bse/psp/inf
       <syn inv> = no
       |
       <syn infl> = ~bse/psp/inf
```

6. Implementational and Practical Issues

The introduction stated three main requirements imposed by the context within which the ELU lexicon system is used: the ability to both analyze and generate word forms, good integration with other components that interface to the lexicon system (for example parsers and generators), and the ability to accommodate a large number of words. Our implementation, though relatively straightforward, achieves these goals, making the system genuinely useful in practice; this section briefly discusses some of the implementational issues involved.

6.1 Analysis and Generation of Word Forms

In Section 3, the set of lexical items admitted by a lexicon was defined formally as the union of the global extensions of all lexical classes in the lexicon. This definition suggests an obvious (bottom-up) implementation of the analysis and generation of word forms: collecting up all the lexical classes in the lexicon, and computing the union of their global extensions, in analysis returning only the FSs with wordform attribute the same as the word form being analyzed, and in generation returning only the word forms whose FSs unify with the structure being generated from.

This form of exhaustive search is computationally expensive for all but the smallest lexicons—however, it does allow all unification and defaulting operations to be traced, so that when developing a lexicon (and working with only a representative sample of lexical classes) the user can precisely monitor the effects of adding new classes and modifying old ones. The tracing information is presented in an order and a manner that appears natural to a user who understands global extension.

6.2 Efficiency

While a lexicon is being debugged, all the words and classes in it are held in main memory, and analysis and generation are performed using exhaustive search, as described above. Once the lexicon is fully debugged, this approach has three major drawbacks: since they are ordinary text files, large lexicons take a long time to load

into the system, once in memory the representations of the classes in them occupy a lot of space (increasing process size and swapping and garbage collection time), and exhaustive search over all the lexical classes on every lookup is too inefficient for practical purposes.

To solve these problems, we have implemented a facility that indexes a lexicon, storing all the information in it in a machine-oriented format in files on disc. The format is such that the information can be retrieved quickly with minimal processing. One index file holds all the classes in the lexicon, each superclass in a class's CPL being stored as an integer representing the byte position in the file at which its definition starts. During the indexing process, for each lexical class, all the lexical items admitted by the class are generated. The index file byte position of the class is associated with each distinct word form in the lexical items.[22] This position is also associated with the name of each semantic relation occurring in the lexical items. At the end of indexing these pairings are lexicographically sorted and written out to index files. Analysis works by searching for the word form (using a binary chop on the word form index file), retrieving the byte positions associated with the form, fetching the lexical classes at these positions together with their superclasses, and performing an exhaustive lookup (as described above) using just these lexical classes. Generation is similar but instead uses the semantic relation index.

If there are n possible word forms implicit in a lexicon, then analyzing one form will take time proportional to $c \cdot log_2(n)$ where c is a small constant, plus in the average case a constant time for lookup with the candidate lexical classes. Generation similarly will take time proportional to log_2 of the total number of semantic relations in the lexicon plus a constant. In practice, the size of the lexicon makes little difference to lookup time; what really matters is the number of classes in the CPLs of lexical classes and their complexity.

6.3 Integration

The ELU lexicon system is fully integrated with the rest of ELU. The fact that the lexicon is based on multiple default inheritance is transparent to the analyzer and generator (and even to the casual user). Indeed, ELU supports another type of lexicon based on finite state morphology; lexicons of this type are accessed via the same interface, and one may be loaded and available for use at the same time as an inheritance-based lexicon.

One of the characteristics of the ELU lexicon system that has allowed it to be so well integrated with the other components in the system is that it represents feature structures in exactly the same way that they do. In particular, ELU does not have to convert from one representation to another at the interface to the lexicon, as does DATR when used within a unification grammar system (Kilbury et al. 1991).

7. 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

22 In theory, the system can deal with lexicons admitting up to around 4,000,000 word forms, but the largest lexicon tested to date contains slightly more than 30,000 entries.

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 Section 2, ambiguity and negation are not essential in the context of a unification-based lexicon.

Acknowledgments

This paper is an expanded version of Russell et al. (1990). We are indebted to Mark Johnson for valuable comments on that earlier work, and to *Computational Linguistics* referees for their suggestions. Participants at the ACQUILEX workshop on default inheritance held in Cambridge in April 1991 also made useful suggestions.

References

Ballim, A.; Candelaria de Ram, S.; and Fass, D. (1990). "Reasoning using inheritance from a mixture of knowledge and beliefs." In *Knowledge Based Computer Systems: Proceedings of KBCS '89* (Lecture Notes in Artificial Intelligence no. 444), edited by S. Ramani, R. Chandrasekar, and K. S. R. Anjaneyulu, 387–396. Springer-Verlag.

Ballim, A.; Fass, D.; and Candelaria de Ram, S. (1988). "Resolving a clash of intuitions: Utilizing strict and defeasible information in an inheritance system." CRL Memoranda MCCS-88-119, Computing Research Laboratory, New Mexico State University.

Bouma, G. (1990). "Defaults in unification grammar." In: *Proceedings, 28th Annual Meeting of the Association for Computational Linguistics*. 165–172.

Briscoe, T.; Copestake, A.; and de Paiva V., eds. (1991). *Proceedings of the* ACQUILEX *Workshop on Default Inheritance in the Lexicon*. Technical Report No. 238, Cambridge University Computer Laboratory.

Calder, J. (1989). "Paradigmatic morphology." In *Proceedings, Fourth Conference of the European Chapter of the Association for Computational Linguistics*. 58–65.

Carpenter, B. (1991). "Skeptical and credulous default unification with applications to templates and inheritance." In *Proceedings,* ACQUILEX *Workshop on Default Inheritance in the Lexicon*, edited by T. Briscoe, A. Copestake, and V. de Paiva. Technical Report No. 238, Cambridge University Computer Laboratory.

Daelemans, W. (1990). "Inheritance in object-oriented natural language processing." In *Proceedings of the Workshop on Inheritance and Natural Language Processing*, edited by W. Daelemans and G. Gazdar, 30–38. Tilburg University.

Daelemans, W., and Gazdar, G., eds. (1990). *Proceedings of the Workshop on Inheritance in Natural Language Processing*. ITK, Tilburg University.

De Smedt, K. (1984). "Using object-oriented knowledge-representation techniques in morphology and syntax programming." In *Proceedings, 3rd European Conference on Artificial Intelligence (ECAI84)*. 181–184.

De Smedt, K. and de Graaf, J. (1990). "Structured inheritance in frame-based representation of linguistic categories." In *Proceedings of the Workshop on Inheritance and Natural Language Processing*, edited by W. Daelemans and G. Gazdar, 39–47. Tilburg University.

Estival, D. (1990). *ELU User Manual*. Technical Report 1, ISSCO, Geneva.

Evans, R., and Gazdar, G., eds. (1990). *The DATR Papers: February 1990*. Cognitive Science Research Paper CSRP 139, School of Cognitive and Computing Sciences, University of Sussex, Falmer.

Flickinger, D. P. (1987). *Lexical rules in the hierarchical lexicon*. Doctoral dissertation, Stanford University, Palo Alto, CA.

Flickinger, D. P.; Pollard, C.; and Wasow, T. (1985). "Structure-sharing in lexical representation." In *Proceedings, 23rd Annual Meeting of the Association for Computational Linguistics*. 262–267.

Gazdar, G. (1990). "An introduction to

DATR." In *The DATR Papers: February 1990*, edited by R. Evans and G. Gazdar, 1–14, Cognitive Science Research Paper CSRP 139, University of Sussex.

Johnson, R., and Rosner, M. (1989). "A rich environment for experimentation with unification grammars." In *Proceedings, Fourth Conference of the European Chapter of the Association for Computational Linguistics*. 182–189.

Kaplan, R. M. (1987). "Three Seductions of Computational Psycholinguistics." In *Linguistic Theory and Computer Applications*, edited by P. Whitelock, M. M. Wood, H. L. Somers, R. Johnson, and P. Bennett, 149–188. Academic Press.

Karttunen, L. (1984). "Features and values." In *Proceedings, 10th International Conference on Computational Linguistics and the 22nd Annual Meeting of the Association for Computational Linguistics*. 28–33.

Karttunen, L. (1986). "D-PATR: A development environment for unification-based grammars." In *Proceedings, 11th International Conference on Computational Linguistics*. 74–80.

Kasper, R. T., and Rounds, W. C. (1986). "A logical semantics for feature structures." In *Proceedings, 24th Annual Meeting of the Association for Computational Linguistics*. 257–266.

Keene, S. (1989). *Object-Oriented Programming in Common Lisp*. Addison-Wesley.

Kilbury, J.; Naerger, P.; and Renz, I., (1991). "DATR as a lexical component for PATR." In *Proceedings, Fifth Conference of the European Chapter of the Association for Computational Linguistics*. 137–142.

Padgham, L. (1988). "A model and representation for type information and its use in reasoning with defaults." In *Proceedings, Seventh National Conference on Artificial Intelligence*. 409–414.

Rosenberg, S. (1983). "HPRL: A language for building expert systems." In *Proceedings, Eighth International Joint Conference on Artificial Intelligence*. 215–217.

Russell, G.; Carroll, J.; and Warwick, S. (1990). "Multiple default inheritance in a unification-based lexicon." In *Proceedings of the Workshop on Inheritance and Natural Language Processing*, edited by W. Daelemans and G. Gazdar. 93–102. Tilburg University.

Sandewall, E. (1986). "Nonmonotonic inference rules for multiple inheritance with exceptions." In *Proceedings, IEEE* 74. 1345–1353.

Schmolze, J. G., and Lipkis, T. A. (1983). "Classification in the KL-ONE knowledge representation system." In *Proceedings, Eighth International Joint Conference on Artificial Intelligence*. 330–332.

Selman, B., and Levesque, H. J. (1989). "The tractability of path-based inheritance." In *Proceedings, 11th International Joint Conference on Artificial Intelligence*. 1140–1145.

Shieber, S. M. (1986a). *An Introduction to Unification-Based Approaches to Grammar*. CSLI.

Shieber, S. M. (1986b). "A simple reconstruction of GPSG." In *Proceedings, 11th International Conference on Computational Linguistics*. 211–215.

Steele, G. L. (1990). *Common Lisp: The Language*, Second Edition. Digital Press.

Touretzky, D. S. (1986). *The Mathematics of Inheritance Systems*. Pitman Publishing.

Touretzky, D. S.; Horty, J. F.; and Thomason, R. H. (1987). "A clash of intuitions: The current state of nonmonotonic multiple inheritance systems." In *Proceedings, Tenth International Joint Conference on Artificial Intelligence*. 476–482.