Multiple Default Inheritance in a Unification-Based Lexicon

Graham Russell
John Carroll*
Susan Warwick-Armstrong
ISSCO, 54 route des Acacias,
1227 Geneva, Switzerland
elu@divsun.unige.ch

Abstract

A formalism is presented for lexical specification in unification-based grammars which exploits defeasible multiple inheritance to express regularity, subregularity, and exceptions in classifying the properties of words. Such systems are in the general case intractable; the present proposal represents an attempt to reduce complexity while retaining sufficient expressive power for the task at hand. Illustrative examples are given of morphological analyses from English and German.

1 Introduction

The primary task of a computational lexicon is to associate character strings representing word forms with information able to constrain the distribution of those word forms within a sentence.[1] The organization of a lexicon requires 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 individual words and subclasses of words. These considerations have provoked interest in applying to the lexicon AI knowledge representation techniques involving the notions of inheritance and default.[2] The system described here is part of the ELU[3] unification grammar development environment intended for research in machine translation, comprising parser, generator, transfer mechanism and lexical components. The user language resembles that of PATR-II (Shieber, 1986), 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 analyse 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.

2 Classes and Inheritance

An ELU lexicon consists of a number of 'classes', each of which is a structured collection of constraint equations and macro calls 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 information by specifying the 'intransitive' class as one of their superclasses – it then becomes unnecessary to specify the relevant properties individually for each such verb. 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, and which therefore inherit no information from elsewhere.

*current address: Cambridge University Computer Laboratory, New Museums Site, Pembroke Street, Cambridge CB2 3QG, UK.

[0]We are indebted to Afzal Ballim, Mark Johnson, and anonymous referees for valuable comments on this paper.

[1]In the general case, the relation between forms and information is many-to-many (rather than one-to-many as often assumed) and this observation has influenced the choice of facilities incorporated within the system. See 3.2 below for an example of how distinct forms share identical morphosyntactic specifications.

[2]See e.g. Daelemans and Gazdar eds. (1990), and the references in Gazdar (1990). The work of Hudson (1984) extends this general approach to sentence syntax.

[3]"Environnement Linguistique d'Unification" – see Estival (1990), and, for a description of the earlier UD system on which ELU is based, Johnson and Rosner (1989).

Lexical entries are themselves classes,[4] and any information they contain is standardly specific to an individual word; lexical and non-lexical classes differ in that analysis and generation take only the former as entry points to the lexicon.

Inheritance of a feature value from a superclass may be overridden by a conflicting value for that feature in a more specific class. This means, for example, that it is possible to place in the class which expresses 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.

A class definition consists of the compiler directive '#Class' (for a non-lexical class) or '#Word' (for a lexical class), followed by the name of that class, a possibly empty list of direct superclasses, a possible empty 'main' or default equation set, and zero or more 'variant' equation sets. The superclass declaration states from which classes the current class inherits; if more than one such superclass is specified, their order is significant, more specific classes appearing to the left of more general ones. If the current class is one of the most general in the lexicon, it inherits no information, and its superclass list is empty.

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 which will be clarified below.

Variant equation sets, loosely speaking, correspond to alternatives at the same conceptual level in the hierarchy, and in many cases reflect the traditional idea of 'paradigm'. 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 2.2). A feature structure is created for each variant set that successfully unifies with the single structure arising from the main set. Each variant set is preceded by the vertical bar '|'. The order of variant sets within a class is not significant, although, if a main set is employed, it must precede any variant sets.

The following simplified example illustrates the form and interaction of class definitions. In equations, unification variables have initial capitals, and negation of constants is indicated by '~'. '&&' is the string concatenation operator – an equation of the form X = Y && Z unifies X nondeterministically with the result of concatenating Y and Z.

```
#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> = ~sg3
   <tense> = present
   <form> = <stem>
```

The lexical class walk is declared as having two direct superclasses, Intransitive and Verb; 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. The latter assign, by default, the values of 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, 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.[5]

[4] Thus no distinction is made between classes and 'instances', as in e.g. KL-ONE (Schmolze and Lipkis, 1983)

[5] We ignore for the moment the question of morphographemic effects in suffixation – see section 3.3 below.

2.1 Class Precedence

In an ELU lexicon, a class may inherit directly from more than one superclass, thus permitting 'multiple inheritance' (Touretzky, 1986:7ff.), in contrast to 'simple inheritance' in which direct inheritance is allowed from only one superclass at a time. The main advantage that multiple inheritance offers over simple inheritance is the ability to inherit several (orthogonal or complementary) sets of properties from classes in more than one path through the hierarchy. In the lexical context, it has often been observed that morphological and syntactic properties are essentially disjoint; the subcategorisation class of a verb is not predictable from its conjugation class, and vice versa, for example. Multiple inheritance permits the two types of information to be separated by isolating them in distinct subhierarchies.

The question to be resolved in systems employing multiple inheritance is that of precedence: which of several superclasses with conflicting properties is to be inherited from? ELU employs the class precedence algorithm of the Common Lisp Object System (CLOS) to compute a total ordering on the superclasses of a lexical class.[6] The resulting 'class precedence list' (CPL) contains the class itself and all of its superclasses, from most specific to most general, and forms the basis for the defaulting behaviour of the lexicon. As an example, consider the following lexicon:

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

Here, the superclass declarations embody the ordering constraints $A<B, A<D, B<D, B<C, C<F, D<E$, and $E<F$; from these are derived a total order assigning to the lexical class A the CPL $\langle A, B, C, D, E, F \rangle$.

2.2 Inheritance of Properties

A lexical class such as `walk` in the example above corresponds to a family of feature structures. Here, as in most analyses, members of this family represent morphosyntactically distinct realizations of a single basic lexeme. Consulting the lexicon involves determining membership of the set of feature structures associated with a given lexical class; the precedence relation encoded in the CPL controls the order in which defeasible information is considered, each class in the CPL adding first default and then non-default information to each FS produced by the previous class.

More formally, we define *default extension*, *superclass extension*, and *global extension* as follows:[7]

(1) The *default extension* of a FS ϕ with respect to a set of FSs Ψ is

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

if $\bigsqcup(\{\phi\} \cup \Psi) \neq \bot$, and $\bot$ otherwise.

(2) 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 the set

$$\{\psi \mid 1 \leq i \leq n \wedge v_i' \sqcup \phi' = \psi \wedge \psi \neq \bot\},$$

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. We refer to this set as $\Sigma(\phi, c)$.

(3) The *global extension* of a lexical class having the CPL $(c_1, \ldots c_n)$ is Γ_n, where $\Gamma_0 = \{\top\}$, and

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

With regard to (1), 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. The condition requiring ϕ to be unifiable with the result of unifying the elements of Ψ takes account of the potential order-sensitivity of the defaulting operation – only those main sets having this property can be applied without regard to order. If this condition is met then the application of defaults always succeeds, producing a feature structure which, if no member of the default set is applicable, is identical to ϕ. This interpretation of default unification is essentially that of Bouma (1990).

The superclass extension $\Sigma(\phi, c)$ 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

[6] See Steele (1990: 782ff.) for details of the algorithm, and Keene (1989:118ff.) for discussion. In circumstances where no such total ordering is possible, the system reports an error.

[7] '$A \sqcup B$' 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.

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.

To speak in procedural terms, the global extension of a lexical class L with the CPL C is computed as follows: $\top$ 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.

It is possible to exert quite fine control over inheritance; one property may override another when assigned in a main equation set, but cause failure when assigned in a variant set. Normally, variant sets are defined so as to be mutually exclusive; a FS that unifies with more than one of the variant sets is in effect multiplied.[8] The inheritance systems of Calder (1989) and Flickinger (1987) make use of lexical rules – the ELU lexicon does not provide such devices, although some of their functionality may be reproduced by the variant set mechanism.

The approach described here differs from some previous proposals for default inheritance in unification-based lexicons in that the process of building FSs is monotonic – classes may add information to a FS, but are unable to remove or alter it. Thus, 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 $\langle c_{i+1}, \ldots c_m \rangle, m \leq n$. Karttunen (1986) and Shieber (1986) describe systems in which FSs may be modified by default statements in such a way that this property does not automatically hold. These schemes permit default statements to override the effect of earlier statements, whereas default information in the ELU lexicon may itself be overridden.

We now turn to some examples illustrating the rôle of defeasible inheritance in the lexicon.

3 Example Analyses

3.1 German Separable Verbs

Two large classes of German verbs are the separable and inseparable prefixed compound verbs. The former 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.[9] Members of both classes share morphological, but not necessarily syntactic, properties of the verb which corresponds in form to their stem. The separable-prefix verb *weglaufen* ('run away') and inseparable *verlaufen* ('elapse') are two such verbs, which the lexicon should be able to relate to their apparent stem *laufen* ('run').

Since word definitions are classes, they can be inherited from like any non-lexical class. Thus the lexical classes `verlaufen` and `weglaufen` may inherit from `laufen`, itself a lexical class:[10]

```
# Word weglaufen (weg laufen)
        <sem> = weglaufen

# Word verlaufen (ver laufen)
        <sem> = verlaufen

# Class weg (separable)
        <morph prefix> = weg

# Class ver (non_separable)
        <morph prefix> = ver

# Word laufen (verb)
        Base_stem = lauf
        <sem> = laufen

# Class non_separable ()
        Prefix = <morph prefix>

# Class separable ()
        |
        Prefix = <morph prefix>
        <syn inv> = no
        <syn infl> = ~inf
        |
        Prefix = ''
        <syn inv> = yes
        <syn infl> = ~inf
        |
        Prefix = <morph prefix>
        <syn infl> = inf

# Class verb ()
        <cat> = v
        Prefix = ''
        <morph prefix> = none
        P_bs = Prefix && Base_stem
        |
        <syn infl> = inf
        <form> = P_bs && en
        |
        <form> = P_bs && e
        <syn infl> = pres_indic_sg_1
```

[8] See 3.2 below for a case where such multiple matches are desirable.

[9] Within the syntactic analysis assumed here, the distribution of verbs is controlled by a binary feature inv, whose value in these contexts is no.

[10] A number of simplifications have been made here; *laufen* is in reality a member of a subclass of the strong verbs, and the verb class itself has been truncated, so that it accounts for only bare infinitive and first person singular present tense indicative forms. Past participle formation also interacts with the presence of separable and inseparable prefixes.

The lexical classes **weglaufen** and **verlaufen** each have two immediate superclasses, containing information connected with the prefix and stem. The classes **weg** and **ver** set the value of the morph : **prefix** path of the verb (overriding the value given in the main set of **verb**), and specify inheritance from the **separable** and **non_separable** classes respectively. The former of these unifies the variable **Prefix** with either the empty string (in the case of tensed 'inverted' verbs) or the value of morph : **prefix** (for other variants), while the latter sets the value uniquely for all forms of the verb in question. As the value of **sem** is fixed in the main equation set of **weglaufen** and **verlaufen**, the corresponding equation in **laufen** is overridden, but **Base_stem** unifies with **lauf**. Finally, in **verb**, the main set supplies default values for **Prefix** and morph : **prefix** (which in the cases under consideration will not be applicable), unifies **P_bs** with the result of concatenating the strings **Prefix** and **Base_stem**, and for each value of **syn infl** assigns to **form** the concatenation of **P_bs** with the appropriate suffix string.

Values for **sem** (antics) are provided in main set equations; those in **weglaufen** and **verlaufen** are thus correctly able to override that in **laufen**.

3.2 English Irregular Verbs

In most cases, lexical items that realise certain morphosyntactic properties in irregular forms do not also have regular realizations of those properties; thus **sinked* is not a well-formed alternative to *sank* or *sunk*, on the analogy of e.g. *walked*. 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.[11] There are exceptions to this general rule, however, and inheritance mechanisms must be sufficiently flexible to permit deviation from the strict behaviour illustrated above.

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 pairings of strings and feature structures, the written form of the word being encoded as the value of the **form** feature:[12]

```
#Word walk (verb)
      <base> = walk

#Word sink (verb)
      <base> = sink
      P_Fin_Form = sank
      PSP_Form = sunk

#Word dream (dual-past verb)
      <base> = dream

#Class dual-past ()
      |
      PSP_Form = <base> && t
      P_Fin_Form = <base> && t
      <morph> = pastfinite/pastnonfinite
      |

#Class verb ()
      <cat> = v
      PSP_Form = <base> && ed
      P_Fin_Form = <base> && ed
      |
      <morph> = present_nonsg3
      <form> = <base>
      |
      <morph> = present_sg3
      <form> = <base> && s
      |
      <morph> = pastnonfinite
      <form> = PSP_Form
      |
      <morph> = pastfinite
      <form> = P_Fin_Form
```

The main set equations in **sink** override those in its superclass **verb**, 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 class **walk**, on the other hand, contains nothing to pre-empt the equations in **verb**, and so its past forms are constructed from its value for **base** and the suffix string **ed**.

The lexical class **dream** differs from these in having as one of its direct superclasses **dual-past**, 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 its equations assigning to **PSP_Form** and **P_Fin_Form** the string formed by concatenating the value of **base** and *t* have precedence over the contradictory statements in the main set of **verb**. Note that this set also includes a disjunctive constraint to the effect that the value of **morph** in this FS must be either **pastfinite** or **pastnonfinite**. The **dual_past** class thus describes two feature

[11] See e.g. Calder (1989).

[12] Again, the analysis sketched here is simplified; several variants within the **verb** class have been omitted, and all inflectional information is embodied as the value of the single feature **morph**.

structures, but adds no information to the second. The absence of contradictory specifications permits the equations in the main set of verb to apply, in addition to those in the first variant set of dual-past. The second, empty, variant set in dual-past permits this class also to inherit all the properties of its superclass, 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* past forms.

3.3 Word-Form Manipulation

The string concatenation operator '&&' allows the lexicon writer to manipulate word forms with ELU equations and macros. In particular, && can be used to add or remove prefixes and suffixes, and also to effect internal modifications, such as German Umlaut, by removing a string of characters from one end, changing the remainder, and then replacing the first string. In this section we show briefly how unification, string concatenation, and defeasible inheritance combine to permit the analysis of some of the numerous orthographic changes that accompany English inflectional suffixation.

The inflectional paradigms of English nouns, verbs, and adjectives are complicated by a number of orthographic effects; *big*, *hop*, etc. undergo a doubling of the final stem character in e.g. *bigger*, *hopped*, stems such as *fox*, *bush*, and *arch* take an epenthetic *e* before the plural or third singular present suffix *s*, stem-final *ie* becomes *y* before the present participle suffix *ing*, and so on. Peripheral alternations of this kind are accomplished quite straightforwardly by macros like those in the following lexicon fragment (in which invocations of user-defined macros are introduced by '!'):[13]

```
Final_Sibilant(String)
    String = _ && sh/ch/s/x/z

Final_Y(String,Prefix)
    String = Prefix && y
    Prefix = _ &&
     b/c/d/f/g/h/j/k/l/m/n/p/r/s/t/v/w/x/z

# Word try (verb_spelling)
    <base> = try

# Word watch (verb_spelling)
    <base> = watch
```

[13] As before, this is a somewhat abbreviated version of a full description; the verb and verb_spelling classes require additional variant sets to account for other morphosyntactic properties. Other string-predicate macros, in particular OK, must be defined in order to cater for the full range of spelling changes observed in verbal inflection.

```
# Class verb_spelling (verb)
    |
    !Final_Y(<base>,P)
    Base_P_PSP = P && i
    Base_3SG = P && ie
    |
    !Final_Sibilant(<base>)
    Base_3SG = <base> && e
    |
    !OK(<base>)

#Class verb ()
    <cat> = v
    Base_3SG = <base>
    Base_P_PSP = <base>
    PSP_Form = Base_P_PSP && ed
    SG3_Form = Base_3SG && s
    |
    !Sing3
    <form> = SG3_Form
    |
    !PastNonFin
    <form> = PSP_Form
```

Two macros definitions are shown here; Final_Y is true of a pair of strings String and Prefix iff String consists of Prefix followed by *y* and the final character of Prefix is one of the set denoted by the disjunction b/c...z, while Final_Sibilant is true of a given string iff that string terminates in *sh*, *ch*, *s*, *x*, or *z*. OK is a macro which is true of only those strings to which neither Final_Sibilant nor Final_Y apply.

The class verb_spelling contains three variant equation sets, the first two of which assign values to variables according to the form of the string which is the value of the base feature. If Final_Y is applicable, Base_P_PSP is unified with the concatenation of the second argument to the macro (e.g. *tr*) and *ie*, while Base_3SG is unified with e.g. *tr* and *i*. If Final_Sibilant is applicable, then Base_3SG is unified with the concatenation of the value of base (e.g. *watch*) and *e*. If neither of these is applicable (because the base string does not match the patterns in the macro definitions), the variables are bound not within this class, but in the main equation set of its superclass verb. Here, their values are unified directly with that of base, and the eventual values of the form feature result from concatenation of the appropriate suffix strings, giving values of *watched*, *watches*, *tried*, and *tries*.

4 Summary

The lexicon system presented above is fully integrated into the ELU environment; in particular, the result of analysis and the starting point for generation is the same type of feature structure as that produced by ELU grammars, and the equa-

tions within classes are of the same sort as those used elsewhere in a linguistic description, being able to exploit re-entrancy, negation, disjunction, direct manipulation of lists, etc.

For the purpose of experimenting with the structure of the class hierarchy and the distribution of information within individual classes, the lexicon is held in memory, and is accessed directly by means of an exhaustive search. Once a stable description is achieved, and the coverage of the lexicon increases, a more efficient access mechanism exists, in which possible word-forms are pre-computed, and used to index into a disk file of class definitions.

We have presented an implemented treatment of a framework for lexical description which is both practical from the perspective of efficiency and attractive in its reflection of the natural organization of a lexicon into nested and intersecting generalizations and exceptions. The system extends traditional unification with a multiple default inheritance mechanism, for which a declarative semantics is provided.

References

Bouma, G. (1990) "Defaults in Unification Grammar," *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, Pittsburgh, June 6th–9th. 165–172.

Calder, J. (1989) "Paradigmatic Morphology," *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, Manchester, April 10th–12th. 58–65.

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

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

Flickinger, D. P. (1987) "Lexical Rules in the Hierarchical Lexicon," PhD Thesis, Stanford University.

Gazdar, G. (1990) "An Introduction to DATR," in R. Evans and G. Gazdar (eds.) *The DATR Papers: February 1990*. Cognitive Science Research Paper CSRP 139, School of Cognitive and Computing Sciences, University of Sussex. 1–14.

Hudson, R. A. (1984) *Word Grammar*. Oxford: Blackwell.

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. 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. 74–80.

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

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. 330–332.

Shieber, S. M. (1986) *An Introduction to Unification-Based Approaches to Grammar*. CSLI Lecture Notes no. 4, Stanford University.

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

Touretsky, D. S. (1986) *The Mathematics of Inheritance Systems*. London: Pitman Publishing.