'Lexical Rules' are just lexical rules

Roger Evans*
University of Brighton

Gerald Gazdar, David Weir†
University of Sussex

Abstract

The question we address in this paper is whether 'Lexical Rules' deserve their grand status, a status that is often conveyed by a special purpose formalism and/or a separate component, one that may even be external to the lexicon proper. We will argue that they do not and that a lexical knowledge representation language that is as expressive as it needs to be for other lexical purposes will, ipso facto, be expressive enough to encode 'Lexical Rules' internally as lexical rules. Such internal encoding is not only possible but also desirable since 'Lexical Rules' will then automatically acquire other characteristics which are now standardly associated with common or garden lexical rules, including inheritance, generalization by default, and the ability to relate lexical information from different levels of linguistic description. We give examples of what we take to be instances of common or garden lexical rules and then show how the same formal machinery provides for the statement of a PATR-like version of passive and the description of unbounded dependencies, inter alia, in Lexicalized Tree Adjoining Grammar (LTAG). We show how to define an LTAG lexicon as an inheritance hierarchy with internal lexical rules. A bottom-up featural encoding is used for LTAG trees and this allows lexical rules to be implemented as covariation constraints within feature structures. Such an approach eliminates the considerable redundancy otherwise associated with an LTAG lexicon.

1 Introduction

If you build a lexicon for English that includes an entry for the lexeme **alumnus** then some component of the lexicon is going to have to state that the plural form of the lexeme is constructed by suffixation of *i* to the root (hence *alumni*). This statement is, uncontroversially, a lexical rule[1]. Your description of English may also contain some much grander component, one, say, that maps subcategorization frames for transitive verbs into those for passive verbs. This much grander component is, by common consent, a 'Lexical Rule'. Carpenter (1991; 1992) and Ritchie et al. (1992, 93-111 & 193-196) provide discussion and exemplification of 'Lexical Rules' in several different grammatical frameworks. Canonically, such rules deal with the phenomena that used to be described by the 'cyclic rules' of late 1960s transformational grammar (i.e., passive, dative[2], subject-auxiliary inversion, etc.). Characteristically, they pertain to rather specific classes of lexical items (e.g., transitive verbs, or tensed auxiliary verbs) and they are subject to exceptions of various kinds. It is these characteristics that have led many linguists to consign them to the lexicon. They usually involve a difference in argument structure and this is sometimes accompanied by a morphological difference. Radically lexicalist frameworks, such as LTAG, which lack any construction-specific grammatical rules outside the lexicon, do not restrict the use of lexical rules to 'cyclic' phenomena.

We shall argue that 'Lexical Rules' do not deserve their grand status and that they do not require special purpose formalism and/or a separate component, one that may even be **external** to the lexicon proper. A lexical knowledge representation language that is as expressive as it needs to be for other lexical purposes

*Information Technology Research Institute, University of Brighton, Brighton BN2 4AT, U.K.; Roger.Evans@itri.bton.ac.uk

†Cognitive & Computing Sciences, University of Sussex, Brighton BN1 9QH, U.K.; Gerald.Gazdar@cogs.sussex.ac.uk, David.Weir@cogs.sussex.ac.uk

[1] The statement has to be a rule because it applies to more than one lexeme and it has to be lexical because its applicability is peculiar to a small lexically specified set.

[2] See Lascarides et al. (1996, 74-79,84) for a detailed formal discussion of the semantic characteristics of dative constructions in the context of HPSG augmented with a default inheritance mechanism.

will, ipso facto, be expressive enough to encode 'Lexical Rules' **internally** as lexical rules. Such encoding is not only possible but also desirable since 'Lexical Rules' then automatically acquire other characteristics which are standardly associated with common or garden lexical rules, including inheritance, generalization by default, and the ability to relate lexical information from different levels of linguistic description.[3]

2 Some lexical rules

We will consider some routine matters of lexical description in the area of morphology. The following partial analysis of English verbs exemplifies a typical approach to the representation of English inflectional morphology using the DATR lexical knowledge representation language (Evans & Gazdar 1996; Keller 1995, 1996) and the SAMPA alphabet for the phonology (Wells 1987).

```
Word:
    <> == Null
    <word> == "<root>" "<suffix>"
    <form> == "<word "<syn atts>">"
Verb:
    <> == Word
    <cat> == verb
    <suffix past> == Suffix_ED
    <suffix pres participle> == Suffix_ING
    <suffix pres tense sing three> == Suffix_S.
Walk:
    <> == Verb
    <root> == w O: k.
```

This fragment employs several devices for capturing relationships within and across lexical entries. The second equation of `Word` provides a (default) morphotactics for English: a word consists of a root and a suffix. The third equation makes the <`form`> of a word depend on its syntactic attributes (<`syn atts`>, which takes values such as `past participle`, `past tense plur one`, `present participle`, `present tense sing three`), thus parameterizing the inheritance path that determines its value. The first `Verb` equation allows all the properties of `Word` to be inherited, whilst the final three equations provide (exhaustively) for regular verbal inflection in English verbs. The inheritance mechanism casts all the `Verb` equations as generalizations across verbs, even that for <`cat`> which at first sight appears to be a simple statement with no rule-like qualities at all.

Turning to the phonological form of the suffix morphemes, we find that one of them can be defined trivially:

```
Suffix_ING:
    <suffix> == I N g.
```

But the other two have a phonological form that depends on the phonology of the root to which they are attached. In the case of the *-ed* suffix, it appears as `/{d/` if the final segment of the root is an alveolar consonant, otherwise it appears as `/d/` if the final segment is voiced and a `/t/` if it isn't.

```
Suffix_ED:
    <suffix> == IF:<ALVEOLAR:<FINAL_SEG:<"<root>">>
                THEN I d
                ELSE IF:<VOICED:<FINAL_SEG:<"<root>">>
                     THEN d
                     ELSE t >>.
```

[3] At least one of us has actually held this view implicitly since 1989 (Gazdar 1989) but not spelled it out in prose before. The view is, of course, not unique to us (cf. Krieger 1994, Lascarides et al. 1996).

Likewise, the *-s* suffix appears as /Iz/ if the final segment of the root is a sibilant, otherwise it appears as /z/ if the final segment is voiced and a /s/ if it isn't.

```
Suffix_S:
    <suffix> == IF:<SIBILANT:<FINAL_SEG:<"<root>">>
                THEN I z
                ELSE IF:<VOICED:<FINAL_SEG:<"<root>">>
                     THEN z
                     ELSE s >>.
```

These are very familiar facts about English inflection and there is nothing surprising or unconventional in our account of them.

Clearly, in the examples we have been considering, rules (rather than isolated statements about individual entries) are being stated. Equally clearly, these rules are lexical. But should they be thought of as 'Lexical Rules'? We do not think they should. They have no special status, either formally or analytically. They cannot easily or sensibly be detached from the lexical description in which they are embedded, and they are not different in kind from the rest of that description. Thus, for example, the equations above that provide for the realization of past forms of verbs are default statements, just like the equations that provide for third person singular present tense forms[4] Any of them can be overridden elsewhere in the description, as in the following lexeme specifications:

```
Go:
    <> == Verb
    <root> == g @ U
    <word past tense> == w E n t
    <word past participle> == g Q n.
Have:
    <> == Verb
    <root> == h { v
    <root past> == h { d
    <root pres tense sing three> == h { z.
Get:
    <> == Verb
    <root> == g E t
    <root past> == g Q t
    <suffix past> == Null.
```

In fact, the use of inheritance means there is no sensible dividing line at all between rules and isolated statements. The only plausible candidates for (pure) isolated statements would be statements made at nodes corresponding to surface word forms which do not override default statements[5]. In particular **all** the above definitions are rules, since none of the nodes introduced denote surface word forms.

3 A 'Lexical Rule'

If one puts aside matters of notation and content, most of what is going on semantically in these fragments of lexical morphology can be reduced to functions which take arguments and function definitions that involve conditionals. Once one has signed up to an active view of lexical knowledge representation (as

[4] Our treatment of third person singular inflection is quite different from that outlined in Lascarides et al. (1996, 81-83) but the differences mostly reflect the fact that they presuppose a (typed) word-based lexicon whilst we are presupposing an (untyped) lexeme-based lexicon. However, their resultant sign is of type **third-person-sing**, which specifies agreement and suffixation, and is indefeasible. It is not clear whether this means that the account is inconsistent with such irregular forms as *is, has, does,* etc.

[5] If they did override a default, one could argue they were simply part of the nonmonotonic definition for the rule partially implemented by the default.

opposed to thinking of a lexicon as a relational database, say), then it is hard to imagine settling for less. But that amount of machinery is quite sufficient to encode the subcategorization-related phenomena that conventionally fall under the rubric of 'Lexical Rules'. Nothing extra is required.

By way of illustration, consider the following syntactic extension of the English verb fragment that implements a lexical rule for the (agentless) passive construction:

```
VERB:
    <syn subcat> == "<syn args>"
    <syn args> == NP_ARG:<>
    <syn args first syn case> == nominative.
TR_VERB:
    <> == VERB
    <mor passive> == "<mor past>"
    <syn args rest> == NP_ARG:<>
    <syn subcat rest> ==
        "<syn args rest "<mood-adjust "<syn form>">">"
    <mood-adjust> ==
        <mood-adjust passive> == rest.
Love:
    <> == TR_VERB
    <mor root> == love.
```

We have added some basic syntactic information, in the form of the features <syn args>, a first/rest list of formal syntactic argument specifications, and <syn subcat>, a similar list for the actual subcategorization frame. The defaults at VERB introduce a nominative subject NP as first formal argument[6], and set the subcategorization list to be the same as the formal argument list.

We have introduced an explicit node, TR_VERB, for transitive verbs, which adds passive morphology and an additional NP as second formal argument. It also extends the definition for the subcategorization list: the inheritance specification for <syn subcat rest> depends on an adjustment feature that takes account of the surface syntactic form. If <syn form> is passive, <mood-adjust> takes the value rest, so <syn subcat rest> inherits from <syn args rest rest>, effectively removing the formal object NP from the subcategorization frame. If <syn form> is not passive, <mood-adjust> takes the empty value, and <syn subcat rest> inherits from <syn args rest> just as it would have by default. The subject argument, <syn subcat first>, is unaffected by the definition at TR_VERB and so always inherits from <syn args first>.

This approach captures the passive alternation using the same descriptive techniques used earlier to describe routine matters of inflectional morphology. The only perceptible increase in complexity is the use of a doubly embedded path specification. Just as before, we can identify no distinction sufficient to warrant the label of 'Lexical Rule'. Passivization here is simply an interdependence among feature specifications much like any other.

Other benefits also accrue from this style of encoding alternations. The scope of the alternation is defined directly by its location within the inheritance hierarchy, rather than by a separate stipulation of constraints on the type or form of some 'input' specification. Thus the restriction of passivization to transitive verbs follows automatically from the location of the relevant definitions at TR_VERB. On the other hand, nonmonotonicity allows subclasses or instances to override or augment the default definitions as appropriate (hence lexical exceptions to passive). A further advantage is that these relationships are not restricted to operating on 'inputs' which are themselves free-standing lexical entries, so 'alternations' may exist for which there is no (visible) base form.

The techniques used in this fairly simple treatment of passive can be adapted for use in encoding other lexical rules and for grammatical frameworks other than that implicit in the HPSGish syntax we have adopted in our example. They can also be readily adapted for use in the semantic domain[7] and used,

[6] We assume the node NP_ARG is defined elsewhere as a prototypical NP complement specification.

[7] This point seems to have been missed by Candito (1996, 195) in her critical remarks on our 1995 paper. Formal manipula-

for example, to implement the distinction between **fixed** and **projective** inheritance of lexical semantic information proposed by Pustejovsky (1991,433-437).

The account of passive presented above lends support to our position, but is scarcely a convincing account for lexical rules in general. Its operation depends on the use of <syn form> as a 'trigger' feature: if its value is passive, the alternation takes effect. But, for other alternations (dative and topicalisation for example), no such feature is obviously available. It is not very modular: interactions **between** rules need to be carefully organised to achieve the correct results. And taken together, these two problems also mean that the control structure for 'applying' rules is very rigid, effectively hard-wired into the inheritance structure.

But these problems reflect the simplicity of the formulation, rather than the inherent limitations of the approach. In the remaining sections of this paper, we will present an encoding of a fragment of LTAG in DATR. The analysis that underlies this work includes an encoding of lexical alternations which addresses the failings of the simpler approach. In addition, LTAG's extended domain of locality allows us to consider a wider range of lexical alternations than is possible in approaches based on local trees alone. For our purposes, the main difference between an LTAG entry and a typical HPSG-style lexical entry is that in LTAG an entry 'subcategorizes' for an arbitrarily large tree structure.

4 Efficient representation of LTAG lexicons

As with all fully lexicalized grammar formalisms, there is really no conceptual distinction to be drawn in LTAG between the lexicon and the grammar: the grammatical rules are just lexical properties. The issue of efficient representation for LTAG is discussed by Vijay-Shanker & Schabes (1992), who draw attention to the considerable redundancy inherent in LTAG lexicons that are expressed in a flat manner with no sharing of structure or properties across the elementary trees. For example, the XTAG grammar (Abeille et al., 1990; Doran et al., 1994b; Doran et al., 1994a; XTAG Research Group, 1995) currently includes over 100,000 lexemes, each of which is associated with a family of trees (typically around 20) drawn from a set of over 500 elementary trees. Many of these trees have structure in common, many of the lexemes have the same tree families, and many of the trees within families are systematically related in ways which other formalisms capture using transformations or metarules. However, the LTAG formalism itself does not provide any direct support for capturing such regularities.

Vijay-Shanker & Schabes address this problem by introducing a hierarchical lexicon structure with monotonic inheritance and lexical rules, using an approach loosely based on that of Flickinger (1987) but tailored for LTAG trees rather than HPSG subcategorization lists. Becker (1993; 1994) proposes a slightly different solution, combining an inheritance component and a set of metarules[8].

There are various ways that an inheritance hierarchy can be used to reduce the number of statements needed to encode the trees in an LTAG. For example, common structure in trees related by lexical rules can be shared; tree structure common to intransitive, transitive and ditransitive verbs can be shared; and tree structure for various forms of complements (which in the case of PP complements can involve several nodes) can be shared across all places where these complements appear.

The principal unit of (syntactic) information associated with an LTAG entry is a tree structure in which the tree nodes are labeled with syntactic categories and feature information and there is at least one leaf node labeled with a **lexical** category (such lexical leaf nodes are known as **anchors**). For example, the canonical tree for a ditransitive verb such as *give* is shown in figure 1. Following LTAG conventions (for the time being), the node labels here are gross syntactic category specifications to which additional featural information may be added[9], and are annotated to indicate node **type**: $\diamond$ indicates an anchor node, and $\downarrow$

tion of (semantic) argument structures is not interestingly different from formal manipulation of (syntactic) subcategorization frames. If you can do one, you can do the other. And if you do one, you can do the other at the same time, hitching it to the same wagon. For this reason we shall have virtually nothing to say about semantics in this paper, and although the minimalism of our purely syntactic account of dative (for example) may be surprising, our examples do illustrate sufficient machinery to fully support a proper semantic account if desired.

[8] See Section 8 for further discussion of these approaches.

[9] In fact, LTAG commonly distinguishes two sets of features at each node (top and bottom), but for simplicity we shall assume just one set in this paper.

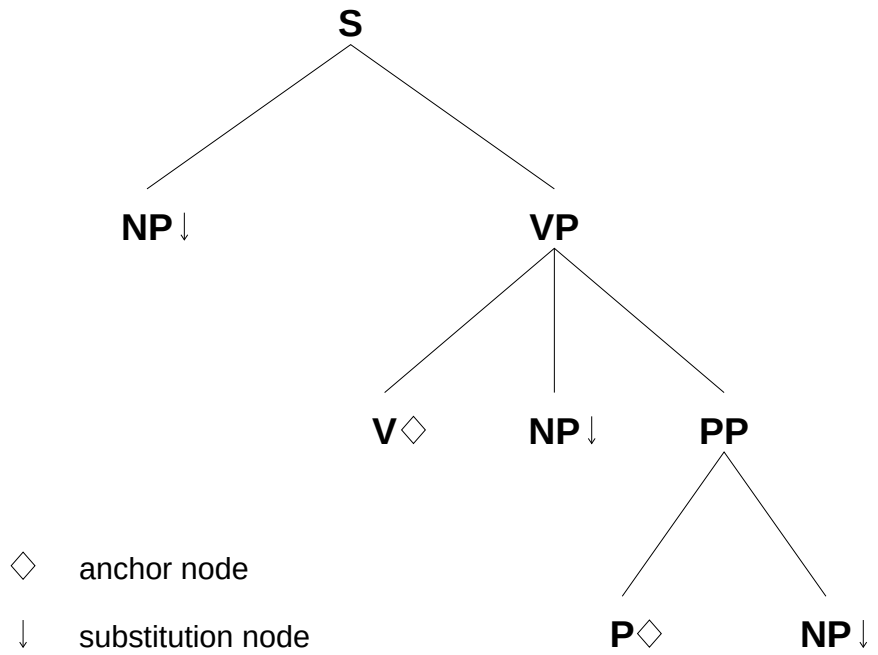

Figure 1: An example LTAG tree for *give*

indicates a substitution node (where a fully specified tree with a compatible root label may be attached)[10].

In representing such a tree, we do two things. First, in keeping with the radically lexicalist character of LTAG, we describe the tree structure from its (lexical) anchor upwards[11], using a variant of Kilbury's (1990) bottom-up encoding of trees. In this encoding, a tree is described relative to a particular distinguished leaf node (here the anchor node), using binary relations `parent`, `left` and `right`, relating the node to the subtrees associated with its parent, and immediate-left and -right sisters, encoded in the same way. Second, we embed the resulting tree structure (i.e., the node relations and type information) in the feature structure, so that the tree relations (`left`, `right` and `parent`) become features. The obvious analogy here is the use of `first/rest` features to encode subcategorisation lists in frameworks like HPSG.

Thus the syntactic feature information directly associated with the entry for *give* relates to the label for the V node (for example, the value of its `cat` feature is `v`, the value of `type` is `anchor`), while specifications of subfeatures of `parent` relate to the label of the VP node. A simple bottom-up representation for the whole tree (apart from the node type information) follows:

```
Give:
    <cat> = v
    <parent cat> = vp
    <parent left cat> = np
    <parent parent cat> = s
    <right cat> = np
    <right right cat> = p
    <right right parent cat> = pp
    <right right right cat> = np.
```

This says that `Give` is a verb, with VP as its parent, an S as its grandparent and an NP to the left of its parent. It also has an NP to its right, and a P to the right of that, which has a PP parent and a

[10] LTAG's other tree-building operation is **adjunction**, which allows a tree-fragment to be spliced into the body of a tree. However, we only need to concern ourselves here with the **representation** of the trees involved, not with the substitution/adjunction distinction.

[11] The tree in figure 1 has more than one anchor – in such cases it is generally easy to decide which anchor is the most appropriate root for the tree (here, the verb anchor).

NP to its right. The implied bottom-up tree structure is shown graphically in figure 2. Here the nodes are laid out just as in figure 1, but related via `parent`, `left` and `right` links, rather than the more usual (implicitly ordered) daughter links. Notice in particular that the `right` link from the object noun-phrase node points to the *preposition* node, not its phrasal parent – this whole subtree is itself encoded bottom-up. Nevertheless, the full tree structure is completely and accurately represented by this encoding. Note that `right` and `left` do not encode sisterhood directly. In this example, the fact that the node labelled PP is a right sister of the V can be derived from the fact that the PP node has no `parent` and is an ancestor of the node (labelled P) which is to the `right` of the V node.

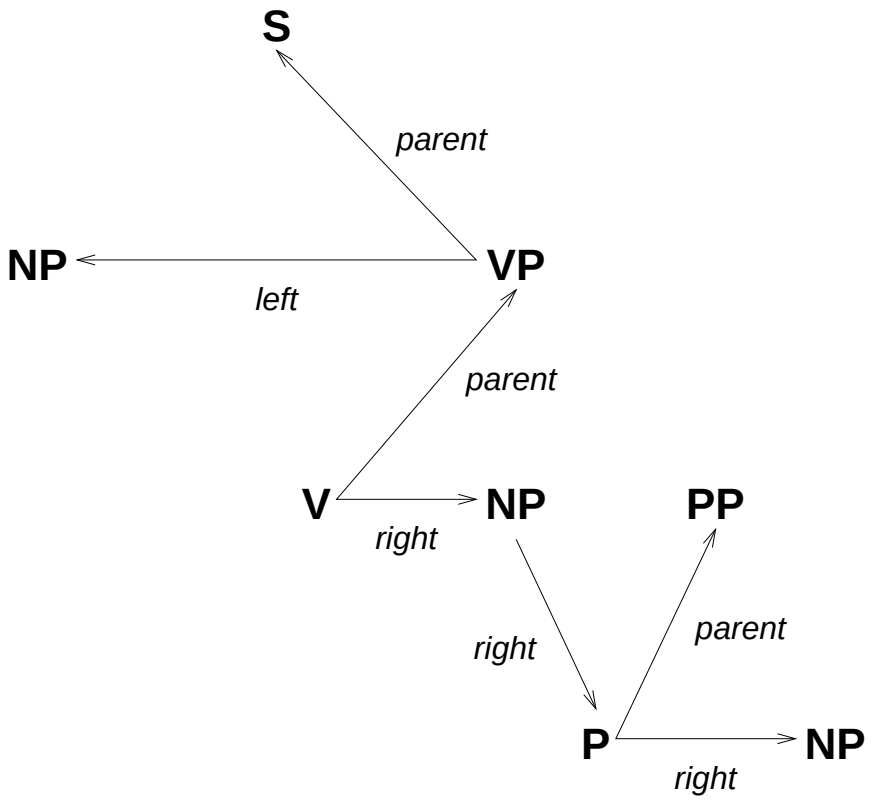

Figure 2: Bottom-up encoding for `Give`

Once we adopt this representational strategy, writing an LTAG lexicon becomes similar to writing any other type of lexicalist grammar's lexicon in an inheritance-based LKRL. In HPSG, for example, the subcategorisation frames are coded as lists of categories, whilst in LTAG they are coded as trees. But, in both cases, the problem is one of concisely describing feature structures associated with lexical entries and relationships between lexical entries. The same kinds of generalization arise and the same techniques are applicable. Of course, the presence of complete trees and the fully lexicalized approach provide scope for capturing generalizations lexically that are not available to approaches that only identify parent and sibling nodes, say, in the lexical entries.

5 Encoding lexical entries

Following conventional models of lexicon organisation, we would expect `Give` to have a minimal syntactic specification itself, since syntactically it is a completely regular ditransitive verb. In fact **none** of the information introduced so far is specific to `Give`. So rather than providing a completely explicit definition for `Give`, as we did above, a more plausible account uses an inheritance hierarchy defining abstract intransitive, transitive and ditransitive verbs to support `Give` (among others), as shown in figure 3.

This kind of organisation is applicable to a wide range of possible lexical formalisms, but in the LTAG

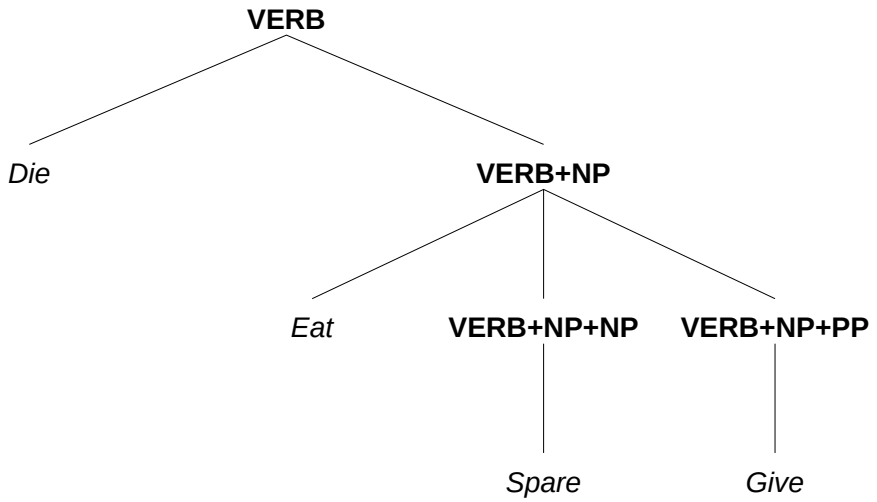

Figure 3: The principal lexical hierarchy

case, figure 3 hides an additional complication: each node in this hierarchy has associated with it a complete (syntactic) tree structure, and the definitions of those structures may also share information via inheritance. To avoid confusion, we shall look at these two layers in turn. Figure 4 shows the tree structures associated with the principal nodes above `Give`. The bold lines represent information defined at the current node, the light lines represent information inherited from the parent node.

This basic organisational structure can be expressed as the following fragment[12]:

```
VERB:
    <> == TREENODE
    <cat> == v
    <type> == anchor
    <parent> == VPTREE:<>.

VERB+NP:
    <> == VERB
    <right> == NPCOMP:<>.

VERB+NP+PP:
    <> == VERB+NP
    <right right> == PTREE:<>
    <right right root> == to.

VERB+NP+NP:
    <> == VERB+NP
    <right right> == NPCOMP:<>.

Die:
    <> == VERB
    <root> == die.
```

Kim died.

```
Eat:
```

[12] To gain the intuitive sense of this DATR fragment, read a line such as `<> == VERB` as "inherit everything from the definition of `VERB`", and a line such as `<parent> == PPTREE:<>` as "inherit the `parent` subtree from the definition of `PPTREE`". Inheritance in DATR is always by default – locally defined feature specifications take priority over inherited ones.

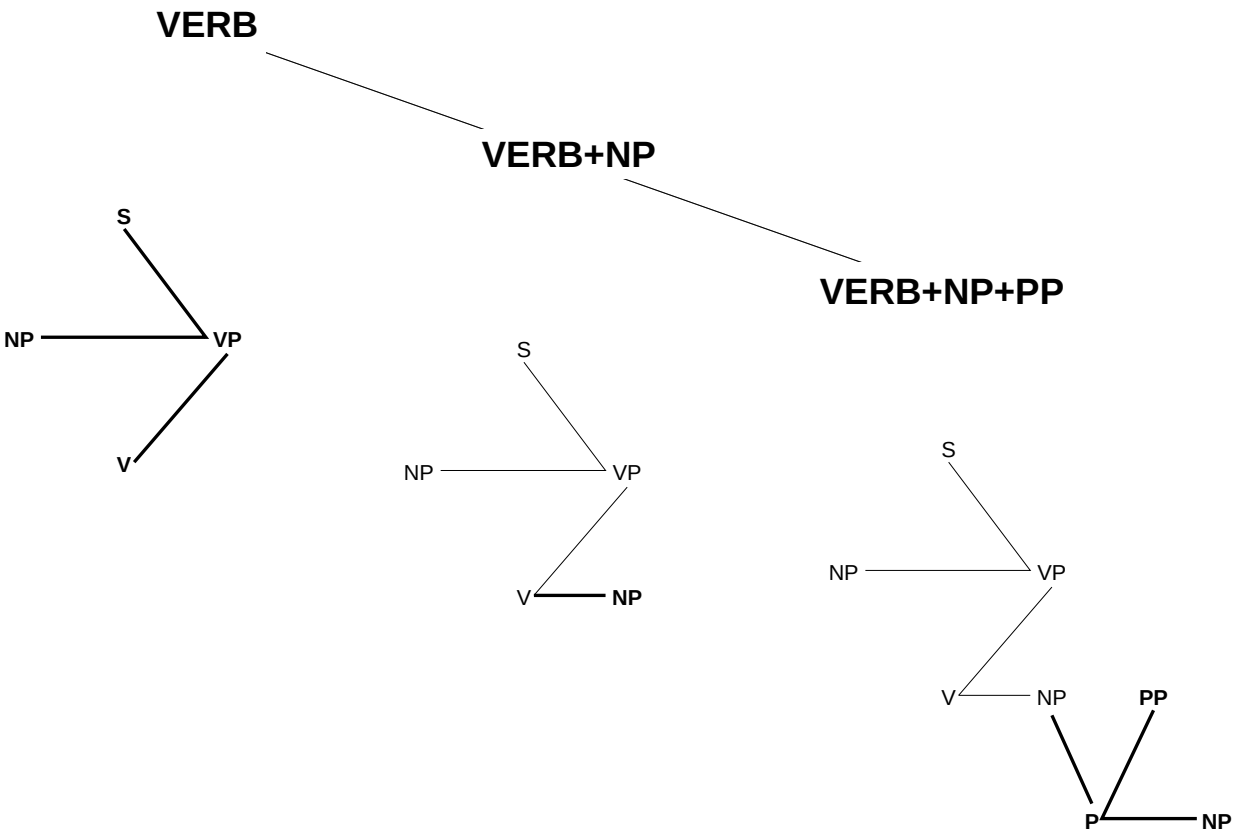

Figure 4: Inheriting tree structure

```
    <> == VERB+NP
    <root> == eat.
```

Sandy ate the meal.

```
Give:
    <> == VERB+NP+PP
    <root> == give.
```

Lee gave a book to Hilary.

```
Spare:
    <> == VERB+NP+NP
    <root> == spare.
```

Rover spared Felix the story of his life.

The references to `TREENODE, VPTREE, NPCOMP` and `PTREE` are links into the second level of structure, which we shall define shortly, so ignoring those, we see that `VERB` defines basic features for all verb entries (and can be used directly for intransitives such as `Die`), `VERB+NP` inherits from `VERB` but adds an NP complement to the right of the verb (for transitives), `VERB+NP+PP` inherits from `VERB+NP` but adds a further PP complement and so on. Entries for regular verb lexemes are then minimal – syntactically they just inherit **everything** from the abstract definitions.

The second level of sharing occurs within the syntactic trees themselves. The following statements complete the fragment, by providing definitions for this internal structure:

```
TREENODE:
    <> == undef
    <type> == internal.

STREE:
    <> == TREENODE
    <cat> == s.

VPTREE:
    <> == TREENODE
    <cat> == vp
    <parent> == STREE:<>
    <left> == NPCOMP:<>.

NPCOMP:
    <> == TREENODE
    <cat> == np
    <type> == substitution.

PPTREE:
    <> == TREENODE
    <cat> == pp.

PTREE:
    <> == TREENODE
    <cat> ==  p
    <type> == anchor
    <parent> == PPTREE:<>
```

At this level of representation, the DATR definitions correspond to different types of node within a syntactic tree – any such node may have category information, node type information (`internal`, `foot` or `anchor`) or subtree information (`left`, `right`, or `parent` subtrees, which are themselves nodes similarly defined). `TREENODE` represents the most abstract node definition, containing common or default information for all other nodes. `STREE` is singleton node with syntactic category `s` – the sentential root node of the standard verb tree. `VPTREE` is the verb phrase subtree in the standard verb tree - with an `STREE` parent and an `NPCOMP` left subtree (the subject NP). The other nodes similarly define fragments of the trees represented. The verb tree itself is not defined here. Instead, `VERB` is itself defined to be a `TREENODE` (and hence `VERB+NP`, `VERB+NP+PP`, etc., are also). This is really an expositional simplification: a more thorough treatment might draw a sharper distinction between nodes in the principal hierarchy and nodes in syntactic trees.

Taken together, these definitions provide a specification for `Give` just as we had it before, but with the addition of `type` and `root` features. They also support some other verbs too, and it should be clear that the basic technique extends readily to a wide range of other verbs and other parts of speech. Also, although the trees we have described are all **initial** trees (in LTAG terminology), we can describe **auxiliary** trees, which include a leaf node of type `foot` just as easily. A simple example is provided by the following definition for auxiliary verbs:

```
AUXVERB:
    <> == TREENODE
    <cat> == v
    <type> == anchor
    <parent cat> == vp
    <right cat> == vp
    <right type> == foot.
```

6 Lexical rules

Having established a basic structure for our LTAG lexicon, we now turn our attention towards capturing other kinds of relationship among trees. We noted above that lexical entries are actually associated with **tree families**, and that these group together trees that are related to each other. Thus in the same family as a standard ditransitive verb, we might find the full passive, the agentless passive, the dative alternation, the various relative clauses, and so forth. It is clear that these families correspond closely to the outputs of transformations or metarules in other frameworks, but the XTAG system currently has no formal component for describing the relationships among families nor mechanisms for generating them. And so far we have said nothing about them either – we have only characterized single trees.

However, LTAG's large domain of locality means that **all** such relationships can be viewed as directly lexical, and thus expressible by lexical rules. In fact we can go further than this: because we have embedded the domain of these lexical rules, namely the LTAG tree structures, within the feature structures, we can view such lexical rules as covariation constraints within feature structures, in much the same way that the covariation of, say, syntactic and morphological form is treated. In particular, we can use the mechanisms that the lexical knowledge representation language already provides for feature covariation, rather than having to invoke in addition some special purpose lexical rule machinery.

We consider six construction types found in the XTAG grammar: passive, dative, subject-auxiliary inversion, *wh*-questions, relative clauses and topicalisation. Our basic approach to each of these is the same. Lexical rules are specified by defining a derived `output` tree structure in terms of an `input` tree structure, where each of these structures is a set of feature specifications of the sort defined above. Each lexical rule has a name, and the input and output tree structures for rule `foo` are referenced by prefixing feature paths of the sort given above with `<input foo ...>` or `<output foo ...>`. So for example, the category of the parent tree node of the output of the passive rule might be referenced as `<output passive parent cat>`. We define a very general default, stating that the `output` is the same as the `input`, so that lexical relationships need only concern themselves with components they modify. This approach to formulating lexical rules is quite general and in no way restricted to LTAG: it can be readily adapted for application in the context of any feature-based lexicalist grammar formalism.

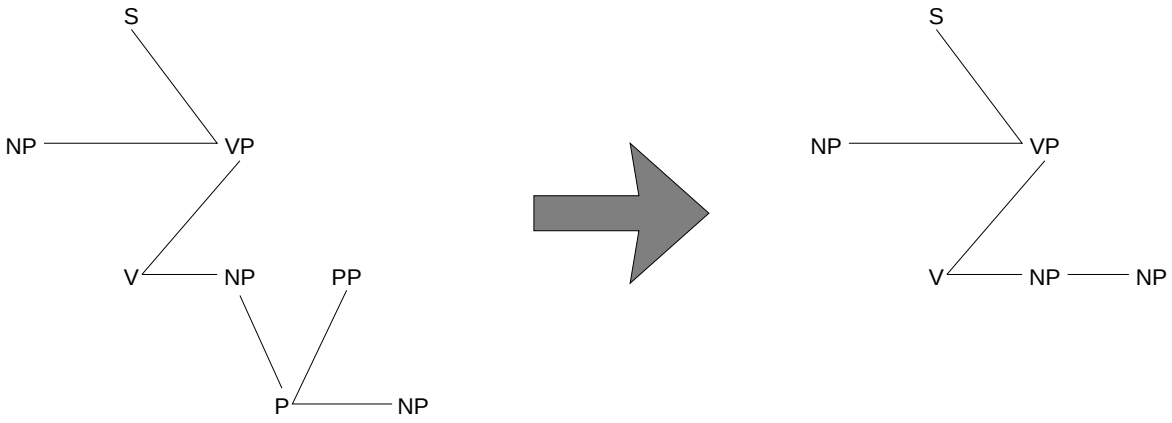

Figure 5: Lexical rule for dative

Using this approach, the dative lexical rule can be given a minimalist implementation by the addition of the following single line to `VERB+NP+PP`, defined above.

```
VERB+NP+PP:
    <output dative right right> == NPCOMP:<>.
```

This causes the second complement to a ditransitive verb in the dative alternation to be an NP, rather

than a PP as in the unmodified case (see figure 5)[13].

Lee gave Hilary a book.

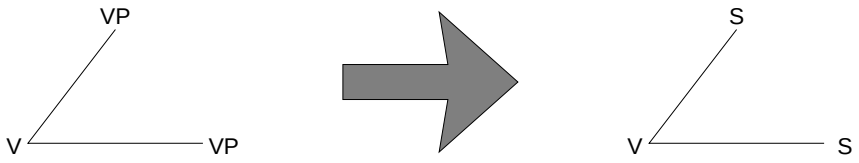

Figure 6: Lexical rule for auxiliary inversion

Subject-auxiliary inversion can be achieved similarly by just specifying the `output` tree structure without reference to the `input` structure (see figure 6):

```
AUXVERB:
    <output auxinv form> == finite-inv
    <output auxinv parent cat> == s
    <output auxinv right cat> == s.
```

Did Kim die?

Passive is slightly more complex, in that it has to modify the given `input` tree structure rather than simply overwriting part of it. For agentless passives, the necessary additions to the `VERB+NP` node are as follows[14]:

```
VERB+NP:
    <output passive form> == passive
    <output passive right> == "<input passive right right>".
```

The meal was eaten.

Here, the first line stipulates the form of the verb in the output tree to be passive, while the second line redefines the complement structure: the `output` of passive has as its first complement the second complement of its `input`, thereby discarding the first complement of its `input`. Since complements are daisy-chained, all the others move up too.

The definitions for passive occur at the `VERB+NP` node, but the definition is inherited by subclasses, so that passive applies to `VERB+NP+PP`, `VERB+NP+S` etc. as well. Figure 7 shows its effect on a standard ditransitive (`VERB+NP+PP`) verb tree. Of course this inheritance is only by default: individual transitive verbs, or whole subclasses, can override it, leaving their passive tree structure undefined if required.

Wh-questions, relative clauses and topicalisation are slightly different, in that the application of the lexical rule causes structure to be added to the top of the tree (above the s node). Although these constructions involve unbounded dependencies, the unboundedness is taken care of by the LTAG adjunction mechanism: for lexical purposes the dependency is local. Since the relevant lexical rules can apply to sentences that contain any kind of verb, they are stated at the `VERB` node. Thus, for example, topicalisation, *wh*-questions and relative clauses can be defined as follows:

[13] Of course if we are concerned about argument structure in our analysis, this overwriting version is insufficient because it fails to track the movement of the semantic nominal arguments. However, the <`input`>/<`output`> formulation makes it easy to extend the dative rule to make the output direct object inherit its argument structure from the input indirect object, and the output indirect object inherit from the input direct object.

[14] Oversimplifying slightly, the double quotes in "`<input passive right right>`" mean that that DATR path will not be evaluated locally (i.e., at the `VERB+NP` node), but rather at the relevant lexeme node (e.g., `Eat` or `Give`).

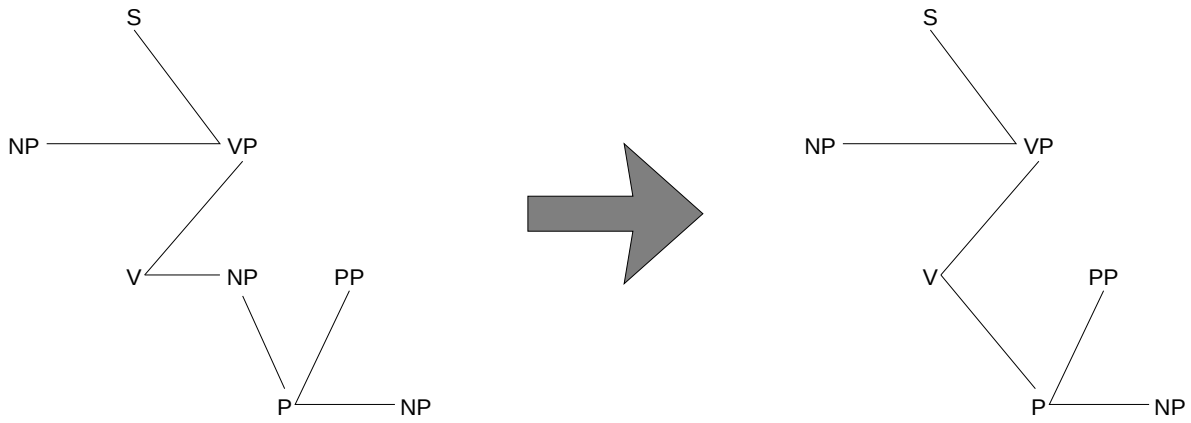

Figure 7: Lexical rule for passive

```
VERB:
    <output topic parent parent parent cat> == s
    <output topic parent parent left cat> == np
    <output topic parent parent left form> == normal
    <output whq> == "<output topic>"
    <output whq parent parent left form> == wh

    <output rel> == "<output topic>"
    <output rel parent parent left form> == rel
    <output rel parent parent parent> == NTREE:<>
    <output rel parent parent left> == NPCOMP:<>
    <output rel parent parent left type> == foot.
```

The meal, Sandy ate.
I asked what Sandy ate.
The meal which Sandy ate was expensive.

Here an additional NP and S are attached above the original S node to create a topicalised structure (see figure 8). The *wh*-rule inherits from the topicalisation rule, changing just one thing: the form of the new NP is marked as `wh`, rather than as `normal`. Finally the relative clause rule also inherits from topicalisation, this time marking the NP as a relative, but adds additional structure above the **new** S node, turning the tree into an auxiliary tree (in LTAG terminology).

Of course, one important aspect of these rules not captured by these definitions is the fact that somewhere within the rest of the modified tree structure is an NP gap, corresponding to the fronted NP in the tree fragment added by the rules. Although the LTAG formulation of these constructions avoids the unboundedness of this correspondence, it remains the case that the location of this NP gap is very underspecified – it can be almost any NP occurring in the input tree. One way of accommodating this is to have many different versions of each rule, one for each possible gap location. Another is to introduce a more powerful nondeterministic matching operation of a rule against its input tree, so different match solutions correspond to different gap locations. Our approach to this problem lies between these two alternatives. However, before describing it in more detail, we need to look at how lexical rules of this sort are 'applied'.

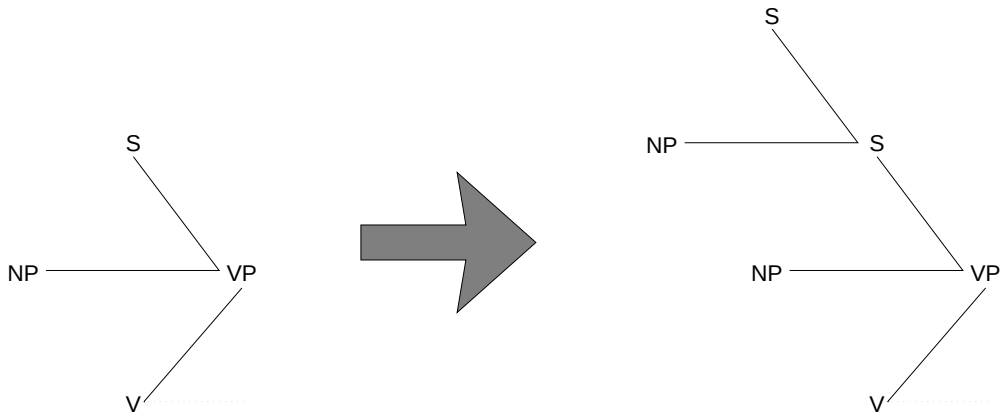

Figure 8: Lexical rule for topicalisation

7 Applying lexical rules

As explained above, each lexical rule is defined to operate on its own notion of an `input` and produce its own `output`. In order for the rules to have an effect, the various `input` and `output` paths have to be linked together using inheritance, creating a chain of inheritances between the base, that is, the canonical definitions we introduced in section 5, and `surface` tree structures of the lexical entry. For example, to 'apply' the dative rule to our `Give` definition, we could construct a definition such as this:

```
Give-dat:
    <> == Give
    <input dative> == <>
    <surface> == <output dative>.
```

Values for paths prefixed with `surface` inherit from the output of the dative rule. The input of the dative rule inherits from the base (unprefixed) case, which inherits from `Give`. The dative rule definition (just the one line introduced above, plus the default that output inherits from input) thus mediates between `Give` and the surface of `Give-dat`.

In a similar way longer sequences of rule application might be constructed by chaining such inheritances together. So applying passive to the dative of `Give` might look like this:

```
Give-dat-pass:
    <> == Give
    <input dative> == <>
    <input passive> == <output dative>
    <surface> == <output passive>.
```

Hilary was given a book.

and so on. While this approach will work, and indicates what relationships need to be established, it clearly misses a generalization about the way the lexicon works. Ideally, what we would like to specify is simply the name of the rule(s) to be applied[15], as illustrated below:

```
Word1:
```

[15] Perhaps also with some ordering information, although in these examples, the order of application is always completely determined.

```
    <> == Give
    <alt dative> == true
    <alt passive> == true.
```

Here, we have created a word which is an instance of `Give` to which both dative and passive have been applied. The definition of `Give` specifies how to apply dative and passive (via `VERB+NP+PP` and `VERB+NP`respectively), the word definition specifies which ones should actually be applied, in the same way that it might specify person, number and tense. These specifications suffice to ensure that the subcategorisation for Word1 will be appropriate for a passive, dative form.

We can achieve this in the following way. Let us assume a top node `LTAG_ENTRY` in the principal lexical hierarchy from which `VERB`, `NOUN`, etc., all inherit[16]. The required chaining together of inheritance specifiers is achieved at this node as the 'ultimate default'. Here is a possible definition:

```
#vars $rname: topic whq rel auxinv dative passive surface.

LTAG_ENTRY:
    <alt> == false
    <output> == "<input>"
    <surface> == <input surface>
    <input $rname> == "<NEXTRULE:<$rname>>".
```

The first equation of the `LTAG_ENTRY` definition simply makes unspecified alternations default to false, while the second is the blanket default supporting the lexical rule definitions, now moved to its rightful place in `LTAG_ENTRY` rather than `VERB`. The third line is a convenience definition – if a lexical entry query is prefixed with <`surface`>, rewrite it (locally) as <`input surface`> to be caught by the following definition. This just allows us to treat the surface as though it were just another lexical rule in the chain (the symmetry we thereby achieve is clearer in the definition for `Give-dat-pass` above).

The fourth statement is the key. It is invoked whenever a lexical rule (or the surface query, because of the preceding line) accesses its input – <`input`> is in general not specified anywhere else in the hierarchy. Its definition uses the node `NEXTRULE` to return the sequence `output` X, where X is the name of the rule to be applied before `$rname`[17]. To achieve the effect in `Give-dat-pass`, when `$rname` is `passive`, `NEXTRULE` returns `output dative`. This line achieves one level of inheritance linking – references to <`input` X> will invoke it again, resulting in the output of the rule to apply before that, etc. ...

The problem of inheritance linking has thus been reduced to one of rewriting rule names in the node `NEXTRULE`. `NEXTRULE` needs to take account of two things: the alternations requested in the surface node, and constraints on rule application. The present analysis encodes the following constraints in a fairly rudimentary fashion: dative precedes passive which precedes topicalisation, *wh*-question and relative clauses, which are mutually exclusive. Here are the definitions[18]:

```
NEXTRULE:
    <> ==
    <surface> == CONSTRAINT1:<>
    <topic>   == CONSTRAINT2:<>
    <whq>     == CONSTRAINT2:<>
    <rel>     == CONSTRAINT2:<>
    <passive> == CONSTRAINT3:<>.
CONSTRAINT1:
```

[16] The earlier examples have been simplified to the point that they cannot support this directly – `VERB` inherits from `TREE_NODE`. However, the full fragment underlying this work is available by anonymous `FTP` from ftp://ftp.cogs.sussex.ac.uk/pub/nlp/DATR/dtr/lexrltag.dtr

[17] `$rname` is a variable which will be instantiated to the rule name whose input we are seeking.

[18] Candito (1996, 195) complains that our 1995 paper, in common with the other approaches she considers, fails to discuss "the automatic triggering, ordering and bounding of ... lexical rules". We thought that we had discussed such matters, albeit briefly, on pages 81-82 of that paper. However, we provide more here.

```
    <> == <check "<alt topic>" "<alt whq>" "<alt rel>">
    <check> == UNDEF
    <check true false false> == output topic
    <check false true false> == output whq
    <check false false true> == output rel
    <check false false false> == CONSTRAINT2:<>.
CONSTRAINT2:
    <> == <check "<alt passive>">
    <check> == UNDEF
    <check true> == output passive
    <check false> == CONSTRAINT3:<>.
CONSTRAINT3:
    <> == <check "<alt dative>">
    <check> == UNDEF
    <check true> == output dative
    <check false> == .
```

NEXTRULE is invoked with the name of the rule whose input is required, and so works backwards through the ordering constraints. If the input is <surface>, CONSTRAINT1 looks to see if any of <alt topic>, <alt whq> and <alt rel> is specified – if at most one is present, it is returned as the result, if none are specified, control passes to CONSTRAINT2, otherwise (i.e., if more than one is specified) the result is undefined. CONSTRAINT2 is invoked if topic, whq or rel request inputs, or if none of them is specified, and checks for <alt passive> – if specified it returns output passive, otherwise passes control to CONSTRAINT3 which operates similarly for dative. Finally, if dative is not specified (or requests input), an empty value is returned which make the inheritance drop down to the basic definitions (without any rule application prefixes).

In these definitions, we see some of the complexity of our first passive example re-emerging as we expected. Nevertheless, we have achieved quite powerful results without additional formal devices, and without working the ones we have very hard[19]. It remains difficult to see that anything here is qualitatively different from our earlier morphological descriptions. The closest one might come is to argue that the mechanism for applying lexical rules that we have just described is perhaps a self-contained external mechanism masquerading as internalized. But even here the case is weak: in our relatively simple exposition all the lexical rules are handled in one place and all the constraints are locally defined, but nothing prevents these mechanism from being distributed through the hierarchy as well.

Let us now return to the issue of the gaps in unbounded dependency constructions. Consider Word2, which properly describes a *wh*-question based on the agentless passive of the dative of Give.

```
Word2:
    <> == Give
    <alt whq> == true
    <alt dative> == true
    <alt passive> == true.
    <parent left form> == null
```

I wonder who was given a book.

Notice here the final line of Word2 marks the subject NP as null. This is the mechanism we use to overcome the problem of underspecificity of the gap location that we discussed above. Whenever an unbounded dependency rule is invoked, in addition to specifying the rule application itself (using the line <alt whq> == true), we also specify the gap, by marking it as null. In effect we are modifying the original

[19] Although presented in a rather different formal framework, our project here is not dissimilar to that of Bouma (1997). Whilst we deprive 'Lexical Rules' of any special ontological status by describing the relevant facts with routine lexical machinery that is already motivated by other lexical phenomena, he eliminates them in favour of a more pervasive application of the recursive constraints that already form part of the HPSG lexical toolkit.

input tree (as defined by `Give`) before we pass it to the rule. This means that the rule itself does not need to mark the gap as null in the output (since by default, the output inherits from the input). But it does use the null specification to cross-reference the fronted NP (in this case the *wh*-NP and the null NP) – the fronted NP is assigned a feature `coindex` whose value is the tree address of the null NP.

This approach avoids the redundancy of multiple instances of each rule, essentially by using the tree address of the gap as a parameter to rule application. It also avoids the need for a nondeterministic matching process to locate the gap in the input – although the full fragment includes support for *locating* the gap in the input (and hence specifying the `coindex` feature), this is an entirely deterministic process.

We can, as we have seen, encode constraints on the applicability of rules in the mapping from boolean flags to actual inheritance specifications. Thus, for example, `whq, rel,` and `topic` are mutually exclusive. If such constraints are violated, then no value for `surface` gets defined. Thus `Word3` improperly attempts topicalisation in addition to *wh*-question formation, and, as a result, will fail to define a `surface` tree structure at all:

```
Word3:
    <> == Give
    <alt whq> == true
    <alt topic> == true
    <alt dative> == true
    <alt passive> == true
    <parent left form> == null.
```

This approach to lexical rules allows them to be specified at the appropriate point in the lexical hierarchy, but overridden or modified in subclasses or lexemes as appropriate. It also allows default generalisation over the lexical rules themselves, and control over their application. The last section showed how the `whq` lexical rule could be built by a single minor addition to that for topicalisation. However, it is worth noting that the lexical rules presented here are **rule instances** which can only be applied once to any given lexeme – multiple application could be supported, by making multiple instances inherit from some common rule specification, but in our current treatment such instances would require different rule names.

8 Comparison with related work

It is worth pausing to compare our approach with a more conventional unification-based view. A common approach is to encode lexical rules as pairs of feature structures, such that when a lexical entry is unified with one of the pair (the 'input'), the other of the pair becomes instantiated as the 'output'. There are clear parallels here between this and our use of input and output features with chaining of inheritance specifications to achieve rule 'application'. But such rules are typically separate from the lexicon proper, and manipulated by an independent control mechanism with its own notions of productivity and constraint.

To take a specific example, in the lexical description language LAUREL, Copestake (1992) introduces lexical rule structures containing two (or more) lexical-sign-valued subfeatures corresponding to the input(s) and output of the rule. By making such lexical rules first class members of the lexicon, she can bring her typed nonmonotonic inheritance mechanisms to bear in their specification, and thus achieve the kind of interdependencies among rules that we illustrated above. But they remain an independent component, not linked into the inheritance structure for the lexical entries in any way, and with a mechanism for controlling application which is entirely outside the lexicon.

LAUREL's overall approach would admit internalization of the rule definitions in a manner analogous to ours: instead of a free-standing rule object such as `grinding` with input and output sign-valued subfeatures (`1` and `0` respectively, in Copestake's formulation), new features `<1 grinding>` and `<0 grinding>` might be added to the ordinary lexical sign structures, and defined at a suitably high point in the lexical hierarchy. As well as removing a distinct component from her lexical ontology, this might result in simpler specifications for the rules (particularly their input specifications) through exploitation of the main lexical inheritance structures, just as it does for us.

However, the **application** of such rules would still be controlled from outside the lexicon: each lexical entry would now 'know' directly which lexical rules can apply to it, but the effects of applying a rule would only be possible after additional unifications have been carried out (unifying the entry itself with one of its own the rule input subfeatures). To achieve the kind of 'dynamic' unification equivalent to the mechanism we described above, the feature logic would need to be powerful enough to support logical implication, so that the presence of a trigger feature such as `<alt grinding>` would imply the unification of the lexical sign itself with its `<1 grinding>` subfeature.

As noted above, Vijay-Shanker & Schabes (1992) have also proposed an inheritance-based approach to the problem of efficient LTAG lexical representation. They use monotonic inheritance to build up partial descriptions of trees: each description is a finite set of dominance, immediate dominance and linear precedence statements about tree nodes in a tree description language developed by Rogers & Vijay-Shanker (1992), and category information is located in the node labels.

This differs from our approach in a number of ways. First, our use of nonmonotonic inheritance allows us to manipulate total instead of partial descriptions of trees. The abstract verb class in the Vijay-Shanker & Schabes account subsumes both intransitive and transitive verb classes but is not identical to either – a minimal-satisfying-model step is required to map partial tree descriptions into actual trees. In our analysis, `VERB` **is** the intransitive verb class, with complements specifically marked as undefined: thus `VERB:<right> == undef` is inherited from `TREENODE` and `VERB+NP` just overrides this complement specification to add an NP complement. Second, we describe trees using only local tree relations (between adjacent nodes in the tree), while Vijay-Shanker & Schabes also use a nonlocal dominance relation.

Both these properties are crucial to our embedding of the tree structure in the feature structure. We want the category information at each tree node to be partial in the conventional sense, so that in actual use such categories can be extended (by unification or whatever). So the feature structures that we associate with lexical entries must be viewed as partial. But we do **not** want the tree structure to be extendible in the same way: we do not want an intransitive verb to be applicable in a transitive context, by unifying in a complement NP. So the tree structures we define must be total descriptions[20]. And of course, our use of only local relations allows a direct mapping from tree structure to feature path, which would not be possible at all if nonlocal relations were present.

So while these differences may seem small, they allow us to take this significant representational step – significant because it is the tree structure embedding that allows us to view lexical rules as feature covariation constraints. The result is that while Vijay-Shanker & Schabes use a tree description language, a category description language and a further formalism for lexical rules, we can capture everything in one framework all of whose components (nonmonotonicity, covariation constraint handling, etc.) have already been independently motivated for other aspects of lexical description.

Candito (1996) presents a variant of the Vijay-Shanker & Schabes approach that eliminates their lexical rule component but employs a three layer ("dimension") monotonic multiple inheritance hierarchy of partial (and possibly nonlocal) tree descriptions. Four components of this structure are then unified together under certain conditions (which she refers to as "principles"[21]) to produce a partial tree description from which a set of minimal-satisfying-models is computed. From a purely formal perspective, her approach is initially appealing. But the monotonicity is worrying from a descriptive or lexicon-maintenance point of view since it requires the full burden of lexical exceptionality to be carried in the syntactic lexicon. So far as we can judge from her paper, every lexical entry that is not completely regular will have to specify precisely which of the trees making up the regular tree family it excludes. There appears to be no scope for capturing subregularities across lexical entries or respecting interdependencies among trees in a family. It is not clear to us that this is a workable approach for syntax, but it does seem clear that it is unlikely to extend to an effective treatment of either semantics or morphology, where complex subregularities are common.

[20] Note that simplified fragment presented here does not get this right. It makes all feature specifications total descriptions. To correct this we would need to change `TREENODE` so that only the values of `<right>`, `<left>` and `<parent>` default to `undef`.

[21] Candito castigates all the alternative approaches to LTAG lexicons, including ours, as "purely technical" and unprincipled (p195). By contrast, in her own work, "the implementation of the principles gives a real generative power to the tool" (p95). The current fashion for linguists to describe their own work as "principled" or "principle-based" induces in us the same reaction that "culture" caused in Hanns Johst's storm-trooper. The quality of analyses is not enhanced by the adoption of self-congratulatory terminology.

Becker's work (1993; 1994) is also directed at the problems we address in the present paper. Like him, we have employed an inheritance hierarchy. And, like him, we have employed a set of lexical rules (corresponding to his metarules). The key differences between our account and his are (i) that we have been able to use an existing lexical knowledge representation language, rather than designing a formal system that is specific to LTAG, and (ii) that we have expressed our lexical rules in exactly the same language as that we have used to define the hierarchy, rather than invoking two quite different formal systems.

Becker's sharp distinction between his metarules and his hierarchy gives rise to some difficulties that our approach avoids. Firstly, he notes that his metarules are subject to lexical exceptions and proposes to deal with these by stating "for each entry in the (syntactic) lexicon ... which metarules are applicable for this entry" (1993,126). We have no need to carry over this use of (meta)rule features since, in our account, lexical rules are not distinct from any other kind of property in the inheritance hierarchy. They can be stated at the most inclusive relevant node and can then be overridden at the exceptional descendant nodes. Nothing specific needs to be said about the nonexceptional nodes.

Secondly, his metarules may themselves be more or less similar to each other and he suggests (1994,11) that these similarities could be captured if the metarules were also to be organized in a hierarchy. However, our approach allows us to deal with any such similarities in the main lexical hierarchy itself[22] rather than by setting up a separate hierarchical component just for metarules (which appears to be what Becker has in mind).

Thirdly, as he himself notes (1993,128), because his metarules map from elementary trees that are in the inheritance hierarchy to elementary trees that are outside it, most of the elementary trees actually used are not directly connected to the hierarchy (although their derived status with respect to it can be reconstructed). Our approach keeps all elementary trees, whether or not they have been partly defined by a lexical rule, entirely within the lexical hierarchy.

In fact, Becker himself considers the possibility of capturing all the significant generalizations by using just one of the two mechanisms that he proposes: "one might want to reconsider the usage of one mechanism for phenomena in both dimensions" (1993,135). But, as he goes on to point out, his existing type of inheritance network is not up to taking on the task performed by his metarules because the former is monotonic whilst his metarules are not. However, he does suggest a way in which the hierarchy could be completely replaced by metarules but argues against adopting it (1993,136).

As will be apparent from the earlier sections of this paper, we believe that Becker's insights about the organization of an LTAG lexicon can be better expressed if the metarule component is replaced by an encoding of (largely equivalent) lexical rules that are an integral part of a nonmonotonic inheritance hierarchy that stands as a description of **all** the elementary trees.

9 Conclusion

Making a sharp distinction between 'Lexical Rules' and the lexical hierarchy gives rise to a variety of problems that our approach avoids. Lexical rules are typically subject to lexical exceptions and an external approach leads one to deal with these by stating "for each entry in the (syntactic) lexicon ... which [lexical] rules are applicable for this entry" (Becker 1993,126). We have no need to carry over this use of rule or exception features since, in our account, lexical rules are not distinct from any other kind of property in the inheritance hierarchy. They can be stated at the most inclusive relevant node and can then be overridden at the exceptional descendant nodes. Nothing specific needs to be said about the nonexceptional nodes.

Lexical rules themselves may also be more or less similar to each other which suggests that these similarities could be captured if the lexical rules themselves were to be organized in a second inheritance hierarchy (Becker 1994,11). However, our approach allows us to deal with any such similarities in the main lexical hierarchy rather than by setting up a separate hierarchical component just for lexical rules.

It is advantageous to express lexical rules in the same formal language as is used to express the lexical hierarchy since lexical rules themselves may well exhibit exactly the kinds of defaulty relations, one to

[22] As illustrated by the way in which the `whq` lexical rule inherits from that for topicalisation in the example given above.

another, that lexical classes do[23]. This approach to lexical rules allows them to be specified at the appropriate point in the lexical hierarchy, but overridden or modified in subclasses or lexemes as appropriate. It also allows default generalization over the lexical rules themselves, and control over their application. Thus, for example, an indirect *wh*-question lexical rule can be created by a single minor addition to that already defined for topicalisation. And a lexical rule for direct *wh*-questions may be a variant of that for indirect *wh*-questions: similar, sharing components, but not identical. With a suitable degree of abstraction, achieved by parameterization of the components, lexical rules can be reified in a lexical knowledge representation language like DATR, allowing one to inherit from another.

Acknowledgements

Precursors of this paper were presented at the September 1994 TAG+ Workshop in Paris, the June 1995 ACL meeting at MIT, and the August 1995 ACQUILEX Workshop on Lexical Rules in Huntingdon. We thank the referees for, and attendees at, those events for a number of helpful comments. We are also grateful to Lynne Cahill, Aravind Joshi, Bill Keller, Jim Kilbury, Owen Rambow, K. Vijay-Shanker and The XTAG Group. This research was partly supported by grants to Evans and Weir from EPSRC (UK) and to Gazdar from ESRC (UK).

References

Anne Abeille, Kathleen Bishop, Sharon Cote, & Yves Schabes. 1990. A lexicalized tree adjoining grammar for English. Technical Report MS-CIS-90-24, Department of Computer & Information Science, Univ. of Pennsylvania.

Tilman Becker. 1993. HyTAG*: A new type of Tree Adjoining Grammar for hybrid syntactic representation of free word order languages.* Ph.D. thesis, Univ. des Saarlandes.

Tilman Becker. 1994. Patterns in metarules. In *Proceedings of the Third International Workshop on Tree Adjoining Grammars*, 9–11.

Gosse Bouma. 1997. Valence alternations without lexical rules. In Jan Landsbergen, Jan Odijk, Kees van Deemter & Gert Veldhuijzen van Zanten, eds. *CLIN 1996, Papers from the Seventh Computational Linguistics in the Netherlands Meeting,* Eindhoven, 25–40.

Marie-Hélène Candito. 1996. A principle-based hierarchical representation of LTAGs. *COLING-96*, Copenhagen, 194-199.

Bob Carpenter. 1991. The generative power of categorial grammars and head-driven phrase structure grammars with lexical rules. *Computational Linguistics* 17, 301-313.

Bob Carpenter. 1992. Categorial grammars, lexical rules, and the English predicative. In Robert Levine, ed. *Formal Grammar: Theory and Implementation.* New York: Oxford University Press, 168-242.

Ann Copestake. 1992. The representation of lexical semantic information. PhD dissertation, University of Sussex, **CSRP 280.**

Christy Doran, Dania Egedi, Beth Ann Hockey, & B. Srinivas. 1994a. Status of the XTAG system. In *Proceedings of the Third International Workshop on Tree Adjoining Grammars*, 20–23.

Christy Doran, Dania Egedi, Beth Ann Hockey, B. Srinivas, & Martin Zaidel. 1994b. XTAG system — a wide coverage grammar for english. In *COLING-94*, 922–928.

Roger Evans & Gerald Gazdar. 1996. DATR: A language for lexical knowledge representation. *Computational Linguistics* **22.2,** 167-216

[23] See Lascarides et al. (1996, 84), and Krieger (1994, 279) who notes some other advantages.

Roger Evans, Gerald Gazdar & David Weir. 1995. Encoding lexicalized tree adjoining grammars with a nonmonotonic inheritance hierarchy. *33rd Annual Meeting of the Association for Computational Linguistics,* 77-84.

Daniel P. Flickinger. 1987. *Lexical Rules in the Hierarchical Lexicon.* Ph.D. thesis, Stanford Univ.

Gerald Gazdar. 1989. English verbal morphology and subcategorization (DATR fragment). In Roger Evans & Gerald Gazdar, eds. *The* DATR *Papers.* University of Sussex, **CSRP 139,** 79-84.

Bill Keller. 1995. DATR theories and DATR models. *Proceedings of the 33rd Annual Meeting of the Association of Computational Linguistics*, 55-62.

Bill Keller. 1996. An evaluation semantics for DATR theories. *COLING-96*, Copenhagen, 646-651.

James Kilbury. 1990. Encoding constituent structure in feature structures. Unpublished manuscript, Univ. of Duesseldorf, Duesseldorf.

Hans-Ulrich Krieger. 1994. Derivation without lexical rules. In C.J. Rupp, M.A. Rosner & R.L. Johnson, eds. *Constraints, Language and Computation.* London: Academic Press, 277-313.

Alex Lascarides, Ted Briscoe, Nicholas Asher & Ann Copestake. 1996. Order independence and persistent default unification. *Linguistics & Philosophy* **19.1,** 1-89.

James Pustejovsky. 1991. The generative lexicon. *Computational Linguistics* **17.4,** 409-441.

Graeme D. Ritchie, Graham J. Russell, Alan W. Black and Stephen G. Pulman. 1992. *Computational Morphology.* Cambridge, Ma.: MIT Press.

James Rogers & K. Vijay-Shanker. 1992. Reasoning with descriptions of trees. In *ACL-92*, 72–80.

K. Vijay-Shanker & Yves Schabes. 1992. Structure sharing in lexicalized tree-adjoining grammar. In *COLING-92*, 205–211.

John C. Wells. 1987. Computer coded phonetic transcription. *Journal of the International Phonetic Association* **17:2,** 94-114.

The XTAG Research Group. 1995. A lexicalized tree adjoining grammar for English. Technical Report IRCS Report 95-03, The Institute for Research in Cognitive Science, Univ. of Pennsylvania.