DATR: A Language for Lexical Knowledge Representation*

Roger Evans†
University of Brighton

Gerald Gazdar‡
University of Sussex

Much recent research on the design of natural language lexicons has made use of nonmonotonic inheritance networks as originally developed for general knowledge representation purposes in Artificial Intelligence. DATR *is a simple, spartan language for defining nonmonotonic inheritance networks with path/value equations, one that has been designed specifically for lexical knowledge representation. In keeping with its intendedly minimalist character, it lacks many of the constructs embodied either in general purpose knowledge representation languages or in contemporary grammar formalisms. The present paper shows that the language is nonetheless sufficiently expressive to represent concisely the structure of lexical information at a variety of levels of linguistic analysis. The paper provides an informal example-based introduction to* DATR *and to techniques for its use, including finite state transduction, the encoding of DAGs and lexical rules, and the representation of ambiguity and alternation. Sample analyses of phenomena such as inflectional syncretism and verbal subcategorisation are given which show how the language can be used to squeeze out redundancy from lexical descriptions.*

1. Introduction

Irregular lexemes are standardly regular in some respect. Most are just like regular lexemes except that they deviate in one or two characteristics. What is needed is a natural way of saying "this lexeme is regular except for this property". One obvious approach is to use nonmonotonicity and inheritance machinery to capture such lexical irregularity (and subregularity), and much recent research into the design of representation languages for natural language lexicons has thus made use of nonmonotonic inheritance networks (or "semantic nets") as originally developed for more general representation purposes in Artificial Intelligence. Daelemans et al. (1992) provide a rationale for, and an introduction to, this body of research and we will not rehearse the content of that paper here, nor review the work cited there[1]. DATR is a rather spartan nonmonotonic language for defining inheritance networks with path/value equations. In keeping with its intendedly minimalist character, it lacks many of the constructs embodied either in general purpose

* This edition of the present paper is to appear in *Computational Linguistics*.

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

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

1 Daelemans & Gazdar (1992) and Briscoe et al. (1993) are collections that bring together much recent work on the application of inheritance networks to lexical description. Other relevant recent work not found there includes Bouma (1993), Briscoe et al. (1995), Calder (1994), Copestake (1992), Daelemans (1994), Daelemans & De Smedt (1994), Ide et al. (1994), Lascarides et al. (forthcoming), Mellish & Reiter (1993), Mitamura & Nyberg (1992), Penn & Thomason (1994), Reiter & Mellish (1992), Young (1992), and Young & Rounds (1993).

knowledge representation languages or in contemporary grammar formalisms. But the present paper seeks to show that the language is nonetheless sufficiently expressive to represent concisely the structure of lexical information at a variety of levels of language description.

The development of DATR has been guided by a number of concerns which we summarise here. Our objective has been a language which (i) has an explicit theory of inference, (ii) has an explicit declarative semantics, (iii) can be readily and efficiently implemented, (iv) has the necessary expressive power to encode the lexical entries presupposed by work in the unification grammar tradition, and (v) can express all the evident generalisations and subgeneralisations about such entries. Our first publications on DATR (Evans & Gazdar 1989a, 1989b) provided a formal theory of inference (i) and a formal semantics (ii) for DATR and we will not recapitulate that material here[2].

With respect to (iii), the core inference engine for DATR can be coded in a page of Prolog (see, e.g., Gibbon 1993, p50). At the time of writing, we know of a dozen different implementations of the language, some of which have been used with large DATR lexicons in the context of big NLP systems (e.g., Andry et al. 1992; Cahill 1993a, 1994; Cahill & Evans 1990). We will comment further on implementation matters in Section 5, below. However, the main purpose of the present paper is to exhibit the use of DATR for lexical description (iv) and the way it makes it relatively easy to capture lexical generalisations and subregularities at a variety of analytic levels (v). We will pursue (iv) and (v) in the context of an informal example-based introduction to the language and to techniques for its use, and we will make frequent reference to the DATR-based lexical work that has been done since 1989.

The paper is organized as follows. Section 2 uses an analysis of English verbal morphology to provide an informal introduction to DATR. Section 3 describes the language more precisely: its syntax, inferential and default mechanisms, and the use of abbreviatory variables. Section 4 describes a wide variety of DATR techniques, including case constructs and parameters, boolean logic, finite state transduction, lists and DAGs, lexical rules, and ways to encode ambiguity and alternation. Section 5 explores more technical issues relating to the language, including functionality and consistency, multiple-inheritance, modes of use and existing implementations. Section 6 makes some closing observations. Finally an appendix to the paper replies to the points made in the critical literature on DATR.

2. DATR by example

We begin our presentation of DATR with a partial analysis of morphology in the English verbal system. In DATR, information is organised as a network of **nodes**, where a node is essentially just a collection of closely related information. In the context of lexical description, a node typically corresponds to a word, a lexeme or a class of lexemes. For example, we might have a node describing an abstract verb, another for the subcase of a transitive verb, another for the lexeme *love* and still more for the individual words that are instances of this lexeme (*love*, *loves*, *loved*, *loving*, etc.). Each node has associated with it a set of **path/value** pairs where a path is a sequence of **atoms** (which are primitive objects), and a value is an atom or a sequence of atoms. We will sometimes refer to atoms in paths as **attributes**.

2 Note, however, that the definitions in the 1989 papers are not given in sufficient generality to cover DATR equations with more than one (non-atomic) descriptor on the right hand side. Keller (1995) effectively replaces our 1989 presentation of a semantics for DATR and his treatment is general enough to cover descriptor sequences.

Path	Value
`syn cat`	`verb`
`syn type`	`main`
`syn form`	`present participle`
`mor form`	`love ing`

Table 1
Path/value pairs for present participle of *love*

For example, a node describing the present participle form of the verb *love* (and called perhaps `Word1`) might contain the path/value pairs shown in Table 1. The paths in this example all happen to contain two attributes, and the first attribute can be thought of as distinguishing syntactic and morphological types of information. The values indicate appropriate linguistic settings for the paths for a present participle form of *love*. Thus its syntactic category is `verb`, its syntactic type is `main` (i.e., it is a main verb, not an auxiliary), its syntactic form is `present participle` (a two atom sequence), its morphological form is `love ing` (another two atom sequence). In DATR this can be written as[3]:

```
Word1:
    <syn cat>  = verb
    <syn type> = main
    <syn form> = present participle
    <mor form> = love ing.
```

Here, angle brackets <...> delimit paths. Note that values can be atomic or they can consist of **sequences** of atoms, as the two last lines of the example illustrate[4]. As a first approximation, nodes can be thought of as denoting partial functions from paths (sequences of atoms) to values (sequences of atoms)[5].

In itself, this tiny fragment of DATR is not persuasive, apparently allowing only for the specification of words by simple listing of path/value statements for each one. It seems that if we wished to describe the passive form of *love* we would have to write:

```
Word2:
    <syn cat>  = verb
    <syn type> = main
    <syn form> = passive participle
    <mor form> = love ed.
```

This does not seem very helpful: the whole point of a lexical description language is to capture generalisations and avoid the kind of duplication evident in the specification of `Word1` and `Word2`. And indeed, we shall shortly introduce an inheritance mechanism which allows us to do just that. But there is one sense in which this listing approach is exactly what we want: it represents the actual information we generally wish to access from the description. So in a sense we do want all the above statements to be present in our description; what we want to avoid is repeated **specification** of the common

3 The syntax of DATR, like its name and its minimalist philosophy, owes more than a little to that of the unification grammar language PATR (Shieber 1986). With hindsight this may have been a bad design decision since similarity of syntax tends to imply a similarity of semantics. And, as we shall see in Section 4.7 below, and elsewhere, there is a subtle but important semantic difference.

4 Node names and atoms are distinct but essentially arbitrary classes of tokens in DATR. In this paper we shall distinguish them by a simple case convention – node names start with an uppercase letter, atoms do not.

5 This is an approximation since it ignores the role of global contexts – see Section 5.1, below.

elements.

This problem is overcome in DATR in the following way: such exhaustively listed path/value statements are indeed present in a description, but typically only **implicitly** present. Their presence is a logical consequence of a second set of statements, which have the concise, generalisation-capturing properties we expect. To make the distinction sharp, we call the first type of statement **extensional** and the second type **definitional**. Syntactically, the distinction is made with the equality operator: for extensional statements (as above), we use `=`, while for definitional statements we use `==`. And, although our first example of DATR consisted entirely of extensional statements, almost all the remaining examples will be definitional. The semantics of the DATR language binds the two together in a declarative fashion, allowing us to concentrate on concise definitions of the network structure from which the extensional "results" can be read off.

Our first step towards a more concise account of `Word1` and `Word2` is simply to change the extensional statements to definitional ones:

```
Word1:
    <syn cat>  == verb
    <syn type> == main
    <syn form> == present participle
    <mor form> == love ing.
Word2:
    <syn cat>  == verb
    <syn type> == main
    <syn form> == passive participle
    <mor form> == love ed.
```

This is possible because DATR respects the unsurprising condition that if at some node a value is specifically **defined** for a path with a definitional statement, then the corresponding extensional statement also holds. So the statements we previously made concerning `Word1` and `Word2` remain true, but now only **implicitly** true.

Although this change does not itself make the description more concise, it allows us to introduce other ways of describing values in definitional statements, in addition to simply specifying them. Such value **descriptors** will include inheritance specifications which allow us to gather together the properties that `Word1` and `Word2` have solely by virtue of being verbs. We start by introducing a `VERB` node:

```
VERB:
    <syn cat>  == verb
    <syn type> == main.
```

and then redefine `Word1` and `Word2` to inherit their verb properties from it. A direct encoding for this is as follows:

```
Word1:
    <syn cat>  == VERB:<syn cat>
    <syn type> == VERB:<syn type>
    <syn form> == present participle
    <mor form> == love ing.
Word2:
    <syn cat>  == VERB:<syn cat>
    <syn type> == VERB:<syn type>
    <syn form> == passive participle
    <mor form> == love ed.
```

In these revised definitions the right hand side of the `<syn cat>` statement is not a direct value specification, but instead an inheritance descriptor. This is the simplest form of

DATR inheritance, it just specifies a new node and path from which to obtain the required value. It can be glossed roughly as "the value associated with `<syn cat>` at `Word1` is the same as the value associated with `<syn cat>` at `VERB`". Thus from `VERB:<syn cat> == verb` it now follows that `Word1:<syn cat> == verb`[6].

However, this modification to our analysis seems to make it less rather than more concise. It can be improved in two ways. The first is really just a syntactic trick: if the path on the right hand side is the same as the path on the left hand side it can be omitted. So we can replace `VERB:<syn type>`, in the example above, with just `VERB`. We can also extend this abbreviation strategy to cover cases like the following, where the path on the right hand side is different but the node is the same:

```
Come:
    <mor root> == come
    <mor past participle> == Come:<mor root>.
```

In this case we can simply omit the node:

```
Come:
    <mor root> == come
    <mor past participle> == <mor root>.
```

The other improvement introduces one of the most important features of DATR – specification by default. Recall that paths are **sequences** of attributes. If we understand paths to start at their left hand end, we can construct a notion of path **extension**: a path *P2* extends a path *P1* if and only if all the attributes of *P1* occur in the same order at the left hand end of *P2* (so `<a1 a2 a3>` extends `<>`, `<a1>`, `<a1 a2>` and `<a1 a2 a3>`, but not `<a2>`, `<a1 a3>`, etc..). If we now consider the (finite) set of paths occurring in definitional statements associated with some node, that set will not include all possible paths (of which there are infinitely many). So the question arises of what we can say about paths for which there is no specific definition. For some path *P1* not defined at node *N*, there are two cases to consider: either *P1* is the extension of some path defined at *N* or it is not. The latter case is easiest – there is simply no definition for *P1* at *N* (hence *N* can be a **partial** function, as already noted above). But in the former case, where *P1* extends some *P2* which **is** defined at *N*, *P1* assumes a definition "by default". If *P2* is the only path defined at *N* which *P1* extends, then *P1* takes its definition from the definition of *P2*. If *P1* extends several paths defined at *N*, it takes its definition from the most specific (i.e., the longest) of the paths that it extends.

In the present example, this mode of default specification can be applied as follows. We have two statements at `Word1` which (after applying the abbreviation introduced above) both inherit from `VERB`:

```
Word1:
    <syn cat> == VERB
    <syn type> == VERB.
```

Because they have a common leading subpath `<syn>`, we can collapse them into a single statement about `<syn>` alone:

```
Word1:
    <syn> == VERB.
```

If this were the entire definition of `Word1`, the default mechanism would ensure that all extensions of `<syn>` (including the two that concern us here) would be given the same definition – inheritance from `VERB`. But in our example, of course, there are other

6 And hence also the extensional version, `Word1:<syn cat> = verb`

statements concerning `Word1`. If we add these back in, the complete definition looks like this:

```
    Word1:
        <syn> == VERB
        <syn form> == present participle
        <mor form> == love ing.
```

The paths <`syn type`> and <`syn cat`> (and also many others, such as <`syn cat foo`>, <`syn baz`>) obtain their definitions from <`syn`> using the default mechanism just introduced, and so inherit from `VERB`. But <`syn form`>, being explicitly defined, is exempt from this default behaviour, and so retains its value definition, `present participle`. And any extensions of <`syn form`> obtain their definitions from <`syn form`> rather than <`syn`> (since it is a more specific leading subpath), and so will have the value `present participle` also.

The net effect of this definition for `Word1` can be glossed as "`Word1` stipulates its morphological form to be `love ing` and inherits values for its syntactic features from `VERB`, except for <`syn form`> which is `present participle` ". More generally, this mechanism allows us to define nodes differentially: by inheritance from default specifications, augmented by any non-default settings associated with the node at hand. In fact, the `Word1` example can take this default inheritance one step further, by inheriting **everything** (not just <`syn`>) from `VERB`, except for the specifically mentioned values:

```
    Word1:
        <> == VERB
        <syn form> == present participle
        <mor form> == love ing.
```

Here the empty path <> is a leading subpath of every path, and so acts as a "catch all" – any path for which no more specific definition at `Word1` exists will inherit from `VERB`. Inheritance via the empty path is ubiquitous in real DATR lexicons but it should be remembered that the empty path has no special formal status in the language.

In this way `Word1` and `Word2` can both inherit their general verbal properties from `VERB`. But of course these two particular forms have more in common than simply being verbs – they are both instances of the **same** verb, *love*. By introducing an abstract `Love` lexeme, we can provide a site for properties shared by all forms of *love* (in this simple example, just its morphological root and the fact that it is a verb).

```
    VERB:
        <syn cat> == verb
        <syn type> == main.
    Love:
        <> == VERB
        <mor root> == love.
    Word1:
        <> == Love
        <syn form> == present participle
        <mor form> == <mor root> ing.
    Word2:
        <> == Love
        <syn form> == passive participle
        <mor form> == <mor root> ed.
```

So now `Word1` inherits from `Love` rather than `VERB` (but `Love` inherits from `VERB`, so the latter's definitions are still present at `Word1`). However, instead of explicitly including the atom `love` in the morphological form, the value definition includes the descriptor <`mor` `root>`. This descriptor is equivalent to `Word1:<mor root>` and, since `<mor root>` is not defined at `Word1`, the empty path definition applies, causing it to inherit from `Love:<mor root>`, and thereby return the expected value, `love`. Notice here that each element of a value can be defined entirely independently of the others; for `<mor form>` we now have an inheritance descriptor for the first element and a simple value for the second.

Our toy fragment is beginning to look somewhat more respectable: a single node for abstract verbs, a node for each abstract verb lexeme, and then individual nodes for each morphological form of each verb. But there is still more that can be done. Our focus on a single lexeme has meant that one class of redundancy has remained hidden. The line

```
        <mor form> == <mor root> ing
```

will occur in every present participle form of every verb. But it is a completely generic statement that can be applied to all English present participle verb forms. So can we not replace it with a single statement in the `VERB` node? Using the mechanisms we have seen so far, the answer is no. The statement would have to be (i), which is equivalent to (ii), whereas the effect we want is (iii):

```
(i)     VERB:<mor form> == <mor root> ing
(ii)    VERB:<mor form> == VERB:<mor root> ing
(iii)   VERB:<mor form> == Word1:<mor root> ing
```

Using (i) or (ii), we would end up with the same morphological root for every verb (or more likely no value at all, since it is hard to imagine what value `VERB:<mor root>` might plausibly be given), rather than a different one for each. And of course, we cannot simply use (iii) as it is, since that only applies to the particular word described by `Word1`, namely *loving*.

The problem is that the inheritance mechanism we have been using is **local**, in the sense that it can only be used to inherit either from a specifically named node (and/or path), or relative to the local context of the node (and/or path) at which it is defined. What we need is a way of specifying inheritance relative to the the **original** node/path specification whose value we are trying to determine, rather than the one we have reached by following inheritance links. We shall refer to this original specification as the **query** we are attempting to evaluate, and the node and path associated with this query as the **global** context[7]. Global inheritance, that is, inheritance relative to the global context, is indicated in DATR by using quoted (`"..."`) descriptors, and we can use it to extend our definition of `VERB` as follows:

```
    VERB:
        <syn cat> == verb
        <syn type> == main
        <mor form> == "<mor root>" ing.
```

Here we have added a definition for `<mor form>` which contains the **quoted** path `"<mor root>"`. Roughly speaking, this is to be interpreted as "inherit the value of `<mor root>` from the node originally queried". With this extra definition, we no longer need a `<mor form>` definition in `Word1`, so it just becomes:

```
    Word1:
        <> == Love
        <syn form> == present participle.
```

To see how this global inheritance works, consider evaluating the query `Word1:<mor form>`. Since `<mor form>` is not defined at `Word1`, it will inherit from `VERB` via `Love`.

7 Strictly speaking, the query node and path form just the **initial** global context, since as we shall see in Section 3.2.2 below, the global context can change during inheritance processing.

This specifies inheritance of <`mor root`> from the query node, which in this case is `Word1`. The path <`mor root`> is not defined at `Word1` but inherits the value `love` from `Love`. Finally, the definition of <`mor form`> at `VERB` adds an explicit `ing`, resulting in a value of `love ing` for `Word1:`<`mor form`>. However, had we begun evaluation at, say, a daughter of the lexeme `Eat`, we would have been directed from `VERB:`<`mor form`> back to the original daughter of `Eat` to determine its <`mor root`>, which would be inherited from `Eat` itself. So we would have ended up with the value `eat ing`.

The analysis is now almost the way we would like it to be. However, by moving <`mor form`> from `Word1` to `VERB`, we have introduced a new problem: we have frozen in the present participle as the (default) value of <`mor form`> for all verbs. Clearly, if we want to specify other forms at the same level of generality, then <`mor form`> is currently misnamed: it should be <`mor present participle`>, so that we can add <`mor past participle`>, <`mor present tense`>, etc. If we make this change, then the `VERB` node will look like this:

```
    VERB:
        <syn cat> == verb
        <syn type> == main
        <mor past> == "<mor root>" ed
        <mor passive> == "<mor past>"
        <mor present> == "<mor root>"
        <mor present participle> == "<mor root>" ing
        <mor present tense sing three> == "<mor root>" s.
```

In adding these new specifications, we have added a little extra structure as well. The passive form is asserted to be the same as the past form – the use of global inheritance here ensures that irregular or subregular past forms result in irregular or subregular passive forms, as we shall see shortly. The paths introduced for the present forms illustrate another use of default definition. We assume that the morphology of present tense forms is specified with paths of five attributes, the fourth specifying number, the fifth, person. Here we define default present morphology to be simply the root, and this generalises to all the longer forms, except the present participle and the third person singular.

So now for `Love`, the following extensional statements hold, inter alia:

```
    Love:
        <syn cat> = verb
        <syn type> = main
        <mor present tense sing one> = love
        <mor present tense sing two> = love
        <mor present tense sing three> = love s
        <mor present tense plur> = love
        <mor present participle> = love ing
        <mor past tense sing one> = love ed
        <mor past tense sing two> = love ed
        <mor past tense sing three> = love ed
        <mor past tense plur> = love ed
        <mor past participle> = love ed
        <mor passive participle> = love ed.
```

There remains one last problem in the definitions of `Word1` and `Word2`. The morphological form of `Word1` is now given by <`mor present participle`>. Similarly, `Word2`'s morphological form is given by <`mor passive participle`>. There is no longer a **unique** path representing morphological form. But this can be corrected by the addition of a single statement to `VERB`:

```
    VERB:
        <mor form> == "<mor "<syn form>">".
```

This statement employs a DATR construct, the **evaluable path**, which we have not encountered before. The right hand side consists of a (global) path specification, one of whose component attributes is itself a descriptor, to be evaluated before the outer path can be. The effect of the above statement is to say that `<mor form>` globally inherits from the path given by the atom `mor` followed by the global value of `<syn form>`. For `Word1`, `<syn form>` is `present participle`, so `<mor form>` inherits from `<mor present participle>`. But for `Word2`, `<mor form>` inherits from `<mor passive participle>`. Effectively, the `<syn form>` is being used as a parameter to control which specific form should be considered **the** morphological form. Evaluable paths may themselves be global (as in our example) or local and their evaluable components may also involve global or local reference.

Our analysis now looks like this:

```
    VERB:
        <syn cat> == verb
        <syn type> == main
        <mor form> == "<mor "<syn form>">"
        <mor past> == "<mor root>" ed
        <mor passive> == "<mor past>"
        <mor present> == "<mor root>"
        <mor present participle> == "<mor root>" ing
        <mor present tense sing three> == "<mor root>" s.
    Love:
        <> == VERB
        <mor root> == love.
    Word1:
        <> == Love
        <syn form> == present participle.
    Word2:
        <> == Love
        <syn form> == passive participle.
```

The entire analysis is somewhat larger than the original, but it encodes all the past and present tense forms as well as all three participial forms. More importantly, almost all the information is in the `VERB` node and is common to many verb lexemes[8]. Indeed, the other nodes are as small as they reasonably could be: `Love` simply states that it is a verb with morphological root `love` and `Word1` simply states that it is a present participle instance of `Love`.

Of course, `Love` is a completely regular verb. But DATR's capacity for definition by default allows subregular and irregular lexemes to be concisely represented also. As an example, consider the class of verbs which take *en* as their past participle ending: *hew, mow, saw, sew,* etc.[9] We can represent this subregularity with a new verbal node which defaults to `VERB`, but overrides just the past participle morphology:

8 Linguistically, the analysis is still not abstract enough since it fails to encode the morphotactic generalisation that, by default, an inflected English word consists of a root optionally followed by a suffix. Such generalisations are easy enough to state in DATR but would entail more elaboration of our running example than its expository purpose requires.

9 Our orthographic representations in this paper presuppose some basic "spelling rules", thus `love ed` is spelt *loved*, `love ing` is spelt *loving* and `mow en` is spelt *mown*. If we had chosen to represent roots and suffixes as letter sequences rather than as atoms then it would have been possible to implement the necessary spelling rules in a finite state transducer written in DATR itself. See, for example, that presented in Section 4.3, below.

```
    EN_VERB:
        <> == VERB
        <mor past participle> == "<mor root>" en.
```

Relevant individual verb lexemes then inherit from this node instead of directly from `VERB`:

```
    Mow:
        <> == EN_VERB
        <mor root> == mow.
    Sew:
        <> == EN_VERB
        <mor root> == sew.
```

As noted above, the passive forms of these subregular verbs will also now be correct, because of the use of a global cross-reference to the past participle form in the `VERB` node. So for example, the definition of the passive form of *sew* is:

```
    Word3:
        <> == Sew
        <syn form> == passive participle.
```

If we seek to establish the `<mor form>` of `Word3`, we are sent up the hierarchy of nodes, first to `Sew`, then to `EN_VERB`, and then to `VERB`. Here we encounter `"<mor "<syn form>">"` which resolves to `"<mor passive participle>"` in virtue of the embedded global reference to `<syn form>` at `Word3`. This means we now have to establish the value of `<mor passive participle>` at `Word3`. Again, we ascend the hierarchy to `VERB` and find ourselves referred to the global descriptor `"<mor past participle>"`. This takes us back to `Word3`, from where we again climb, first to `Sew`, then to `EN_VERB`. Here, `<mor past participle>` is given as the sequence `"<mor root>" en`. This leads us to look for the `<mor root>` of `Word3` which we find at `Sew` giving the result we seek:

```
    Word3:
        <mor form> = sew en.
```

Irregularity can be treated as just the limiting case of subregularity, so, for example, the morphology of `Do` can be specified as follows[10]:

```
    Do:
        <> == VERB
        <mor root> == do
        <mor past> == did
        <mor past participle> == done
        <mor present tense sing three> == does.
```

Likewise, the morphology of `Be` can be specified as follows[11]:

```
    Be:
        <> == EN_VERB
        <mor root> == be
        <mor present tense sing one> == am
        <mor present tense sing three> == is
```

10 Orthographically, the form `does` could simply be treated as regular (from `do s`). However, we have chosen to stipulate it here since, although the spelling appears regular, the phonology is not, so in a lexicon that defined phonological forms it would need to be stipulated.

11 In their default unification reconstruction of this DATR analysis of English verbal inflection, Bouma & Nerbonne (1994) invoke "a feature `-SG3` to cover all agreement values other than third person singular" in order "to avoid redundancy". But they do not explain how they would then account for the first person singular present tense form of *be* without reintroducing the redundancy that they are seeking to avoid. And the use of this purely morphological feature leads them to introduce a set of lexical rules in order to map the relevant information across from the (different) syntactic features.

```
    <mor present tense plur> == are
    <mor past tense sing one> == <mor past tense sing three>
    <mor past tense sing three> == was
    <mor past tense plur> == were.
```

In this section we have moved from simple attribute/value listings to a compact, generalisation-capturing representation for a fragment of English verbal morphology. In so doing, we have seen examples of most of the important ingredients of DATR: local and global descriptors, definition by default, and evaluable paths.

3. The DATR language

3.1 Syntax

A DATR description consists of a sequence of sentences corresponding semantically to a set of statements. Sentences are built up out of a small set of basic expression types, built up out of sequences of lexical tokens, which we take to be primitive.

In the previous section, we referred to individual lines in DATR definitions as **statements**. Syntactically however, a DATR description consists of a sequence of **sentences**, where each sentence starts with a node name and ends with a period, and contains one or more path equations relating to that node, each corresponding to a statement in DATR. This distinction between sentences and statements is primarily for notational convenience (it would be cumbersome to require repetition of the node name for each statement) and statements are the primary unit of specification in DATR. For the purposes of this section, where we need to be particularly clear about this distinction, we shall call a sentence containing just a single statement a **simple** sentence.

3.1.1 Lexical tokens. The syntax of DATR distinguishes four classes of lexical token: **nodes**, **atoms**, **variables** and reserved symbols. The complete list of reserved symbols is as follows:

```
    : " < > = == . ' % #
```

We have already seen the use of the first seven of these. Single quotes can be used to form atoms that would otherwise be ill-formed as such; `%` is used for end-of-line comments, following the Prolog convention; `#` is used to introduce declarations and other compiler directives[12].

The other classes, **nodes**, **atoms** and **variables**, must be distinct, and distinct from the reserved symbols, but are otherwise arbitrary[13]. For this discussion, we have already adopted the convention that both nodes and atoms are simple words, with nodes starting with uppercase letters. We extend this convention to variables, discussed more fully in Section 3.4 below, which we require to start with the character `$`. And we take white-space (spaces, newlines, tabs, etc.) to delimit lexical tokens but otherwise to be insignificant.

3.1.2 Right-hand-side expressions. The expressions that may appear as the right-hand-sides of DATR equations are sequences of zero[14] or more **descriptors**. Descriptors

12 Aside from their use in Section 3.4, we will completely ignore such directives in this paper.
13 Formally, we require them to be finite classes, but this is not of great significance here.
14 DATR makes a distinction between a path not having a value (i.e., being undefined) and a path having the **empty** sequence as a value:

```
    NUM:
        <two> ==
        <one> == one.
```

In this example, `NUM:<one>` has the value `one`, `NUM:<two>` has the empty sequence as its value, and `NUM:<three>` is simply undefined.

are defined recursively, and come in seven kinds. The simplest descriptor is just an atom or variable:

```
atom1
$var1
```

Then there are three kinds of **local inheritance descriptor**: a node, an (evaluable) path, and a node/path pair. Nodes are primitive tokens, paths are descriptor sequences (defined below) enclosed in angle brackets and node/path pairs consist of a node and a path separated by a colon:

```
Node1
<desc1 desc2 desc3 ...>
Node1:<desc1 desc2 desc3 ...>
```

Finally there are three kinds of **global inheritance descriptor**, which are quoted variants of the three local types just described:

```
"Node1"
"<desc1 desc2 desc3 ...>"
"Node1:<desc1 desc2 desc3 ...>"
```

A descriptor sequence is a (possibly empty) sequence of descriptors. The recursive definition of evaluable paths in terms of descriptor sequences allows arbitrarily complex expressions to be constructed, such as[15]:

```
"Node1:<"<atom1>" Node2:<atom2>>"
"<"<<Node1:<atom1 atom2> atom3>" Node2 "<atom4 atom5>" <> >">"
```

But the value sequences determined by such definitions are **flat**: they have no structure beyond the simple sequence and in particular do not reflect the structure of the descriptors that define them.

We shall sometimes refer to descriptor sequences containing only atoms as **simple** values, and similarly (unquoted) path expressions containing only atoms as simple paths.

3.1.3 Sentences. DATR sentences represent the statements which make up a description. As we have already seen, there are two basic statement types, extensional and definitional, and these correspond directly to simple extensional and definitional sentences, which are made up from the components introduced in the preceding section.

Simple extensional sentences take the form

```
Node:Path = Ext
```

where `Node` is a node, `Path` is a simple path, and `Ext` is a simple value. Extensional sentences derivable from the examples given in Section 2 include:

```
Do:<mor past participle> = done.
Mow:<mor past tense sing one> = mow ed.
Love:<mor present tense sing three> = love s.
```

Simple definitional sentences take the form

```
Node:Path == Def.
```

where `Node` and `Path` are as above and `Def` is an arbitrary descriptor sequence. Definitional sentences already seen in Section 2 include:

```
Do:<mor past> == did.
VERB:<mor form> == "<mor "<syn form>">".
EN_VERB:<mor past participle> == "<mor root>" en.
```

15 A descriptor containing an evaluable path may include nested descriptors which are either local or global. Our use of the local/global terminology always refers to the outermost descriptor of an expression.

Each of these sentences corresponds directly to a DATR statement. However we extend the notion of a sentence to include an abbreviatory convention for sets of statements relating to a single node. The following single sentence:

```
Node:
    Path1 == Def1
    Path2 == Def2
    ...
    PathN == DefN.
```

abbreviates (and is entirely equivalent to):

```
Node:Path1 == Def1.
Node:Path2 == Def2.
    ...
Node:PathN == DefN.
```

Extensional statements, and combinations of definitional and extensional statements, may be similarly abbreviated, and the examples used throughout this paper make extensive use of this convention. Such compound sentences correspond to a number of individual (and entirely independent) DATR statements.

Finally, it is worth reiterating that DATR descriptions correspond to **sets** of statements: the order of sentences, or of definitions within a compound sentence is immaterial to the relationships described.

3.2 Inheritance in DATR

DATR descriptions associate values with node/path pairs. This is achieved in one of three ways: a value is explicitly stated, or it is explicitly inherited, or it is implicitly specified (stated or inherited) via the default mechanism. We have already seen how values are explicitly stated; in this and the following subsections, we continue our exposition by providing an informal account of the semantics of specification via inheritance or by default. The present subsection is only concerned with explicit (i.e., non-default) inheritance. Section 3.3 deals with implicit specification via DATR's default mechanism.

3.2.1 Local inheritance. The simplest type of inheritance in DATR is the specification of a value by local inheritance. Such specifications may provide a new node, a new path, or a new node and path to inherit from. An example definition for the lexeme `Come` illustrates all three of these types:

```
Come:
    <> == VERB
    <mor root> == come
    <mor past> == came
    <mor past participle> == <mor root>
    <syn> == INTRANSITIVE:<>.
```

Here the empty path inherits from `VERB` so the value of `Come:<>` is equated to that of `VERB:<>`. And the past participle inherits from the root: `Come:<mor past participle>` is equated with `Come:<mor root>` (i.e., `come`). In both these inheritances, only one of the node or path was specified: the other was taken to be the same as that found on the left-hand-side of the statement (<> and `Come` respectively). The third type of local inheritance is illustrated by the final statement, in which both node and path are specified: the syntax of `Come` is equated with the empty path at `INTRANSITIVE`, an abstract node defining the syntax of intransitive verbs[16].

16 Bear in mind that the following are not synonymous

There is a natural procedural interpretation of this kind of inheritance, in which the value associated with the definitional expression is determined by "following" the inheritance specification and looking for the value at the new site. So given a DATR description (i.e., a set of definitional statements) and an initial node/path query, we look for the node and path as the left hand side of a definitional statement. If the definitional statement for this pair provides a local descriptor, then we follow it, by changing one or both of the node or path, and then repeat the process with the resulting node/path pair. We continue until some node/path pair specifies an explicit value. In the case of multiple expressions on the right hand side of a statement, we pursue each of them entirely independently of the others. This operation is local in the sense that each step is carried out without reference to any context wider than the immediate definitional statement at hand.

Declaratively speaking, local descriptors simply express equality constraints between definitional values for node/path pairs. The statement:

```
    Node1:Path1 == Node2:Path2.
```

can be read approximately as "if the value for `Node2:Path2` is defined, then the value of `Node1:Path1` is defined and equal to it". There are several points to notice here. First, if `Node2:Path2` is **not** defined, then `Node1:Path1` is unconstrained, so this is a weak directional equality constraint. However, in practice this has no useful consequences, due to interactions with the default mechanism – see Section 5.1 below. Second, "defined" here means "defined by a definitional statement", that is a "`==`" statement: local inheritance operates entirely with definitional statements, implicitly introducing new ones for `Node1:Path1` on the basis of those defined for `Node2:Path2`. Finally, as we shall discuss more fully in the next subsection, "value" here technically covers both simple values **and** global inheritance descriptors.

3.2.2 Global inheritance. Like local inheritance, global inheritance comes in three types: node, path, and node/path pair. However, when either the node or the path is omitted from a global inheritance descriptor, rather than using the node or path of the left-hand-side of the statement it is contained in (the **local** context of the definition), the values of a **global** context are used instead. This behaviour is perhaps also more easily introduced procedurally rather than declaratively. As we saw above, we can think of local inheritance in terms of following descriptors starting from the query. The local context is initially set to the node and path specified in the query, and when a local descriptor is encountered, any missing node or path components are filled in from the local context, and then control passes to the new context created (that is, we look at the definition associated with the new node/path pair). In doing this, the local context also changes to be the new context. Global inheritance operates in exactly the same way: the global context is initially set to the node and path specified in the query. It is **not** altered when local inheritance descriptors are followed (so it "remembers" where we started from), but when a global descriptor is encountered, it is the global context that is used to fill in any missing node or path components in the descriptor, and hence to decide where to pass control to. In addition, **both** global and local contexts are updated to the new settings. So global inheritance can be seen as essentially the same mechanism as local inheritance, but layered on top of it – following global links alters the local context too, but not vice versa.

```
        Come:<syn> == INTRANSITIVE:<>.
        Come:<syn> == INTRANSITIVE.
```
since the latter is equivalent to
```
        Come:<syn> == INTRANSITIVE:<syn>.
```

For example, when a global path is specified, it effectively "returns control" to the current global node (often the original query node) but with the newly given path. Thus in Section 2, above, we saw that the node `VERB` defines the default morphology of present forms using global inheritance from the path for the morphological root:

```
    VERB:<mor present> == "<mor root>".
```

The node from which inheritance occurs is that stored in the global context. So a query of `Love:<mor present>` will result in inheritance from `Love:<mor root>` (via `VERB:<mor present>`), while a query of `Do:<mor present>` will inherit from `Do:<mor root>`.

Similarly, a quoted **node** form accesses the globally stored **path** value, as in the following example:

```
    Declension1:
        <vocative> == -a
        <accusative> == -am.
    Declension2:
        <vocative> == "Declension1"
        <accusative> == -um.
    Declension3:
        <vocative> == -e
        <accusative> == Declension2:<vocative>.
```

Here, the value of `Declension3:<accusative>` inherits from `Declension2:<vocative>` and then from `Declension1:< accusative>`, using the global path (in this case the query path), rather than the local path (`<vocative>`) to fill out the specification. So the resulting value is `-am` and not `-a` as it would have been if the descriptor in `Declension2` had been local rather than global.

We observed above that when inheritance through a global descriptor occurs, the global context is altered to reflect the new node/path pair. Thus after `Love:<mor present>` has inherited through `"VERB:<mor root>"`, the global path will be `<mor root>` rather than `<mor present>`. When we consider quoted node/path pairs, it turns out that this is the only property that makes them useful. Since a quoted node/path pair completely respecifies both node and path, its immediate inheritance characteristics are the same as the unquoted node/path pair. However, because it also alters the global context, its effect on any **subsequent** global descriptors (in the evaluation of the same query) will be different:

```
    Declension1:
        <vocative> == "<nominative>"
        <nominative> == -a.
    Declension2:
        <vocative> == Declension1
        <nominative> == -u.
    Declension3:
        <nominative> == -i
        <accusative> == "Declension2:<vocative>".
```

In this example, the value of `Declension3:<accusative>` inherits from `Declension2:<vocative>` and then from `Declension1:<vocative>` and then from `Declension2:<nominative>` (because the global node has changed from `Declension3` to `Declension2`) giving a value of `-u` and not `-i` as it would have been if the descriptor in `Declension3` had been local rather than global.

There are a number of ways of understanding this global inheritance mechanism. The description we have given above amounts to a "global memory" model, in which a DATR query evaluator is a machine equipped with two memories: one containing the current local node and path, and another containing the current global node and path. Both are initialised to the query node and path, and the machine operates by repeatedly examining the definition associated with the current local settings. Local descriptors alter just the local memory, while global descriptors alter both the local and global settings.

This model is the basis of at least one implementation of DATR but it is not, of course, declarative. Nevertheless, the notion of global inheritance does have a declarative reading, very similar to local inheritance but, as we have already suggested, layered on top of it. Recall that local inheritance establishes a network of weak equality relationships among node/path pairs, and these equalities are used to distribute values across this network. As we mentioned above, formally speaking the local inheritance network controls the distribution not only of simple values, but of global descriptors as well. That is, to local inheritance, values and global descriptors are one and the same, and are inherited through the network. Indeed, this is the intuitive warrant for the use of the quote notation: the quotes turn an inheritance descriptor into a (kind of) value. Consequently, global descriptors are also distributed through the local inheritance network, and so are implicitly present at many node/path pairs in addition to those they are explicitly defined for. In fact, a global descriptor is implicitly present at every node/path pair which could ever occur as the global context for evaluation of the descriptor at its original explicitly defined location. This means that once distributed like this, the global descriptors form a network of weak equality relationships just like the local descriptors, and distribute the simple values (alone) in just the same way.

To see this interpretation in action, we consider an alternative analysis of the past participle form of `Come`. The essential elements of the analysis are as follows:

```
BARE_VERB:
    <mor past participle> == "<mor root>".
Come:
    <mor root> == come
    <mor past participle> == BARE_VERB.
```

Local inheritance from `BARE_VERB` to `Come` implicitly defines the following statement (in addition to the above):

```
Come:
    <mor past participle> == "<mor root>".
```

Because we have now brought the global inheritance descriptor to the node corresponding to the global context for its interpretation, **global** inheritance can now operate entirely **locally** – the required global node is the local node, `Come`, producing the desired result:

```
Come:
    <mor past participle> = come.
```

Notice that, in this last example, the final statement was extensional, not definitional. So far in this paper we have almost entirely ignored the distinction we established between definitional and extensional statements, but with this declarative reading of global inheritance we can do so no longer. Local inheritance uses definitional inheritance statements to distribute simple values and global descriptors. The simple-valued definitional statements thereby defined map directly to extensional statements, and global inheritance uses the global inheritance statements (now distributed), to further distribute these extensional statements about simple values. The statements have to be of a formally distinct type, to prevent local inheritance descriptors from distributing them still further. In practice, however, we need not be too concerned about the distinction: descriptions are written as definitional statements, queries are read off as extensional statements[17].

17 However, in principle, there is nothing to stop an extensional statement being specified as part of a

The declarative interpretation of global inheritance suggests an alternative procedural characterisation to the one already discussed, which we outline as follows. Starting from a query, local descriptors alone are used to determine either a value or a global descriptor associated with the queried node/path pair. If the result is a global descriptor, this is used to construct a new query, which is evaluated in the same way. The process repeats until a value is returned. The difference between this and the earlier model is really just one of perspective. When a global descriptor is encountered one can either bring the global context to the current evaluation context (first model), or take the new descriptor back to the global context and continue from there (second model). The significance of the latter approach is that it reduces both kinds of inheritance to a single basic operation which has a straightforward declarative interpretation. Thus we see that DATR contains two instances of essentially the same declarative inheritance mechanism. The first, local inheritance, is always specified explicitly, while the second, global inheritance, is specified implicitly in terms of the first.

Extending these inheritance mechanisms to the more complex DATR expressions is straightforward. Descriptors nested within definitional expressions are treated independently — as though each was the entire value definition rather than just an item in a sequence. In particular, global descriptors which alter the global context in one nested definition have no effect on any others. Each descriptor in a definitional sequence or evaluable path is evaluated from the **same** global state. In the case of global evaluable paths, once the subexpressions have been evaluated, the expression containing the resultant path is also evaluated from the same global state.

3.3 Definition by default

The other major component of DATR is definition by default. This mechanism allows a DATR definitional statement to be applicable not only for the path specified in its left-hand-side, but also for any rightward extension of that path for which no more specific definitional statement exists. In effect, this "fills in the gaps" between paths which are defined at a node, on the basis that an undefined path takes its definition from the path which best approximates it without being **more** specific[18]. Of course, to be effective, this "filling in" has to take place **before** the operation of the inheritance mechanisms described in the previous section.

Consider for example, the definition of `Do` we gave above.

```
Do:
    <> == VERB
    <mor root> == do
    <mor past> == did
    <mor past participle> == done
    <mor present tense sing three> == does.
```

Filling in the gaps between these definitions, we can see that many paths will be implicitly defined only by the empty path specification. Examples include:

```
Do:
    <mor> == VERB
    <syn> == VERB
    <mor present> == VERB
    <syn cat> == VERB
```

DATR description directly. Such a statement would respect global inheritance but not local inheritance and might be useful to achieve some exotic effect.

18 For formal discussion of the semantics of the DATR default mechanism, see Keller (1995).

```
        <syn type> == VERB
        <mor present tense sing one> == VERB.
```

If there had been no definition for <>, then none of these example paths would have been defined at all, since there would have been no leading subpath with a definition. Note how `<mor>` itself takes its definition from <>, since all the explicitly defined `<mor` ...> specifications have at least one further attribute.

The definition for `<mor past>` overrides default definition from <> and in turn provides a definition for longer paths. However, `<mor past participle>` blocks default definition from `<mor past>`. Thus the following arise[19]:

```
    Do:
        <mor past tense> == did
        <mor past tense plur> == did
        <mor past tense sing three> == did
        <mor past participle plur> == done
        <mor past participle sing one> == done.
```

Similarly all the `<mor present>` forms inherit from `VERB` except for the explicitly cited `<mor present tense sing three>`.

Definition by default introduces new DATR sentences each of whose left-hand-side paths is an extension of the left-hand-side paths of some explicit sentence. This path extension carries over to any paths occurring on the **right-hand-side** as well. So for example, the sentence:

```
    VERB:
        <mor present tense> == "<mor root>"
        <mor form> == <mor "<syn form>">.
```

gives rise to the following, inter alia:

```
    VERB:
        <mor present tense sing> == "<mor root sing>"
        <mor present tense plur> == "<mor root plur>"
        <mor form present> == <mor "<syn form present>" present>
        <mor form passive> == <mor "<syn form passive>" passive>.
```

This extension occurs for all paths in the right-hand-side, whether they are quoted or unquoted and/or nested in descriptor sequences or evaluable paths.

The intent of this path extension is to allow descriptors to provide not simply a single definition for a path but a whole set of definitions for extensions to that path, without losing path information. In some cases this can lead to gratuitous extensions to paths – path attributes specifying detail beyond any of the specifications in the overall description. However, this does not generally cause problems since such gratuitously detailed paths, being unspecified, will always take their value from the most specific path that is specified (effectively, gratuitous detail is ignored)[20]. Indeed, DATR's approach to default information always implies an infinite number of unwritten DATR statements, with paths of arbitrary length.

3.4 Abbreviatory variables

The default mechanism of DATR provides for generalisation across sets of atoms by means of path extension and is the preferred mechanism to use in the majority of cases. However,

19 The past participle extensions here are purely for the sake of the formal example – they have no role to play in the morphological description of English (but cf. French where past participles inflect for gender and number).

20 Thus, for example, the path `<mor plur acc>` is a gratuitous extension of the path `<mor plur>` for English common nouns since the latter are not differentiated for case.

when one wants to transduce atoms in the path domain to atoms in the value domain (see Section 4.3, below), it is extremely convenient to be able to make use of abbreviatory variables over finite sets of atoms. This is achieved by declaring DATR variables whose use constitutes a kind of macro: they can always be eliminated by replacing the equations in which they occur with larger sets of equations that spell out each value of the variables. Conventionally, variable names begin with the $ character and are declared in one of the following three ways:

```
    # vars $Var1: Range1 Range2 ... .
    # vars $Var2: Range1 Range2 ... - RangeA RangeB ... .
    # vars $Var3.
```

Here, the first case declares a variable `$Var1` that ranges over the values `Range1, Range2` ..., where each `RangeN` is either an atom or a variable name, the second case declares `$Var2` to range over the same range, but **excluding** values in `RangeA RangeB` ..., and the third declares `$Var3` to range over the full (finite) set of atoms in the language[21]. For example:

```
    # vars $letters: a b c d e f g h i j k l m n o p q r s t u v w x y z.
    # vars $vowels: a e i o u.
    # vars $consonants: $letters - $vowels.
    # vars $not_z: $letters - z.

    # vars $odd: 1 3 5 7 9.
    # vars $even: 0 2 6 4 8.
    # vars $digit: $odd $even.
```

Caution has to be exercised in the use of DATR variables for two reasons. One is that their use makes it hard to spot multiple confilcting definitions:

```
    # vars $vowel: a e i o u.
    DIPTHONG:
        <e> == e i <>
        <$vowel> == $vowel e <>.
```

Here, <e> appears on the left hand side of two conflicting definitions. Exactly what happens to such an improper description in practice depends on the implementation and usages of this kind can be the source of hard to locate bugs (See also Section 5.1, below.).

The other reason is that one can fall into the trap of using variables to express generalisations that would be better expressed using the path extension mechanism. Here is a very blatant example:

```
    # vars $number: singular plural.
    NOUNX:
        <third $number> == <second $number>.
```

This would almost certainly be better expressed as:

```
    NOUNX:
        <third> == <second>.
```

The following example is a variant on the same theme:

```
    # vars $number: singular plural.
    NOUNY:
        <$number third> == <$number second>.
```

which suggests, not a real need for the use of DATR variables, but rather an inappropriate choice of attribute order in the design of the description.

21 **Undeclared** variables are similarly assumed to range over the full set of atoms. Some implementations may also include implicit definitions of more restricted variables, such as `$integer`.

4. DATR techniques

The DATR fragments introduced above illustrate the basic descriptive resources provided by the language. We now present some further examples, showing how these basic components combine to provide a powerful representation tool.

4.1 Case constructs and parameters

Evaluable paths allow the value of one path to be determined by the value of another. More generally, the values of an arbitrary number of descriptors can be invoked as parameters in an evaluable path and thus determine the value of a particular node/path pair. The familiar **case** construct of procedural programming languages is readily implemented, as the following example describing English plural suffixes shows:

```
    NOUN:
        <plural> == <case of "<origin>">

        <case of latin masculine> == -i
        <case of latin neuter>    == -a
        <case of>                 == -s

        <origin> == norman.

    Cat:
        <> == NOUN.
    Datum:
        <> == NOUN
        <origin> == latin neuter.
    Alumnus:
        <> == NOUN
        <origin> == latin masculine.
```

Here the value of the <`origin`> attribute of a noun (denoting its etymological source) is used to determine the value of its <`plural`> suffix. Thus we can derive the following extensional statements:

```
    Cat:
        <plural> = -s.
    Datum:
        <plural> = -a.
    Alumnus:
        <plural> = -i.
```

We do not need to invoke an attribute called `case` to get this technique to work. For example, in Section 2, we gave the following definition of <`mor form`> in terms of <`syn form`>:

```
    VERB:
        <mor form> == <mor "<syn form>">.
```

Here the feature <`syn form`> returns a value (such as `passive participle` or `present tense sing three`) which becomes part of the path through which <`mor form`> inherits. This means that nodes for surface word forms need only state their parent lexeme and <`syn form`> feature in order for their <`mor form`> to be fully described[22]. So, as

22 More generally, evaluable paths provide "structured inheritance" in the sense of Daelemans & De Smedt (1994, 161-168).

we saw in Section 2 above, the passive participle form of *sew* is fully described by the node definition for `Word3`.

```
Word3:
    <> == Sew
    <syn form> == passive participle.
```

For finite forms, we could use a similar technique. From this,

```
Word4:
    <> == Sew
    <syn form> == present sing third.
```

we would want to be able to infer this:

```
Word4:
    <mor form> = sew s
```

However, the components of `<syn form>`, `present`, `sing`, `third` are themselves values of features we probably want to represent independently. One way to achieve this is to define a value for `<syn form>` which is itself parameterised from the values of these other features. And the appropriate place to do this is in the `VERB` node, thus:

```
VERB:
    <syn form> == "<syn tense>" "<syn number>" "<syn person>".
```

This says that the default value for the syntactic form of a verb is a finite form, but exactly which finite form depends on the settings of three other paths, `<syn tense>`, `<syn number>` and `<syn person>`. So now we can express `Word4` as:

```
Word4:
    <> == Sew
    <syn tense> == present
    <syn number> == sing
    <syn person> == third.
```

This approach has the advantage that the attribute ordering used in the `<mor...>` paths is handled internally: the leaf nodes need not know or care about it[23].

4.2 Boolean logic

We can, if we wish, use parameters in evaluable paths that resolve to `true` or `false`. We can then define standard truth tables over DATR paths:

```
Boolean:
    <> == false
    <or> == true
    <if> == true
    <not false> == true
    <and true true> == true
    <if true false> == false
    <or false false> == false.
```

This node defines the standard truth tables for all the familiar operators and connectives of the propositional calculus expressed in Polish rather than infix order[24]. Notice, in particular, how the DATR default mechanism completes most of the truth table rows without explicit listing. The definability of the propositional calculus may appear, at first sight, to be a curiosity, one which has no relevance to real-life lexical representation. But

23 `Word3` remains unchanged, overriding the definition of `<syn form>` and so not requiring these additional features to be defined at all.

24 We can, of course, use the same technique to define many-valued logics if we wish.

that is not so. Consider a hypothetical language in which personal proper names have one of two genders, masculine or feminine. Instead of the gender being wholly determined by the sex of the referent, the gender is determined partly by sex and partly by the phonology. Examples of this general type are quite common in the world's languages[25]. In our hypothetical example, the proper name will have feminine gender either if it ends in a consonant and denotes a female or if it ends in a stop consonant but does not denote a female. We can encode this situation in DATR as follows[26]:

```
Personal_name:
    <> == Boolean
    <ends_in_consonant> == "<ends_in_stop>"
    <gender_is_feminine> ==
              <or <and "<female_referent>" "<ends_in_consonant>">
                  <and <not "<female_referent>"> "<ends_in_stop>">>.
```

We can then list some example lexical entries for personal proper names[27]:

```
Taruz:
    <> == Personal_name
    <female_referent> == true
    <ends_in_consonant> == true.
Turat:
    <> == Personal_name
    <female_referent> == true
    <ends_in_stop> == true.
Tarud:
    <> == Personal_name
    <ends_in_stop> == true.
Turas:
    <> == Personal_name
    <ends_in_consonant> == true.
```

Note that both `Turas` and `Tarud` turn out not to denote females, given the general `false` default in `Boolean`[28]. The genders of all four names can now be obtained as theorems:

```
Taruz: <gender_is_feminine> = true.
Turat: <gender_is_feminine> = true.
Tarud: <gender_is_feminine> = true.
Turas: <gender_is_feminine> = false.
```

25 For example, Fraser & Corbett (1995) use DATR to capture a range of phonology/morphology/semantics interdependencies in Russian. And Brown & Hippisley (1994) do the same for a Russian segmental phonology/prosody/morphology interdependency. But one can find such interdependencies in English also: see Ostler & Atkins (1992: 96-98).

26 Note that complex expressions require path embedding. Thus, for example, the well-formed negation of a conditional is `<not <if .. ..>>` rather than `<not if .. ..>`.

27 For the sake of simplicity, we have assumed that the truth values of `<ends_in_consonant>` and `<ends_in_stop>` are just stipulated in the entries, and indeed the second definition in `Personal_name` means that `<ends_in_stop>` implies `<ends_in_consonant>`. But if the entries represented the phonology of the words in DATR also, then these predicates could be defined on the basis of the feature composition of the stem-final segment. As a number of researchers have shown, the highly defaulty character of lexical phonology and morphophonology makes DATR a very suitable medium of representation (Bleiching 1992, 1994; Cahill 1993b; Gibbon 1990, 1992; Gibbon & Bleiching 1991; Reinhard 1990; Reinhard & Gibbon 1991).

28 It is straightforward to add extra DATR code so as to derive `<gender>` = `feminine` when `<gender_is_feminine>` is `true` and `<gender>` = `masculine` when `<gender_is_feminine>` is `false`, or conversely.

4.3 Finite state transduction

Perhaps surprisingly, DATR turns out to be an excellent language for defining finite state transducers (FSTs)[29] A path can be used as the input tape and a value as the output tape (recall that the DATR default mechanism means that extensions to left-hand-side paths are automatically carried over as extensions to right-hand-side paths, as discussed in Section 3.3, above). Nodes can be used for states, or else states can be encoded in attributes that are prefixed to the current input path. Here, for example, is a very simple DATR FST that will transduce a path such as `<subj 1 sg futr obj 2 sg like>` into the value `ni ta ku penda` (Swahili for *I will like you*)[30]:

```
S1:
    <subj 1 sg> == ni S2:<>
    <subj 2 sg> == u  S2:<>
    <subj 3 sg> == a  S2:<>
    <subj 1 pl> == tu S2:<>
    <subj 2 pl> == m  S2:<>
    <subj 3 pl> == wa S2:<>.
S2:
    <past> == li S3:<>
    <futr> == ta S3:<>.
S3:
    <obj 1 sg> == ni S4:<>
    <obj 2 sg> == ku S4:<>
    <obj 3 sg> == m  S4:<>
    <obj 1 pl> == tu S4:<>
    <obj 2 pl> == wa S4:<>
    <obj 3 pl> == wa S4:<>.
S4:
    <like> == penda.
```

Although the example is trivial, the technique is both powerful and useful. Gibbon (1990), for example, has made notably effective use of it in his treatments of African tone systems[31]. Much of the computational phonology and morphology of the last decade and a half has depended on FSTs (Kaplan & Kay 1994). Potential lexical applications come readily to mind – for example, the orthographic spelling rules for English suffixation (cf. *sky/skies*). We give below a small fragment of such an FST in which + is used as a morpheme boundary marker. Note the role of DATR variables in giving concise expression to the rules:

```
# vars $abc: a b c d e f g h i j k l m n o p q r s t u v w x y z.
# vars $vow: a e i o u.
SPELL:
    <> ==
    <+> == <>
    <$abc> == $abc <>
    <e + $vow> == $vow <>.
```

These axioms then give rise to theorems such as these:

```
SPELL:
```

29 Cf. Krieger et al. (1993), who reconstruct finite state automata in a feature description language.
30 For clarity, this FST does not exploit default inheritance to capture the 50% overlap between the subject and object pronoun paradigms. See Gazdar (1992) for a version that does.
31 And see McFetridge & Villavicencio (1995) for a less exotic application.

```
    <l o v e>           = l o v e
    <l o v e + s>       = l o v e s
    <l o v e + e d>     = l o v e d
    <l o v e + e r>     = l o v e r
    <l o v e + l y>     = l o v e l y
    <l o v e + i n g>   = l o v i n g
    <l o v e + a b l e> = l o v a b l e.
```

4.4 Representing lists

DATR's foundation in path/value specifications means that many of the representational idioms of unification formalisms transfer fairly directly. A good example is the use of `first` and `rest` attributes to represent list-structured features, such as syntactic arguments and subcategorised complements. The following definitions could be used to extend our verb fragment by introducing the path <`syn args`>, which determines a list of syntactic argument specifications.

```
NIL:
    <> == nil
    <rest> == UNDEF
    <first> == UNDEF.
VERB:
    <syn cat> == verb
    <syn args first syn cat> == np
    <syn args first syn case> == nominative
    <syn args rest> == NIL:<>.
```

Here extensions of <`syn args first`> specify properties of the first syntactic argument, while extensions of <`syn args rest`> specify the others (as a first/rest list). `UNDEF` is the name of a node that is not defined in the fragment, thus ensuring that <`syn args rest first`>, <`syn args rest rest`>, and so forth are all undefined. The fragment above provides a default specification for <`syn args`> for verbs consisting of just one argument, the subject NP. Subclasses of verb may, of course, override any part of this default; for instance, transitive verbs add a second syntactic argument for their direct object:

```
TR_VERB:
    <> == VERB
    <syn args rest first syn cat> == np
    <syn args rest first syn case> == accusative
    <syn args rest rest> == NIL:<>.
```

The description can be improved by using a separate node, `NP_ARG`, to represent the (default) properties of noun-phrase arguments:

```
NP_ARG:
    <first syn cat> == np
    <first syn case> == accusative
    <rest> == NIL:<>.
VERB:
    <syn cat> == v
    <syn args> == NP_ARG:<>
    <syn args first syn case> == nominative.
TR_VERB:
    <> == VERB
    <syn args rest> == NP_ARG:<>.
```

TR_VERB accepts the NP_ARG default unconditionally for the direct object argument, but VERB overrides the default case for its subject argument. The effect of the empty path (<>) specification in the NP_ARG inheritances is to "strip off" the leading subpath from the path whose value is inherited. The default mechanism adds the same path extension to both sides, giving rise to statements such as the following:

```
    VERB:<syn args first syn cat> == NP_ARG:<first syn cat>.
    TR_VERB:<syn args rest first syn cat> == NP_ARG:<first syn cat>.
    TR_VERB:<syn args rest first syn case> == NP_ARG:<first syn case>.
```

Three element argument lists, such as that needed for ditransitive verbs, are constructed in the obvious way (where PP_ARG is assumed to be like NP_ARG but for prepositional-phrase complements):

```
    DI_VERB:
        <> == TR_VERB
        <syn args rest rest> == PP_ARG:<>.
```

4.5 Lexical rules

A lexical representation language needs to be able to express the relations that are now widely thought to be in the domain of lexical rules[32]. Canonically, such rules deal with the phenomena that used to be described by the "cyclic rules" of late 1960s transformational grammar. 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[33]. 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. The combination of evaluable paths with a standard encoding of argument lists make it rather easy to define lexical rules in DATR[34].

Here, by way of illustration, is a partial analysis of verbs that implements a lexical rule for syntax of the (agentless) passive construction[35]:

```
    VERB:
        <mor past> == "<mor root>" ed
        <mor form> == "<mor "<syn form>">"
        <syn cat> == verb
        <syn subcat> == "<syn args>"
        <syn args> == NP_ARG:<>
        <syn args first syn case> == nominative.
    PASSIVE_VERB:
        <> == VERB
        <mor passive> == "<mor past>"
        <syn subcat rest> == "<syn args rest rest>".
    TR_VERB:
```

32 See Carpenter (1991; 1992) and Ritchie et al. (1992, 93-111) for thorough discussion and exemplification of lexical rules in several different grammatical frameworks. More generally, Briscoe & Copestake (1991) and Copestake & Briscoe (1992) argue that, in the context of a default inheritance lexicon, the very same lexical rule mechanism can be invoked for both sense extensions and morphological processes.

33 Radically lexicalist frameworks, which lack any construction-specific grammatical rules outside the lexicon, do not restrict the use of lexical rules to "cyclic" phenomena. Thus, for example, Evans et al. (1995) report the use of DATR to formulate a lexical rule for *wh*-questions in LTAG, inter alia.

34 Evaluable paths are not essential in this domain: thus Kilgarriff (1993) does not employ them in his DATR analysis of verbal alternations in the context of an HPSG lexicon, although he does use the standard encoding of argument lists.

35 Since our purpose here is expository, we have deliberately kept the analysis to minimum. Dealing with the semantics of passive, for example, involves more of the same rather than any issue of principle.

```
        <syn args rest> == NP_ARG:<>
        <> == <<mood "<syn form>">>
        <mood> == active
        <mood passive> == passive
        <active> == VERB:<>
        <passive> == PASSIVE_VERB:<>.
```

This example introduces several new techniques. Firstly, in `TR_VERB` we have a double parametrisation on <`syn form`>: the value of <`syn form`> is evaluated and used to create a <`mood`> path. The value returned by this path is then used to route the inheritance. This allows us to employ the default mechanism to make the default mood active (for arbitrary <`syn form`> values other than those that begin with the atom `passive`), and thus just pick out <`syn form`> passive (and its extensions) for verbs in the passive mood. Secondly, <`active`> and <`passive`> path prefixes are provided for the explicit purpose of controlling the inheritance route. Thirdly, the example presupposes a distinction between the syntactic arguments list (<`syn args`>) associated with a lexeme and the subcategorisation frame list (<`syn subcat`>) associated with a particular syntactic form of a lexeme. If the mood of the form is active (and the `TR_VERB` node says that anything that is not passive is active), then the subcategorisation frame is the same as the argument list. But if the mood of the form is passive, then the part of the subcategorisation frame that deals with objects and complements is stripped of its first item – i.e., its direct object. By default, this dependency of subcategorisation frame on mood will be inherited by all the descendants of `TR_VERB` whether these be instances of simple transitive verb lexemes or nodes defining specific types of transitive verbs (ditransitives, object-plus-infinitive verbs, *bet*-class verbs, etc.) and their descendants. Thus, if we assume, for example, that the lexeme `Donate` is an instance of `DI_VERB` as defined above, and that `Word5` and `Word6` are inflected tokens of `Donate`, then we will be able to derive the following theorems:

```
    Word5:
        <mor form> = donate ed
        <syn form> = past tense
        <syn subcat first syn cat> = np
        <syn subcat first syn case> = nominative
        <syn subcat rest first syn cat> = np
        <syn subcat rest first syn case> = accusative
        <syn subcat rest rest first syn cat> = pp
        <syn subcat rest rest first syn pform> = to
        <syn subcat rest rest rest> = nil.
    Word6:
        <mor form> = donate ed
        <syn form> = passive participle
        <syn subcat first syn cat> = np
        <syn subcat first syn case> = nominative
        <syn subcat rest first syn cat> = pp
        <syn subcat rest first syn pform> = to
        <syn subcat rest rest> = nil.
```

Finally, notice that the equation that specifies passive morphology appears on the `PASSIVE_VERB` node. This ensures that passive morphology is undefined for verbs which are not syntactically passive.

The techniques used in this rather simple treatment of passive can be readily adapted for use in encoding other lexical rules and for grammatical frameworks other than that implicit in the PATRish syntax we have adopted in our example. Thus, as noted above, Evans et al. (1995) formulate various lexical rules for LTAG. They can also be readily adapted for use in the semantic domain and used, for example, to implement the distinction between **fixed** and **projective** inheritance of lexical semantic information proposed by Pustejovsky (1991,433-437).

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 another, that lexical classes do[36]. Thus 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 parameterisation of the components, lexical rules can be reified in a language like DATR, allowing one to inherit from another.

4.6 Representing ambiguity and alternation

DATR is a language that allows the lexicon writer to define sets of partial functions from sequences of atoms to sequences of atoms. That is actually all that it allows the lexicon writer to do. Because DATR deals in functions it does not embody any notion of disjunction or any possibility of multiple values being associated with a single node/path pair. It might seem, at first glance, as if such a language would be quite inappropriate to a domain such as the lexicon where ambiguities are common. In practice, however, this turns out not to be the case. Consider the homonymy of *bank*:

```
Bank1:
    <> == NOUN
    <mor root> == bank
    <sem gloss> == side of river.
Bank2:
    <> == NOUN
    <mor root> == bank
    <sem gloss> == financial institution.
```

This is simply the traditional analysis of homonymy, encoded in DATR: there are two entirely distinct lexemes with unrelated meanings that happen both to be nouns and to have indistinguishable morphological roots.

Or consider the polysemy of *cherry*[37]:

```
Cherry:
    <> == NOUN
    <mor root> == cherry
    <sem gloss 1> == sweet red berry with pip
    <sem gloss 2> == tree bearing <sem gloss 1>
    <sem gloss 3> == wood from <sem gloss 2>.
```

Again, this is a rather traditional analysis. There are (at least) three distinct but related senses[38]. They are not freely interchangeable alternative values for a single attribute or

36 Cf. Krieger (1994, 279) who notes some other advantages.

37 The example is due to Kilgarriff (1995) who shows that the kind of polysemy exhibited by *cherry* applies generally to fruit trees and can thus be specified at a higher node in the lexical network, removing the need for stipulation (as in our example) at the `Cherry` node, the `Apple` node, and so on. Kilgarriff & Gazdar (1995) also present an extended example showing how DATR can be used to encode the regular and subregular polysemy associated with the crop, fibre, yarn, fabric and garment senses of words like *cotton* and *silk*. See also Copestake & Briscoe (1995) for related work on regular and subregular polysemy.

38 For perspicuity, we provide these in DATR-augmented English here. But in a serious treatment they could just as well be given in a DATR-encoding of the lambda calculus, say (as used in Cahill & Evans 1990, for example).

path. Instead, DATR allows their relatedness of meaning to be captured by using the definition of one in the definition of another.

A very few words in English have alternative morphological forms for the same syntactic specification. An example noted by Fraser & Hudson (1990, 62) is the plural of *hoof* which, for many English speakers, can appear as both *hoofs* and *hooves*[39]. DATR does not permit a theorem set such as the following to be derived from a consistent description:

```
Word7:
    <syn number> = plural
    <mor form> = hoof s
    <mor form> = hoove s.
```

But it is quite straightforward to define a description that will lead to the following theorem set:

```
Word7:
    <syn number> = plural
    <mor form> = hoof s
    <mor form alternant> = hoove s.
```

Or something like this:

```
Word7:
    <syn number> = plural
    <mor forms> = hoof s | hoove s .
```

Or this:

```
Word7:
    <syn number> = plural
    <mor forms> = { hoof s , hoove s }.
```

Of course, as far as DATR is concerned `{ hoof s , hoove s }` is just a sequence of seven atoms. It is up to some component external to DATR which makes use of such complex values to interpret it as a two member set of alternative forms. Likewise, if we have some good reason for wanting to put together the various senses of *cherry* into a value returned by a single path, then we can write something like this:

```
Cherry:
    ...
    <sem glosses> == { <sem gloss 1> , <sem gloss 2> , <sem gloss 3> }.
```

which will then provide this theorem:

```
Cherry:
    <sem glosses> = { sweet red berry with pip ,
                      tree bearing sweet red berry with pip ,
                      wood from tree bearing sweet red berry with pip }.
```

Also relevant here are the various techniques for reducing lexical disjunction discussed in Pulman (forthcoming).

4.7 Encoding DAGs

As a feature-based formalism with a syntax modelled on PATR, it would be reasonable to expect that DATR can be used to describe directed acyclic graphs (DAGs) in a PATR-like fashion. Consider an example such as the following:

```
DAG1:
    <vp agr> == <v agr>
```

39 See also the *dreamt/dreamed* verb class discussed by Russell et al. (1992, 330-331).

```
        <v agr per> == 3
        <vp agr gen> == masc.
```

This looks like simple reentrancy from which we would expect to be able to infer:

```
    DAG1:
        <vp agr per> = 3.
```

And, indeed, this turns out to be valid. But matters are not as simple as the example makes them appear: if `DAG1` was really the DAG it purports to be, then we would also expect to be able to infer:

```
    DAG1:
        <v agr gen> = masc.
```

But this is not valid, in fact <`v agr gen`> is undefined. It might be tempting to conclude from this that the equality operator in DATR is very different from the corresponding operator in PATR, but this would be to misunderstand what has happened in this example. In fact, the semantics of the statement

```
    DAG1:
        <vp agr> == <v agr>.
```

taken in isolation is very similar to the semantics of the corresponding PATR statement: both assert equality of values associated with the two paths. The DATR statement is slightly weaker in that it allows the left-hand-side to be defined when the right-hand-side is undefined. But, even in DATR, if both sides are defined they must be the same, so, in principle, the value of the left-hand-side does semantically constrain the value of the right-hand-side. However, in a DATR description, specifying explicit values for extensions of the left-hand-side of such an equality constraint **overrides** its effect, and thus does not influence the values on its right-hand-side.

Another difference lies in the fact that DATR subpaths and superpaths can have values of their own:

```
    DAG2:
        <v agr> == sing
        <v agr per> == 3.
```

From this little description we can derive the following statements, inter alia:

```
    DAG2:
        <v agr> = sing
        <v agr num> = sing
        <v agr per> = 3
        <v agr per num> = 3.
```

From the perspective of a standard untyped DAG-encoding language like PATR, this is strange. In PATR, if <`v agr per`> has value `3`, then neither <`v agr`> nor <`v agr per num`> can have (atomic) values.

As these examples clearly show, DATR descriptions do not map **trivially** into (sets of) standard DAGs (although neither are they entirely dissimilar). But that does not mean that DATR descriptions cannot **describe** standard DAGs. Indeed, there are a variety of ways in which this can be done. An especially simple approach is possible when the DAGs one is interested are all built out of a set of paths whose identity is known in advance (Kilbury et al. 1991). In this case, we can use DATR paths as DAG paths, more or less directly:

```
    PRONOUN2:
        <referent> == '<' 'NP' referent '>'.
    She2:
        <> == PRONOUN2
        <case> == nominative
        <person> == third
        <number> == singular.
```

From this description, we can derive the following theorems:

```
    She2:
        <case> = nominative
        <person> = third
        <number> = singular
        <referent> = < NP referent >.
```

We can also derive the following un-DAG-like consequences, of course:

```
She2:
    <case person> = nominative
    <person number> = third
    <referent referent referent> = < NP referent >.
```

But these nonsensical theorems will be of no concern to a DATR-invoking NLP system that is able to specify in advance which paths are of interest for DAG-construction and to ignore all the rest[40].

A more sophisticated approach uses DATR itself to construct a DAG description (in the notation of your choice) as a value[41]:

```
    IDEM:
        <> ==
        <$atom> == $atom <>.
    PATH:
        <> == '<' IDEM '>'.
    LHS_EQ:
        <> == PATH '='.
    LHS_EQ_RHS:
        <> == LHS_EQ "<>".
    PRONOUN1:
        <dag> == [ LHS_EQ_RHS:<case>
                   LHS_EQ_RHS:<person>
                   LHS_EQ_RHS:<number>
                   LHS_EQ:<referent> PATH:<'NP' referent>  ].
    She1:
        <> == PRONOUN1
        <case> == nominative
        <person> == third
        <number> == singular.
```

From this description, we can derive the following theorem:

```
    She1:
        <dag> = [ < case > = nominative
                  < person > = third
                  < number > = singular
                  < referent > = < NP referent > ].
```

40 In this connection, see the discussion of "closure definitions" in Andry et al. (1992, 259-261).

41 This approach is due to recent unpublished work by Jim Kilbury. He has shown that the **same** DATR theorems can have their values realised as conventional attribute-value matrix representations, Prolog terms, or expressions of a feature logic, simply by changing the fine detail of the transducer employed.

The sequence of atoms on the right hand side of this equation is **just** a sequence of atoms as far as DATR is concerned. But a grammar or a parser that expects to see DAGs represented as they are here can interpret the DATR values as easily as it can the contents of a file[42].

5. Technical Issues

In this section we briefly discuss a number of technical issues, relating both to DATR as a formal language, and also to practical aspects of DATR in use.

5.1 Functionality

Most DATR descriptions consist only of definitional statements, and include at most one statement for each node/path pair. In this section we examine the significance of this observation from a formal perspective. As already noted in Section 2, DATR nodes can be thought of semantically as denoting partial functions from paths (sequences of atoms) to values (sequences of atoms)[43]. Generalising this view in the obvious way, whole DATR descriptions can be thought of as denoting functions from nodes to (partial) functions from paths to values. This semantic notion induces a notion of consistency for DATR descriptions: we say that a DATR description is **consistent** if and only if it has a coherent interpretation as a function, that is, if the extensional sentences defined (explicitly or implicitly) for each node constitute a (partial) function from paths to values.

The syntax of DATR does not itself prevent one from writing down inconsistent descriptions:

```
VERB:
    <syn cat> == verb
    <syn cat> == noun.
```

However, such descriptions are of no utility and it would be desirable to find a mechanical way of eliminating them. In pursuit of this, we can define a **syntactic** notion of functionality over DATR descriptions as follows:

> A DATR description is **functional** if and only if (i) it contains only definitional statements and (ii) those statements constitute a (partial) function from node/path pairs to descriptor sequences.

The significance of this syntactic notion arises from the following property:

> Every functional DATR description is consistent.

To understand why this is[44] note first that the default extension process preserves functionality, since it only adds definitional statements about new node/path pairs not already present in the original description. Local inheritance derives new statements associated

42 Indeed, it **will** be interpreting the contents of a file if DATR has been used to define a lexicon that has subsequently been compiled out, rather than being accessed directly by components of the NLP system (see Section 5.3, below). We are not, of course, claiming that textual representations will standardly provide the optimal interface between an implementation of DATR and the larger NLP system in which it is embedded (cf., e.g., Duda & Gebhardi 1994).

43 We continue to oversimplify matters here. As Keller (1995) points out, the meaning of a node depends on the global context, and a node thus really denotes a function from global contexts to partial functions from paths to values. Though important, this point is tangential to the issue addressed here.

44 For simplicity here, we consider only the case of length one descriptor sequences – the general case involves complications not relevant to the main point.

with a node/path pair, but at most one of these defines a value or global inheritance descriptor (since local inheritance ceases at that point). Thus although the local inheritance makes the description become syntactically non-functional, the specification of values or global descriptors remains functional. The value specifications map directly to extensional statements, while the global inheritance descriptors operate just as the local ones, adding at most one further value statement for each global inheritance statement, so that ultimately the consistency of the set of (extensional) value statements is assured.

This theorem cannot be strengthened to a biconditional, however, since consistent but non-functional DATR descriptions exist, as in the following examples:

```
    NONFUNC1:
        <a> == UNDEF
        <a> == 1.
    NONFUNC2:
        <a> == <b>
        <a> == 1
        <b> == 1.
    NONFUNC3:
        <a> == <b>
        <b> == <a>
        <a> == 1.
```

In NONFUNC1, UNDEF is a node with no associated definitions, so the first statement imposes no constraint on the value of <a>; in NONFUNC2, two definitions for <a> are provided which happen to define the same value; in NONFUNC3, we establish a mutual dependence between <a> and <b>, and then define a value for one (either) of them. However, we have not encountered any examples of nonfunctional but consistent descriptions which are not better expressed by a straightforward functional counterpart[45]. Indeed, we suspect (but have no proof) that every consistent DATR description is extensionally equivalent to (that is, defines the same extensional sentences as) a functional one.

In the light of these considerations, we assume here, and elsewhere, that functionality is a reasonable restriction to place on DATR descriptions[46]. The advantage of this is that it is completely trivial to check that a DATR description is functional and hence guarantee its consistency. In other words, we can substitute a straightforward syntactic constraint on descriptions for the less tractable notion of semantic consistency, apparently without significant loss of expressive power. Among other things, this means that implementations of DATR can thus either treat attempted violations of functionality as syntactic errors and require the user to eliminate them, or (more commonly in existing implementations) they can treat apparent violations as intentional corrections and silently erase earlier statements for the node and path for which a violation has been detected.

5.2 Multiple inheritance

Multiple inheritance, in inheritance network terminology, describes any situation where a node in an inheritance network inherits information from more than one other node in the network. Wherever this phenomenon occurs there is the potential for conflicting inheritance, i.e., when the information inherited from one node is inconsistent

45 NONFUNC3 perhaps comes closest, but adding statements about extensions of either <a> or <b> quickly breaks the illusion that the two are in some sense "unified".

46 This only applies to original source descriptions: as we mentioned above, the formal inference mechanisms that implement inheritance necessarily add statements to make a description nonfunctional. But since these can always be automatically determined, they need never appears explicitly in source descriptions.

with that inherited from another. Because of this, the handling of multiple inheritance is an issue which is central to the design of any formalism for representing inheritance networks.

For the formalism to be coherent, it must provide a way of avoiding or resolving any conflict which might arise. This might be by banning multiple inheritance altogether, restricting it so that conflicts are avoided, providing some mechanism for conflict resolution as part of the formalism itself, or providing the user of the formalism with the means to specify how the conflict should be resolved. Putting aside considerations of functionality for the moment, we see that, in DATR, both the second and third of these options are employed. The "longest-defined-subpath-wins" principle amounts to conflict resolution built into the formalism; however, it does not deal with every case: definitions such as

```
Node3:
    <> == Node1
    <> == Node2.
```

may result in unresolvable conflicts. Such conflicts could, of course, just be ruled out by appealing to their inconsistency, which, following a logical tradition, is grounds for ruling the description to be "improper".

Touretzky (1986, p70ff) provides a formal description of a number of properties that an inheritance network may have, and discusses their significance with respect to the problem of multiple inheritance. Tree-structured networks, as their name suggests, allow any node to inherit from at most one other node, so multiple inheritance conflicts cannot arise. **Orthogonal** networks allow a node to inherit from more than one other node, but the properties it inherits from each must be disjoint, so that again, no conflict can possibly arise.

The basic descriptive features of DATR allow the specification of simple orthogonal networks similar to Touretzky's. For example, if we write:

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

then we are specifying a network of three nodes (`A` `B`, and `C`), and two "predicates" (boolean-valued attributes coded as DATR paths <`a`> and <`b`>), with `C` inheriting a value for <`a`> from `A`, and for <`b`> from `B`. The network is orthogonal, since <`a`> and <`b`> represent distinct (sets of) predicates.

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

```
VERB:
    <cat> == verb.
NP_ARG:
    <arg cat> == np
    <arg case> == acc.
TR_VERB:
    <cat> == VERB
    <arg> == NP_ARG.
```

Here `TR_VERB` inherits from both `VERB` and `NP_ARG` but the path prefixes `cat` and `arg` ensure that the inheritance is orthogonal and that no conflict (e.g., in respect of <`cat`> values) can arise.

More generally, OMI is invaluable for partitioning the various different, and largely independent, aspects of lexical description conventionally associated with such initial path prefixes as `phn` (phonology), `mor` (morphology), `syn` (syntax), and `sem` (semantics). In the English verbal system, for example, most morphological subregularities (such as having a past participle form in *-en*) operate entirely independently of most syntactic subregularities (such as having a ditransitive subcategorisation frame). Within the semantic domain, Pustejovsky & Boguraev (1993, 214) introduce the expression **typed inheritance** for OMI and argue for its advantages in connection with the consistent assembly of the different facets of meaning associated with a lexical item.

The above examples of OMI are in fact instances of a more general phenomenon in DATR. We have already noted that the combination of the longest-defined-subpath-wins and logical consistency are the basis of DATR's support for coherent multiple inheritance. It turns out that functionality (which of course implies consistency) ensures orthogonality, so that OMI falls out as the most normal, natural mode of definition using DATR.

Finally here, we note that a number of recent lexical theories have invoked a form of inheritance in which multiple parents with **overlapping** domains are specified, and a priority ordering imposed to resolve potential inheritance conflicts (e.g., Flickinger 1987; Russell et al. 1992). In this **prioritised** multiple inheritance (PMI), precedence is given to nodes that come earlier in the ordering, so that the inherited value for a property comes from the first parent node in the ordering that defines that property, regardless of whether other later nodes also define it (possibly differently).

Surprisingly perhaps, DATR's version of OMI can be used to reconstruct PMI without making syntactic and semantic additions to the language. In fact, we have described elsewhere no fewer than three different techniques for capturing PMI in DATR (Evans et al. 1993). But DATR was primarily designed to facilitate OMI analyses of natural language lexicons and we do not believe that PMI treatments of the lexicon offer significant analytical or descriptive advantages.

5.3 Modes of use

Lexicons can either be developed by hand or, in principle at least, they can be induced from relevant data. Once created, lexicons get used for language understanding, language generation, or both. Lexicons that are in use also have to be maintained. At present, implementations of lexical representation systems are typically specialised to one or two of these tasks. A language for lexical knowledge representation is merely one component of a lexical representation system, of course, but its design may well have implications for the tasks noted above. A language that coded everything into bit strings might be fully adequate for the induction and generation tasks, say. But it would probably not facilitate manual lexicon maintenance.

From a more formal point of view, Barg (1994) provides the useful tabular conceptualisation of the inferential tasks that may be associated with a lexical representation language like DATRshown in Table 2, below. Consider the English verbal morphology facts that provided our running example in Section 2, above. The conventional inference task presupposes that we have a description (such as that given in that section) and a query (such as `Love: <mor past participle>`): the task is to infer the appropriate value for this query, namely `love ed`. This task is crucial to lexicon development and maintenance since it provides the means by which the developer can check the empirical adequacy of their analysis. It is also a task that is likely to figure in the on-line use of the lexicon in a language processing system, once the relevant lexical entry (i.e., the

	Theory	Query	Value
Conventional inference	*given*	*given*	*unknown*
Reverse query	*given*	*unknown*	*given*
Theory induction	*unknown*	*given*	*given*

Table 2
Possible inference tasks *(adapted from Barg 1994)*

relevant DATR node) has been determined, to recover information associated with the entry. And it is the task that does the compilation in systems that use a partially or fully compiled-out off-line lexicon (as in Andry et al. 1992).

The reverse query task again presupposes that we have a description available to us. But instead of starting with a known query, we start instead with a known value (`love ed`, say, and the task is to infer what queries would lead to that value (`Love`: `<mor past participle>`, `Love`: `<mor past tense sing one>`, etc.)[47], The ability to perform this kind of inference may also be useful in lexicon development and maintenance. However, its most obvious application is to "bootstrap" lexical access in language processing systems that make direct use of an on-line lexicon: given a surface form (in analysis) or a semantic form (in generation), we need to identify a lexical entry associated with that form by reverse query, and then access other lexical information associated with the entry by conventional inference. Langer (1994) gives an inference algorithm, based on the familiar chart data structure, for reverse querying DATR lexicons; and Gibbon (1993) describes EDQL (Extended DATR Query Language) which permits quantification into components of multisentence DATR queries.

The final task is that of theory induction. Here one starts with a set of known query-value pairs (`Love`: `<mor past participle> = love ed`., `Love`: `<mor pres tense sing three> = love s`., etc.) and the task is to induce a description that has those pairs as theorems under the application of conventional inference. In a world in which all the relevant data was already clearly set out in descriptive linguistic work, an algorithm that efficiently achieved this kind of induction would be the philosopher's stone to the construction of computational lexicons. In the real world, such an algorithm would still be useful for domains like morphology (where the quality and clarity of extant descriptive linguistic work is very high), for bootstrapping lexical descriptions for subsequent manual development by humans, for updating lexicons in the light of newly encountered lexical information, and for converting one kind of lexicon into a completely different kind of lexicon by inducing the latter from the output of the former. The automatic induction of (symbolic) lexicons from data is a very new research area in computational linguistics: Kilbury (1993), Kilbury et al. (1994), Light (1994) and Light et al. (1993) have proposed a variety of incremental algorithms that take a partial lexical hierarchy and elaborate it as necessary in the light of successively presented data sets, whilst Barg (1994) has presented a nonincremental algorithm that induces full DATR hierarchies from suitable data sets.

Since DATR is no more than a language, it does not itself dictate how a DATR lexicon is to be used. As it turns out, different researchers have used it very differently. Andry et al. (1992), in the context of a speech recognition task involving the parsing of "extremely large lattices of lexical hypotheses" (p248), opted for off-line compilation of their 2000 word DATR lexicons into pairs of on-line lexicons, one of which was encoded

47 An alternative formulation is to start with a known value and path, and the task is to infer the appropriate nodes.

with bit-vectors for speed and compactness. At the other extreme, Duda & Gebhardi (1994) present an interface between a PATR-based parser and a DATR lexicon where the former is dynamically linked to the latter and able to query it freely, in both conventional and reverse modes, without restriction. And Gibbon (1993) presents an implementation of a very flexible query language, EDQL, which allows quantification over any constituents of (possibly complex) DATR queries.

5.4 Implementations

As already noted, the inferential core of DATR is extremely simple to implement. We know of the existence of around a dozen different implementations of the language but there may well be others that we do not know of. The best known, and most widely available are our own (Brighton/Sussex), which is written in Prolog and runs on most Unix platforms, Gibbon's (Bielefeld) DDATR Scheme and NODE Sicstus Prolog implementations, and Kilbury's (Duesseldorf) QDATR Prolog implementation which runs (in compiled form) on PCs and on Sicstus Prolog under Unix. All of these are freely available on request, as is an extensive archive of over one hundred example fragments some of which illustrate formal techniques and others of which are applications of DATR to the lexical phonology, morphology, syntax or semantics of a wide variety of different languages[48]. Other interesting implementations that we are familiar with include the experimental reverse query implementation by Langer (Osnabrueck), Duda & Gebhardi's (Berlin) implementation that is dynamically linked to PATR, and Barg's (Duesseldorf) implementation of a system that induces DATR descriptions from extensional data sets.

6. Concluding Remarks

Our title for this paper is to be taken literally – DATR is a **language** for lexical knowledge representation. It is a kind of programming language, not a theoretical framework for the lexicon (in the way that HPSG is a theoretical framework for syntax, say). Clearly, the language is well suited to lexical frameworks that embrace, or are consistent with, nonmonotonicity and inheritance of properties through networks of nodes. But those two dispositions hardly constitute a restrictive notion of suitability in the context of contemporary NLP work. Nor are they absolute requirements: it is, for example, entirely possible to write useful DATR fragments that never override inherited values (and so are monotonic) or which define isolated nodes with no inheritance.

It is true, of course, that our examples, in this paper and elsewhere, reflect a particular set of assumptions about how NLP lexicons can be best organised. But, apart from the utility of inheritance and nonmonotonicity, we have been careful not to build those assumptions into the DATR language itself. There is, for example, no built-in assumption that lexicons should lexeme-based rather than, say, word- or morpheme-based.

Unlike some other NLP inheritance languages, DATR is not intended to provide the facilities of a particular syntactic formalism. Rather, it is intended to be a lexical formalism that can be used with any syntactic representation which can be encoded in terms of attributes and values. Thus, at the time of writing, we know of nontrivial DATR lexicons written for GPSG, LTAG, PATR, Unification Categorial Grammar, and Word Grammar. Equally, the use of DATR does not commit one, in advance, to adopting any particular set of theoretical assumptions with respect to phonology, morphology or semantics. In phonology, for example, the language allows one to write transducers that

48 Anonymous FTP to `ftp.cogs.sussex.ac.uk` and directory `/pub/nlp/DATR` provides access to various DATR implementations, the example archive, and some relevant papers and documentation.

map strings of atomic phonemes to strings of atomic phones. But it also allows one to encode full-blown feature and syllable-tree based prosodic analyses.

Unlike the formalisms typically proposed by linguists, DATR does not attempt to embody in its design any substantive and restrictive universal claims about the lexicons of natural language. That does not distinguish it from most NLP formalisms, of course. However, we have also sought to ensure that its design does not embody features that would restrict its use to a single language (English, say) or to a particular class of closely related languages (the Romance class, say). The available evidence suggests that we have succeeded in the latter aim since, at the time of writing, nontrivial DATR fragments of the lexicons of Arabic, Arapesh, Czech, English, French, German, Gikuyu, Italian, Latin, Polish, Portuguese, Russian and Spanish have been developed. There are also smaller indicative fragments for Baoule, Dakota, Dan, Dutch, Japanese, Nyanja, Sanskrit, Serbo-Croat, Swahili and Tem.

Unlike most other languages proposed for lexical knowledge representation, DATR is not intended to be restricted in the levels of linguistic description to which it can sensibly be applied. It is designed to be equally applicable at phonological, orthographic, morphological, syntactic and semantic levels of description. But it is not intended to replace existing approaches to those levels. Rather, we envisage descriptions of different levels according to different theoretical frameworks being implementable in DATR: thus an NLP group might decide, for example, to build a lexicon with DRT-style semantic representations, HPSG-style syntactic representations, "item & arrangement" style morphological representations and a KIMMO-style orthographic component, implementing all of these, including the HPSG lexical rules, in DATR. DATR itself does not mandate any of the choices in this example, but equally nor does it allow such choices to be avoided[49]. DATR cannot be (sensibly) used without a prior decision as to the theoretical frameworks in which the description is to be conducted; there is no "default" framework for describing morphological facts in DATR, say. Thus, for example, Gibbon (1992) and Langer & Gibbon (1992) use DATR to implement their ILEX theory of lexical organisation, Corbett & Fraser (1993) and Fraser & Corbett (in press) use DATR to implement their Network Morphology framework, and Gazdar (1992) shows how Paradigm Function Morphology analyses (Stump 1992) can be mapped into DATR. Indeed, it would not be entirely misleading to think of DATR as a kind of assembly language for constructing (or reconstructing) higher level theories of lexical representation.

Acknowledgments
We are grateful to the four *CL* referees for their criticisms and suggestions; to Lynne Cahill, Dafydd Gibbon, Jim Kilbury and David Weir for their comments on an earlier draft of the paper; and to Walter Daelemans and John Nerbonne for their comments on the first draft of the appendix. We thank Petra Barg, Lynne Cahill, Norman Fraser, Dafydd Gibbon, Elizabeth Jenkins, Jim Kilbury, Lionel Moser, and Ingrid Renz for their role(s) in the development of the DATR language and the coding techniques discussed above; Fernando Pereira for early critical comments that led directly to the introduction of evaluable paths into the language; and Bill Keller and David Weir for much relevant recent interaction. This research was supported by various grants to the authors from ESRC (UK) and SERC/EPSRC (UK).

49 However, DATR's framework-agnosticism may make it a plausible candidate for the construction of polytheoretic lexicons. For example, one that would allow either categorial or HPSG-style subcategorisation specifications to be derived, depending on the setting of a parameter.

References

Francois Andry, Norman Fraser, Scott McGlashan, Simon Thornton, & Nick Youd (1992) Making DATR work for speech: lexicon compilation in SUNDIAL. *Computational Linguistics* **18,** 245-267.

Petra Barg (1994) Automatic acquisition of DATR theories from observations. Theories des Lexicons: Arbeiten des Sonderforschungsbereichs 282, Heinrich-Heine University of Duesseldorf.

Doris Bleiching (1992) Prosodisches Wissen in Lexicon. In G. Goerz, ed., *Proceedings of KONVENS-92,* Berlin: Springer-Verlag, 59-68.

Doris Bleiching (1994) Integration von Morphophonologie und Prosodie in ein hierarchisches Lexicon. In Harald Trost, ed., *Proceedings of KONVENS-94,* Vienna: Oesterreichische Gesellschaft fuer Artificial Intelligence, 32-41

Gosse Bouma (1993) *Nonmonotonicity and categorial unification grammar.* Proefschrift, Rijksuniversiteit Groningen.

Gosse Bouma & John Nerbonne (1994) Lexicons for feature-based systems. In Harald Trost, ed., *Proceedings of KONVENS-94,* Vienna: Oesterreichische Gesellschaft fuer Artificial Intelligence, 42-51.

Ted Briscoe & Ann Copestake (1991) Sense extensions as lexical rules. In D. Fass, E. Hinkelman & J. Martin, eds. *Computational approaches to Non-Literal Language, Proceedings of the IJCAI Workshop,* Sydney, 12-20.

Ted Briscoe, Ann Copestake & Alex Lascarides (1995) Blocking. In Patrick Saint-Dizier & Evelyne Viegas, eds. *Computational Lexical Semantics.* Cambridge: Cambridge: Cambridge University Press, 272-302.

Ted Briscoe, Valeria de Paiva & Ann Copestake, eds. (1993) *Inheritance, Defaults, and the Lexicon,* Cambridge: Cambridge University Press.

Dunstan Brown & Andrew Hippisley (1994) Conflict in Russian genitive plural assignment: A solution represented in DATR . *Journal of Slavic Linguistics,* **2(1),** 48-76.

Lynne Cahill (1993a) Some reflections on the conversion of the TIC lexicon into DATR. In Ted Briscoe, Valeria de Paiva and Ann Copestake, eds. *Inheritance, defaults, and the lexicon.* Cambridge, Cambridge University Press, 47-57.

Lynne Cahill (1993b) Morphonology in the lexicon. *Sixth Conference of the European Chapter of the Association for Computational Linguistics,* 87-96.

Lynne Cahill (1994) An inheritance-based lexicon for message understanding systems. *Fourth ACL Conference on Applied Natural Language Processing,* 211-212

Lynne Cahill & Roger Evans (1990) An application of DATR: the TIC lexicon, in *Proceedings of the 9th European Conference on Artificial Intelligence,* Stockholm, 120-125.

Jo Calder (1994) Feature-value logics: some limits on the role of defaults. In C.J. Rupp, M.A. Rosner & R.L. Johnson, eds. *Constraints, Language and Computation.* London: Academic Press, 205-222.

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, *Cognitive Science Research Paper* **CSRP 280.**

Ann Copestake & Ted Briscoe (1992) Lexical operations in a unification based framework. In James Pustejovsky & Sabine Bergler, eds. *Lexical Semantics and Knowledge Representation.* Berlin: Springer-Verlag, 101-119.

Ann Copestake & Ted Briscoe (1995) Regular polysemy and semi-productive sense extension. *Journal of Semantics* **12,** 15-67.

Greville Corbett & Norman Fraser (1993) Network Morphology: a DATR account of Russian nominal inflection. *Journal of Linguistics* **29,** 113-142.

Walter Daelemans (1994) Review of Ted Briscoe, Valeria de Paiva & Ann Copestake, eds. (1993) *Inheritance, Defaults, and the Lexicon,* Cambridge: Cambridge University Press. *Computational Linguistics* **20.4,** 661-664.

Walter Daelemans & Koenraad De Smedt (1994) Inheritance in an object-oriented representation of linguistic categories. *International Journal of*

Human-Computer Studies **41.1/2,** 149-177.

Walter Daelemans, Koenraad De Smedt & Gerald Gazdar (1992) Inheritance in natural language processing . *Computational Linguistics* **18.2,** 205-218.

Walter Daelemans & Gerald Gazdar, eds. (1992) *Computational Linguistics* **18.2** & **18.3**, special issues on inheritance.

Walter Daelemans & Erik-Jan van der Linden (1992) Evaluation of lexical representation formalisms. In Jan van Eijck & Wilfried Meyer, eds. *Computational Linguistics in the Netherlands: Papers from the Second CLIN Meeting.* Utrecht: OTS, 54-67.

Marc Domenig & Pius ten Hacken (1992) *Word Manager: A System for Morphological Dictionaries.* Hidesheim: Georg Olms Verlag.

Markus Duda & Gunter Gebhardi (1994) DUTR – A DATR-PATR interface formalism. In Harald Trost, ed., *Proceedings of KONVENS-94,* Vienna: Oesterreichische Gesellschaft fuer Artificial Intelligence, 411-414.

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

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

Roger Evans, Gerald Gazdar & Lionel Moser (1993) Prioritised multiple inheritance in DATR. In Ted Briscoe, Valeria de Paiva and Ann Copestake, eds. *Inheritance, defaults, and the lexicon.* Cambridge, Cambridge University Press, 38-46.

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,* doctoral dissertation, Stanford University.

Norman Fraser & Greville Corbett (1995) Gender, animacy, and declensional class assignment: a unified account for Russian. In Geert Booij & Jaap van Marle, eds. *Year Book of Morphology 1994.* Dordrecht: Kluwer, 123-150.

Norman Fraser & Greville Corbett (in press) Gender assignment in Arapesh: a Network Morphology analysis. *Lingua* **00**, 00-00.

Norman Fraser & Richard Hudson (1990) Word Grammar: an inheritance-based theory of language. In Walter Daelemans & Gerald Gazdar, eds. *Proceedings of the Workshop on Inheritance in Natural Language Processing.* Tilburg: Institute for Language Technology, 58-64.

Gerald Gazdar (1992) Paradigm function morphology in DATR. In Lynne Cahill & Richard Coates, eds. *Sussex Papers in General and Computational Linguistics.* Brighton, University of Sussex, Cognitive Science Research Paper **CSRP 239,** 43-53.

Dafydd Gibbon (1990) Prosodic association by template inheritance. In Walter Daelemans & Gerald Gazdar, eds. *Proceedings of the Workshop on Inheritance in Natural Language Processing.* Tilburg: Institute for Language Technology, 65-81.

Dafydd Gibbon (1992) ILEX: a linguistic approach to computational lexica. In Ursula Klenk, ed. *Computatio Linguae: Aufsaze zur algorithmischen und quantitativen Analyse der Sprache (Zeitschrift fur Dialektologie und Linguistik,* Beiheft 73), Stuttgart: Franz Steiner Verlag, 32-53.

Dafydd Gibbon (1993) Generalized DATR for flexible lexical access: PROLOG specification. Bielefeld: *Verbmobil Report* **2**.

Dafydd Gibbon & Doris Bleiching (1991) An ILEX model for German compound stress in DATR. *Proceedings of the FORWISS-ASL Workshop on Prosody in Man-Machine Communication,* 1-6.

Nancy Ide, Jacques Le Maitre & Jean Véronis (1994) Outline of a model for lexical databases. In Antonio Zampolli, Nicoletta Calzolari & Martha Palmer, eds. *Current Issues in Computational Linguistics: In Honour of Don Walker.* Pisa: Kluwer, 283-320.

Ronald M. Kaplan & Martin Kay (1994) Regular models of phonological rule systems. *Computational Linguistics* **20.3,** 331-378.

William Keller (1995) DATR theories and DATR models. *33rd Annual Meeting of the Association for Computational Linguistics,* 55-62.

James Kilbury (1993) Strict inheritance and the taxonomy of lexical types in DATR. Unpublished manuscript, University of Duesseldorf.

James Kilbury, Petra [Barg] Naerger & Ingrid Renz (1991) DATR as a lexical component for PATR. *Fifth Conference of the European Chapter of the Association* *for Computational Linguistics,* 137-142.

James Kilbury, Petra [Barg] Naerger & Ingrid Renz (1994) Simulation lexicalischen Erwerbs In Sascha W. Felix, Christopher Habel & Gert Rickheit *Kognitive Linguistik: Repraesentation und Prozesse.* Opladen: Westdeutscher Verlag, 251-271.

Adam Kilgarriff (1993) Inheriting verb alternations. *Sixth Conference of the European Chapter of the Association for Computational Linguistics,* 213-221.

Adam Kilgarriff (1995) Inheriting polysemy. In Patrick Saint-Dizier & Evelyne Viegas, eds. *Computational Lexical Semantics.* Cambridge: Cambridge: Cambridge University Press, 00-00.

Adam Kilgarriff & Gerald Gazdar (1995) Polysemous relations. In F.R. Palmer, ed. *Grammar and Meaning: Essays in Honour of Sir John Lyons.* Cambridge: Cambridge University Press, 1-25.

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.

Hans-Ulrich Krieger & John Nerbonne (1993) Feature-based inheritance networks for computational lexicons. In Ted Briscoe, Valeria de Paiva and Ann Copestake, eds. *Inheritance, defaults, and the lexicon.* Cambridge, Cambridge University Press, 90-136.

Hans-Ulrich Krieger, Hannes Pirker & John Nerbonne (1993) Feature-based allomorphy. *31st Annual Meeting of the Association for Computational Linguistics,* 140-147.

Hagen Langer (1994) Reverse queries in DATR. *COLING-94,* 1089-1095.

Hagen Langer & Dafydd Gibbon (1992) DATR as a graph representation language for ILEX speech oriented lexica. Technical Report **ASL-TR-43-92/UBI,** University of Bielefeld.

Alex Lascarides, Nicholas Asher, Ted Briscoe & Ann Copestake (forthcoming) Order independent and persistent typed default unification. To appear in *Linguistics & Philosophy.*

Marc Light (1994) Classification in feature-based default inheritance hierarchies. In Harald Trost, ed., *Proceedings of KONVENS-94,* Vienna: Oesterreichische Gesellschaft fuer Artificial Intelligence, 220-229.

Marc Light, Sabine Reinhard & Marie Boyle-Hinrichs (1993) INSYST: an automatic inserter system for hierarchical lexica. *Sixth Conference of the European Chapter of the Association for Computational Linguistics,* 471.

Paul McFetridge & Aline Villavicencio (1995) A hierarchical description of the Portuguese verb. *Proceedings of the XIIth Brazilian Symposium on Artificial Intelligence,* 00-00.

Chris Mellish & Ehud Reiter (1993) Using classification as a programming language. *IJCAI-93,* 696-701.

Teruko Mitamura & Eric H. Nyberg III (1992) Hierarchical lexical structure and interpretive mapping in machine translation. *COLING-92* Vol. IV, 1254-1258.

John Nerbonne (1992) Feature-based lexicons – an example and a comparison to DATR. In Dorothee Reimann, ed. *Beitrage des ASL-Lexicon-Workshops.* Wandtlitz, 36-49.

Nicholas Ostler & B.T.S. Atkins (1992) Predictable meaning shift: some linguistic properties of lexical implication rules. In James Pustejovsky & Sabine Bergler, eds. *Lexical Semantics and Knowledge Representation.* Berlin: Springer-Verlag, 87-100.

Gerald Penn & Richmond Thomason (1994) Default finite state machines and finite state phonology. *Computational Phonology: Proceedings of the 1st Meeting of the ACL Special Interest Group in Computational Phonology,* 33-42.

Stephen G. Pulman (forthcoming) Unification encodings of rich grammatical formalisms. To appear in *Computational Linguistics.*

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

James Pustejovsky & Branimir Boguraev (1993) Lexical knowledge representation and natural language processing. *Artificial Intelligence* **63.1-2,** 193-223.

Sabine Reinhard (1990) Verarbeitungsprobleme nichtlinearer Morphologien: Umlautbeschreibung in einem hierarchischen Lexikon. In Burghard Rieger & Burkhard Schaeder *Lexikon und Lexikographie.* Hildesheim: Olms Verlag, 45-61.

Sabine Reinhard & Dafydd Gibbon (1991) Prosodic inheritance and morphological generalisations. *Fifth Conference of the European Chapter of the Association for Computational Linguistics,* 131-136.

Ehud Reiter & Chris Mellish (1992) Using classification to generate text. *30th Annual Meeting of the Association for Computational Linguistics,* 265-272.

Graeme D. Ritchie, Graham J. Russell,

Alan W. Black and Stephen G. Pulman (1992) *Computational Morphology.* Cambridge, Ma.: MIT Press.

Graham Russell (1993) Review of Marc Domenig & Pius ten Hacken (1992) *Word Manager: A System for Morphological Dictionaries.* Hidesheim: Georg Olms Verlag. *Computational Linguistics* **19.4,** 699-700.

Graham Russell, Afzal Ballim, John Carroll & Susan Warwick-Armstrong (1992) A practical approach to multiple default inheritance for unification-based lexicons. *Computational Linguistics* **183,** 311-337.

Harvey Sacks (1973) On some puns with some intimations. In Roger W. Shuy. ed. *Report of the 23rd Annual Roundtable Meeting on Linguistics and Language Studies.* Washington D.C.: Georgetown University Press, 135-144.

Stuart M. Shieber (1986) *An Introduction to Unification Approaches to Grammar.* Stanford: CSLI/Chicago University Press.

Greg Stump (1992) On the theoretical status of position class restrictions on inflectional affixes. In Geert Booij & Jaap van Marle, eds. *Year Book of Morphology 1991.* Dordrecht: Kluwer, 211-241.

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

Mark A. Young (1992) Nonmonotonic sorts for feature structures. *AAAI-92,* 596-601.

Mark A. Young & Bill Rounds (1993) A logical semantics for nonmonotonic sorts. *Proceedings of the 31st Annual Meeting of the ACL,* 209-215.

APPENDIX: The critical literature on DATR reviewed

Since DATR has been in the public domain for the last half dozen years and been widely used in Europe during that period (by the standards of lexical knowledge representation languages), it is not surprising that it has attracted some critical attention from others working in the field. In this appendix, we consider and respond to the critical material that has been published: Domenig & ten Hacken (1992), Bouma & Nerbonne (1994), Nerbonne (1992), Krieger & Nerbonne (1993), and Daelemans & van der Linden (1992). Langer & Gibbon (1992) also respond to the last three papers in the context of a thorough general review of appropriate evaluation criteria for lexical knowledge representation formalisms. We are indebted to their discussion.

Domenig & ten Hacken (1992) base part of their critique of DATR on an idiosyncratic analysis of the English *-s/-es* suffix choice as a matter of pure morphology. This may be because they are considering DATR as a candidate "FMP" – formalism for morphological processing – even though, as they note "DATR is strictly speaking not an FMP" (p8) and "is not specifically geared to morphological processing" (p15). As they point out, dealing with the choice morphologically leads to undermotivated inflectional subclasses, obscures the role of phonology in the choice of form, and misses the morphophonological generalisation that unites nouns and verbs in respect of the choice. But their critique is based on the assumption that they have identified "the most natural way to express [the choice] in DATR" (p17). Given the well-known facts of this phenomenon[50], their analysis seems to us to be about as unnatural as it could be. Depending on the nature and purpose of one's lexicon, it would be much more natural to deal with the choice orthographically with a DATR-coded FST of the kind discussed in Section 4.3, above, or morphophonologically using the kind of phonological representation adopted by Reinhard & Gibbon (1991), say.

Domenig & ten Hacken actually cite this latter paper in connection with German umlaut and suggest that the *-s/-es* alternation might be handled in the same way. However, they go on to claim, quite incorrectly, that "morphophonological generalisations can actually not be expressed as such" in DATR because "they are represented as properties of the individual lexemes" (pp23-24). This claim appears to be based on a false assumption that DATR nodes are somehow restricted to the description of lexemes. This is an odd assumption to make given that the Reinhard & Gibbon paper postulates nodes for vowels, syllables, stems, stem types, prefixes, plural inflection, and syntactic categories, as well as lexemes. But then they dismiss the analysis given in that paper as "incomprehensible" (p24).

A related straw man appears in their discussion of how alternation within a morphological paradigm might be represented in DATR (p22). They once again postulate an analysis that proliferates nodes beyond necessity and fail to spot the possibility of path domain or value domain analyses such as those sketched in Section 4.6, above. They go on to offer a "slightly speculative" evaluation of ways in which DATR might be able to represent word formation, concluding that they "do not see any possibility of representing the rules involved in word formation" (p22). This conclusion again appears to be based on their assumption that DATR nodes are somehow restricted to the description of lexemes. But DATR, of course, knows nothing about lexemes, affixes, vowels, words, lexical rules, or whatever. These are higher level notions that the analyst may choose to represent in a wide variety of ways.

50 The orthographic alternation applies to the third person singular present tense forms of verbs and the plural forms of nouns. The choice between the alternants is **wholly** governed by the phonology of the verb or noun stem.

Finally, Domenig & ten Hacken contend that lexical inheritance formalisms (and thus DATR) are unusable for the purpose for which they were designed because the humans who have to work with them for lexicon development cannot keep track of all the interactions. They provide no evidence for this assertion and the widespread adoption, development and use of a variety of large inheritance lexicons in working NLP systems over the last few years make the assertion seem somewhat implausible. They conclude that their evaluation of DATR has been "unfair" (p29) because they failed to consider the language in its natural environment. We agree that their evaluation is unfair but ascribe the cause to the ways in which they attempted to apply DATR to their chosen tasks[51].

Daelemans & van der Linden (1992) review a number of approaches to lexical knowledge representation, including DATR, with respect to their notational adequacy and expressivity. They argue that adequate approaches will allow (i) recursive path formation; (ii) multiple inheritance, preferably orthogonal multiple inheritance; (iii) non-monotonic inheritance; and require (iv) that irregular items take precedence over regular ones without explicit coding (p61). Since, as we have seen, and as Langer & Gibbon (1992) note, DATR has all four of these properties, one might expect it to emerge from their review with at least a low alpha grade. But in fact they find fault with it on a number of grounds.

The first of these is the use of double quotes to mark global inheritance in the concrete syntax of DATR. They claim that global inheritance is the normal kind of inheritance in DATR and should thus not be marked in any special way, whilst (unquoted) local inheritance is exceptional and should therefore have a special notation (like quotes) associated with it (p63)[52]. The small example they give lends some plausibility to their claim. However, the claim is nonetheless misleading. Quoted paths (the only instances of global inheritance to be found in their example fragment) are indeed ubiquitous at the highest nodes of existing DATR fragments. But unquoted nodes, unquoted paths and unquoted node/path pairs all also occur very frequently in existing DATR fragments, whilst quoted nodes and quoted node/path pairs are hardly found at all. And, in some common applications of DATR, such as FSTs, no use at all may be made of global inheritance.

Their second objection is to the way path extension in DATR permits the derivation of theorems that have no interpretation in the domain being modelled (p63). Thus, for example, a description that had `(a)` as a (sensible) theorem might also have `(b)` as one of an infinity of (silly) theorems:

```
(a)  Parrot:<mor plur> = parrot s.
(b)  Parrot:<mor plur past perfect> = parrot s.
```

The issue here is that while DATR encourages abstraction away from the most specific level of detail wherever possible, it does not itself provide a built-in mechanism for stating what that most specific level is. Our position is that this is part of the lexical metatheory, rather than the lexical description itself. It needs to be known by anyone (or any system) wishing to access the lexicon properly, and it may be practically useful to constrain access by checking for the well-formedness of queries according to such a metatheory – this could be done quite straightforwardly in DATR as an adjunct to the main lexicon if desired. But this notion is external to, and independent of, the lexical description itself: the range of sensible queries only weakly constrains the manner in which their values are defined.

Their third objection concerns multiple inheritance. They draw attention to the fact that DATR's normal mode of multiple inheritance is orthogonal and complain that prioritised multiple inheritance can only be expressed with additional DATR code (p63). How-

51 For another critical discussion of the same Domenig & ten Hacken material, see Russell (1993).

52 One of our referees comments that "the issue .. appears to be rather scholastic". We agree.

ever, we agree with their earlier comment "that orthogonal multiple default inheritance is at this stage the best solution for conflicts" (p61) and can see no computational linguistic motivation for equipping DATR with a further primitive inheritance mechanism[53].

Their fourth objection consists of the claim that "it is not possible in DATR to have complex structured objects as values" (p64). In one sense this is true since DATR values are simply sequences of atoms. But although true, it does not provide support for a sound objection. DATR can construct those sequences of atoms on the basis of a complex recursive description, and the atom sequences themselves can **represent** complex recursive objects so far as NLP system components outside the lexicon are concerned. The sequences of atoms that DATR provides as values simply constitute an interface for the lexicon that is entirely neutral with respect to the representational requirements of external components. For what is intended to be a general purpose lexical knowledge representation language, not tied to any particular conceptions of linguistic structure or NLP formalism, this neutrality seems to us to be a feature, not a bug.

In a fifth objection, they note correctly that the semantics of paths in DATR and PATR is different but then go on to claim that DATR paths "could be better described as atomic attributes" that "do not correspond with a recursive structure" and whose "only function is to support prefix matching" (p64). None of these latter claims are true. If DATR paths were atomic attributes then our Section 4.3, above, on finite state transducers could not have been written; DATR paths are the same as PATR paths as far as recursive structure is concerned; and, as we have seen throughout this paper, DATR paths have many functions in addition to prefix matching.

In a final set of comments, they briefly raise various issues connected with the integration of DATR lexicons with unification based grammar (p64). We have dealt with these issues in earlier parts of the present paper and will not rehearse them here[54].

Krieger & Nerbonne (1993) claim that "the basic insight of DATR" lies in its use in characterising "the inflectional variants of a lexeme as alternative (disjunctive) realisations" (p135). This claim confuses the basic insight of a very traditional approach to inflectional morphology with the application of DATR in implementing that approach. Elsewhere they note that "the fundamental idea in our characterisation is due to the work in DATR, in which paradigms are treated as alternative further specifications of abstract lexemes" (p104). Unfortunately, their own implementation of this fundamental idea turns out to be significantly less satisfactory than that provided in the DATR analysis to which they refer. In order to reconstruct paradigms in their feature language, they invoke distributed disjunctions (fixed length term expressions)[55]. The descriptive problem with this approach, as they admit, is that "there is no way to note that a single form in a paradigm is exceptional without respecifying the entire paradigm – the disjunction must be respecified as a whole .. there is no way to identify a particular alternation within the distributed disjunction" (p107). Anyone familiar with the way inflection works in Romance languages will immediately see that this is a very serious weakness. In Latin, for example, there are many individual words and small subclasses of words that deviate from a major declension or conjugation in just one or two parts of the paradigm. Under Krieger & Nerbonne's approach every such case will require one to "respecify the entire paradigmatic disjunction" (p107). This is exactly the kind of redundancy that the

53 But see Daelemans & De Smedt (1994) for articulation of the methodological principle that underlies the third objection.

54 The issues are also discussed in detail by Langer & Gibbon (1992).

55 Langer & Gibbon (1992) argue, at some length, that it is formally inappropriate to add distributed disjunction to a typed feature structure language of the kind otherwise assumed by Krieger & Nerbonne.

introduction of defaults is meant to eliminate[56].

At the root of Krieger & Nerbonne's (1993) critique of DATR is a complaint that it fails to provide all the resources of a modern fully equipped unification grammar formalism (p90-91). From our perspective, this is a bit like criticising STANDARD ML on the grounds that it lacks the functionality provided in ADA. Thus, for example, they complain that disjunction is missing from DATR and that nobody seems to be trying to add it to the language (p110). They cite their own "extensive employment of [distributed] disjunction" (p110) as evidence for its utility in lexical description, apparently forgetting that their use of distributed disjunction to describe inflection was motivated by a desire to reconstruct a treatment of inflection that they had seen implemented in DATR. They provide no motivation for adding distributed disjunction to DATR's (rather small) list of available descriptive resources because that list of resources already allows better analyses of the phenomena they discuss than does their version of distributed disjunction, as noted above.

They also object to the fact that use of a DATR lexicon will require an "interface" (p110) between the lexicon and a feature-based parser. But, as we have seen in Section 4.7 above, such an interface will normally be trivial and required in any case (since Krieger & Nerbonne's parser must be able to access and read files that contain text descriptions of feature structures). As it is, they seem happy to countenance an interface to a separate two-level morphophonemic processor (p103, n9) whereas, in DATR, the morphophonemics can be done entirely in the lexicon if one wishes.

From remarks they make on pages 109 and 111 of their paper, Krieger & Nerbonne appear to believe that it is impossible to implement a particular inflectional analysis of the passive in Latin in DATR. They do not provide much of an argument but what they do say suggests that the simple treatment of passive given in Section 4.5, above, is likewise impossible. This may be because they regard their own **internal** interpretation of lexical rules as "novel" (p113) although examples of that interpretation of lexical rules appear in earlier DATR work that they cite.

Many of the points made in Nerbonne (1992) are repeated from the more accessible Krieger & Nerbonne (1993)[57] and we have considered them in our discussion of the latter. Some of the points from the 1992 and 1993 papers resurface again in Bouma & Nerbonne (1994). Nerbonne appears to misconstrue Evans et al. (1993) as an attempt to augment DATR with reentrancy and goes on to suggest that DATR is somehow forced to maintain that "all linguistic generalisations tend to follow the lines of morphological form" (p47) when, in fact, the attribute ordering used in a DATR treatment of morphology is entirely independent of the use and ordering of those same attributes elsewhere in the lexicon (see the discussion at the end of Section 4.1, above). Like Daelemans & van der Linden (1992), he makes some pessimistic comments about the integration of a DATR lexicon with feature-based grammars. Some of these are effectively dealt with elsewhere in this paper, but two of them need to be noted here. He asserts that if a rich feature formalism is encoded in DATR then "distinctions must be lost". It is not clear from the context exactly which distinctions he has in mind or what the basis for the claim is. But the expressions of all existing feature formalisms can be represented by sequences of atoms (and thus by DATR values) and all existing lexicons for feature-based NLP systems use such representations. We therefore find the claim deeply implausible. He also asserts that

56 Krieger & Nerbonne are not **forced** to use distributed disjunction to describe inflectional paradigms. Their very well-equipped feature description language provides alternative analytical resources. What puzzles us is why they **chose** to use distributed disjunction for this purpose. Bouma & Nerbonne (1994) propose using lists of specified phonological forms instead.

57 Although the joint 1993 paper has a later publication date, it appears to have been written first.

the fact that an atom may mean one thing in the semantics of DATR and something quite different in the semantics of a feature formalism will lead to "massive redundancy" (p47) in lexical specifications (the phrase gets repeated in Bouma & Nerbonne 1994). Again, no argument in support of this conclusion is offered. And we cannot see how semantic overloading of atoms gives rise, of itself, to any kind of redundancy[58]. Indeed, those who design programming languages normally introduce semantic overloading in order to achieve economy of expression.

Finally, Bouma & Nerbonne (1994) comment that "in spite of Kilgarriff's (1993) interesting work on modelling some derivational relations in the pure inheritance machinery of DATR, we know of no work attempting to model potentially recursive derivational relations, and we remain sceptical about relying on inheritance alone for this". We are not sure what they mean by "the pure inheritance machinery of DATR" or why they think that someone attempting an analysis of recursive derivation in DATR would want to do so using "pure inheritance" alone. Here is a trivial (and linguistically rather pointless) DATR analysis of the more complex of their two examples:

```
    Word:
        <v> == "<>"
        <a from n> == <n> + al
        <v from a> == <a> + ize
        <n from v> == <v> + tion.
    Institute:
        <> == Word
        <root> == institute.
```

From this description we can derive theorems like these:

```
    Institute:
        <root> = institute
        <n from v root> = institute + tion
        <a from n from v root> =
                          institute + tion + al
        <v from a from n from v root> =
                          institute + tion + al + ize
        <n from v from a from n from v root> =
                          institute + tion + al + ize + tion.
```

Note the recursive reintroduction of the *tion* suffix in the last theorem shown.

58 Sacks (1973) makes interesting reading in this connection.