1

Polysemous relations

ADAM KILGARRIFF AND GERALD GAZDAR

1 Introduction

In the section of Lyons' *Semantics* that deals with the distinction between homonymy and polysemy, he notes that a major criterion 'is unrelatedness vs. relatedness of meaning . . . indeed, it is arguable that it is the only synchronically relevant consideration' (1977: 551). However, he goes on to argue that all attempts 'to explicate the notion of relatedness of meaning in terms of a componential analysis of the senses of lexemes . . . have so far failed' (1977: 552–3). Although Lyons is sympathetic to a treatment of the lexicon that seeks to maximise polysemy at the expense of homonymy, he does not himself go on in that book to reconstruct the notion of relatedness of meaning that such a treatment requires.

Discussing polysemy some nine years earlier, Lyons was a little more explicit about what might be required: 'Various . . . types of "extension" or "transference" of meaning were recognized by the Greek grammarians, and have passed into traditional works on rhetoric, logic, and semantics. Meanings that are more or less closely "related" in accordance with such principles are not traditionally regarded as being sufficiently different to justify the recognition of distinct words' (1968: 406). Metaphorical extension is the only kind of extension explicitly discussed in connection with polysemy in *Introduction to Theoretical Linguistics* although it is clear from the quotation just given that it was not the only one that Lyons had in mind. It is surprising, therefore, to find the following passage in a later work: 'it is metaphorical extension . . . that is at issue when one refers to the related meanings of polysemous lexemes. There are, of course, other kinds of relatedness of meaning, which are irrelevant in this connection' (1981: 47).

The opening lines of the entry for *silk* in Flexner (1987: 1780) read as follows:

> *silk* (silk), *n.* 1. the soft lustrous fiber obtained as a filament from the cocoon of the silkworm. 2. thread made from this fiber. 3. cloth made from this fiber. 4. a garment of this cloth.

If the notion of polysemy is to earn its keep, then it must surely be applicable to such a set of senses. And yet the relation between a meaning denoting a fibre and a meaning denoting a thread made from that fibre, or that between a meaning denoting a cloth and a meaning denoting a garment made from that cloth, is surely that of metonymy rather than metaphorical extension. Our concern in this chapter, however, is not to explore the range of polysemous relations that dictionaries attest to, but rather to attend to their systematic and partial regularity. If a relation holds for *silk* then it may well also hold for *cotton*. And if a relation holds for *silk*, *cotton* and *wool*, then it probably also holds for less familiar words like *guanaco*.

Our topic is thus what Apresjan calls 'regular polysemy' (1974). He distinguishes cases of polysemy where the same relationship holds between the senses for two or more polysemous words from those where the relationship is particular to a single word. He points to a similarity between relations of regular polysemy and those of derivational morphology and proceeds to catalogue the regular polysemous relations of Russian. But he does not explore the formal structure of the regularities, nor does he address their exception-prone character. And he does not consider how such relations might be called into service as part of an account of lexical structure. Apresjan's phrase, 'regular polysemy', is potentially misleading in a couple of respects. Firstly, by a standard Gricean inference, use of the phrase implies that there might be a category of 'irregular polysemy'. However, it seems to us that, once we have a fully developed theory of subregularity, it is unlikely that a distinction between 'irregular polysemy' and 'homonymy' would serve any purpose in a synchronic description of the lexicon. Secondly, the phrase fails to convey the problematic subregular character of the relations involved. Our thesis is that 'regular polysemy', while often less regular than lexical syntax or inflectional morphology, is, like them, subregular and is appropriately described using the same formal machinery. Arguably, polysemy is simply null derivation.

Until very recently, and with odd exceptions, polysemous relations have received little attention in the literature on the lexicon. We suspect that this neglect has a lot to do with the fact that linguists have not had plausible machinery for dealing with subregular phenomena. That situation has changed over the last few years in the wake of the development of a number of lexical description languages by computational linguists and their appli-

cation to, *inter alia*, the representation of polysemy (see Kilgarriff (1992) and Copestake (1993) for full discussion and references to the literature). In this chapter we will use the lexical representation language DATR to represent polysemous relations such as those evidenced in the extract from the *Random House Dictionary* cited above. In section 2, we provide an informal introduction to the DATR language and, in section 3, we go on to define a lexicon fragment that illustrates how DATR can be used to express the kind of subregular generalisations that are pervasive over lexical senses.

2 Lexical representation

Evans & Gazdar (1989a, 1989b) give formal presentations of a semantics, and a theory of inference, for DATR, a lexical knowledge representation language. The goal of the DATR work was to define (and implement) a simple language that (i) has the necessary expressive power to encode the lexical entries presupposed by contemporary work in the unification grammar tradition, (ii) can express all the evident generalisations about the information implicit in those entries, (iii) embodies an explicit theory of inference, (iv) is readily implementable, and (v) has an explicit declarative semantics. DATR defines networks allowing multiple default inheritance of information through links typed by attribute paths. This typing provides the basis for a 'most specific path wins' default inheritance principle. The language is functional, that is, it defines a mapping which assigns unique values to node attribute-path pairs. Recovery of these values is deterministic – no search is involved.

In addition to punctuation, DATR contains two kinds of primitive object: nodes, which, by convention, are always marked with an initial capital letter, and atoms, which appear in lower case. When atoms appear between angle brackets they are referred to as attributes, and sequences of attributes are known as paths. Atoms that appear elsewhere are called values. DATR theories (in the logician's sense of theory – a set of axioms from which theorems may be derived) consist of a set of equations.[1] Every DATR equation has a pair of a node and a path as its left-hand side (LHS). The simplest kind of DATR equation simply has zero or more values on the RHS.

```
Node:Path==Values.
```

Here are some examples:

```
ENTITY:<collocates>==.
FIBRE:<genus>==fibre.
```

```
Jersey:<alt garment collocates>==football.
Felt:<made-of>==matted fibre.
```

The first example says, unsurprisingly, that the set of collocates associated with the ENTITY node is empty; the second that the genus attribute for the FIBRE node has the value fibre; the third that the collocates list for the alternant garment sense of the Jersey lexeme consists of football; and the fourth that the made-of attribute for the lexeme Felt equates to the two-element value sequence matted fibre.[2] (Sequences of) values are the principal 'results' of a DATR theory: the typical operation involves proving a theorem that will provide the value sequence associated with a given node/path pair.

More generally, the RHS of equations can be values, inheritance descriptors (quoted or unquoted) or (possibly empty) sequences of values and/or descriptors. Inheritance descriptors specify where the required values can be inherited from, and sequences allow arbitrary lists of atoms to be built as values. Inheritance descriptors come in several forms with two dimensions of variation. The unquoted/quoted distinction specifies whether the inheritance context is local (the most recent context employed) or global (in simple cases, the initial context employed). Once the context is established, the descriptor specifies a new node, a new path or both to be used to determine the inherited value. We will give examples of most of the syntactic possibilities below.

A second type of simple DATR equation uses a node/path pair as its inheritance descriptor and thus has the following form,

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

which says that the value of Path1 at Node1 is to be found by getting the value of Path2 at Node2. A variant of this second DATR statement type uses a Node as an inheritance descriptor,

```
Node1:Path==Node2.
```

but this can be seen as simply an abbreviation for

```
Node1:Path==Node2:Path.
```

And such a statement tells us that the value of Path at Node1 is to be sought at Node2. Here are some examples of this frequently used statement type:

```
Flax:<>==CROP.
CROP:<>==PLANT.
```

```
PLANT:<>==ENTITY.
Canvas:<alternants>==ARTEFACT .
```

Here, the first three statements tell us that the lexeme Flax inherits values from corresponding paths at the CROP node; that CROP inherits values from PLANT and that the latter inherits from ENTITY. The final example says that the value for the <alternants> path of the lexeme Canvas is to be sought from the corresponding path at the ARTEFACT node.

Another type of simple DATR equation uses a path as an inheritance descriptor and has the form

```
Node:Path1==Path2.
```

which can be seen as no more than an abbreviation for

```
Node:Path1==Node:Path2.
```

and this says that the value of Path1 at Node is to be sought by finding the value of Path2 at Node.

```
ENTITY:<alt genus word>==<word>.
```

This example can be glossed as saying that the value for the <alt genus word> path at the ENTITY node is to be obtained by finding the value for the attribute word at the same node.

A third type of DATR equation turns out to be invaluable in DATR analyses, but is conceptually rather different from the equation types that we have just introduced. It uses a quoted path as the inheritance descriptor and looks like this.

```
Node:Path1=="Path".
```

Its interpretation involves a global reference to the node from which one's query originated – at the risk of oversimplification, we can paraphrase it as saying that the value of Path1 at Node is whatever the value of Path2 is at the original query node. The following is a characteristic example of the use of this quoted path form:

```
YARN:<made-of>=="<source>".
```

This says, in effect, that yarns are made of whatever substance is identified as the value of the source attribute at the node from which the query originated. If we had said,

```
YARN:<made-of>==<source>.
```

then we would most likely get no value at all, since this is just equivalent to

```
YARN:<made-of>==YARN:<source>.
```

and <source> is unlikely to be defined at the (abstract) YARN node, nor, in the absence of a global reference mechanism, could it be usefully defined so as to return exactly the source of whatever particular lexeme we happened to be interested in.

There are two other basic DATR equation types, which use quoted nodes and quoted node/path pairs as their descriptors:

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

Their semantics is subtle and, although our treatment of the flax/linen relation makes use of an instance of the latter, they will not be discussed further here.

One further aspect of DATR is illustrated in the fragment that we present in section 3. This is the possibility of having sequences of values and/or inheritance descriptors on the right-hand side of equations. Thus all of the following are perfectly legal, along with an infinity of others:

```
Node1:Path1==Value1.
Node1:Path1==Value1 Path2.
Node1:Path1==Path2 Value1.
Node1:Path1==Value1 Node2.
Node1:Path1==Node2 Value1.
Node1:Path1==Value1 Value2.
Node1:Path1==Value1 "Path2".
Node1:Path1==Node2:Path2 Value1.
Node1:Path1=="Path3" Value1 Node2:Path2.
Node1:Path1==Node2:Path2 Value1 "Path3" Value2.
```

Here are some concrete examples taken from the fragment with which the present chapter deals.

```
PLANT:<collocates>==grow ENTITY.
Cotton:<alt fibre collocates>==wool FIBRE:<collocates>.
YARN:<made-of>=="<source>" fibre.
CROP:<alternants>==fibre seed PLANT.
```

These behave as the notation would lead you to expect – instead of getting atomic values from them, one gets value sequences.

Now that we have presented the syntax of the DATR language, we will turn briefly to a couple of key rules of inference that apply in the language. The first rule implements local inheritance, and uses the following addi-

tional metanotational device: the expression *E0{E2/E1}* denotes the result of substituting *E2* for all occurrences of *E1* in *E0*.

(LOC) `Node2:Path2==A.`
`Node1:Path1==B.`

`Node1:Path1==B{A/Node2:Path2}.`

Rule LOC says that if we have a theorem `Node1:Path1==B.`, where B contains `Node2:Path2` as a subexpression, and we also have a theorem `Node2:Path2==A.`, then we can derive a theorem in which all occurrences of `Node2:Path2` in B are replaced by `A`. In the simplest case, this means that we can interpret a statement of the form

`Node1:Path1==Node2:Path2.`

as an inheritance specification meaning 'the value of `Path1` at `Node1` is inherited from `Path2` at `Node2`'. So, for example, from:

`Flax:<word>==flax.`
`Linen:<source>==Flax:<word>.`

one can infer:

`Linen:<source>==flax.`

LOC can also handle inheritance for node and path descriptors, in view of the following, already noted, equivalences:

`Node1:Path1==Node2.` *is equivalent to*
`Node1:Path1==Node2:Path1.`

`Node1:Path==Path2.` *is equivalent to*
`Node1:Path1==Node1:Path2.`

Rule LOC implements a local notion of inheritance in the sense that the new node or path specifications are interpreted in the current local context. The second inference rule considered here implements a non-local notion of inheritance: quoted paths specify paths which are to be interpreted in the context of the node in which the original query was made (the global context), rather than the current context.[3]

(GLO) `Node1:Path2==Value.`
`Node1:Path1==A.`

`Node1:Path1==A{Value/"Path2"}.`

To see how the operation of the GLO rule differs from LOC, consider the following theory:

```
YARN:
  <source>==undefined
  <made-of>==<source>.

Linen:
  <source>==flax
  <made-of>==YARN.
```

The intention here is that the YARN node expresses the generalisation that yarns are made of whatever the source material is for the instance of yarn involved. But the theory as stated fails to express this intention since we can derive the following unwanted theorem:

```
Linen:
  <made-of>==undefined.
```

To achieve the desired result, we must modify the theory as follows:

```
YARN:
  <source>==undefined
  <made-of>==“<source>”.

Linen:
  <source>==flax
  <made-of>==YARN.
```

With this change, the GLO inference rule allows us to derive the theorem we want:

```
Linen:
  <made-of>==flax.
```

The proof is as follows:

```
1 Linen:(made-of)==YARN.        <given>
2 YARN:(made-of)==“<source>”.   <given>
3 Linen:(made-of)==“<source>”. <LOC on 1 and 2>
4 Linen:(source)==flax.         <given>
5 Linen:(made-of)==flax.        <GLO on 3 and 4>
```

For completeness, we state below the only other inference rule that is relevant to the fragment in this chapter, but we will not discuss it.

```
(QNP) Node2:Path2==Value.
      Node1:Path1==A.
      ---------------------------------
      Node1:Path1==A{Value/“Node2:Path2”}.
```

In addition to the conventional inference described above, DATR has a non-monotonic notion of inference by default: each equation about some node/path combination implicitly determines additional equations about all the extensions to the path at that node for which no more specific equation exists in the theory.

To characterise this notion of default inference, we need some auxiliary definitions. The expression Path1^Path2 denotes the path formed by concatenating the two paths. And we say that Path3 is an extension of Path1 if and only if there is a Path2 such that Path3=Path1^Path 2, and that Path3 is a strict extension of Path1 if and only if Path2 is non-empty. We also use the ^ operator to denote extension of all the paths in a DATR equation, as in the following examples:

```
If    S       is   N:<a>==v.
then  S^<bc>  is   N:<a b c>==v.

If    S       is   N1:<a>==N2:<bc>.
then  S^<bc>  is   N1:<a b c>==N2:<b c b c>.

If    S       is   N1:<a>==“<>”.
then  S^<bc>  is   N1:<a b c>==“<b c>”.
```

Given an equation S, we define the root of S to be the node/path expression appearing to the left of the equality in S (e.g. the root of Node:Path== Value. is Node:Path). Given a set of equations T, a Node and a Path, we say Node:Path is specified in T if and only if T contains an equation S whose root is Node:Path.

Let Node1:Path1 and Node1:Path2 be such that Node1:Path1 is specified in T. We say Node1:Path2 is connected to Node1:Path1 (relative to T) if and only if:

1 Path2 is an extension of Path1;
2 there is no strict extension Path3 of Path1 of which Path2 is an extension such that Node1:Path3 is specified in T.

So Node1:Path2 is connected to Node1:Path1 if and only if Path1 is the maximal subpath of Path2 that is specified (with Node1) in T.

Given a set of equations T, we define the path closure of T to be:

{S^Q | S is an equation in T, with root Node:Path, and
Node:Path^Q is connected to Node:Path}

It is clear from these definitions that any Node:Path is connected to itself and thus that T is always a subset of the path closure of T. The path closure contains all those theorems which can be inferred by default from T. The

operation of path closure is non-monotonic: if we add more equations to our original theory, some of our derived equations may cease to be true. The two forms of inference in DATR are combined by taking the path closure of a theory first, and then applying the inference rules to the result.

To illustrate path closure, consider the following example theory:

```
CROP:
  <>==PLANT
  <syntax>==MASS-NOUN:<>
  <alt fibre>==FIBRE:<>.

Cotton:
  <>==CROP
  <word>==cotton.
```

We can infer by default the following theorems, among others:

```
CROP:
  <genus>==PLANT:<genus>
  <made-of>==PLANT:<made-of>
  <artefact>==PLANT:<artefact>
  <collocates>==PLANT:<collocates>
  <syntax cat>==MASS-NOUN:<cat>
  <syntax count>==MASS-NOUN:<count>
  <syntax concrete>==MASS-NOUN:<concrete>
  <alt fibre alt yarn>==FIBRE:<alt yarn>
  <alt fibre alt yarn alt fabric>==FIBRE:<alt yarn alt fabric>.

Cotton:
  <word form>==cotton
  <genus>==CROP:<genus>
  <word root form>==cotton
  <made-of>==CROP:<made-of>
  <artefact>==CROP:<artefact>
  <collocates>==CROP:<collocates>
  <syntax cat>==CROP:<syntax cat>
  <syntax count>==CROP:<syntax count>
  <syntax concrete>==CROP:<syntax concrete>
  <alt fibre alt yarn>==CROP:<alt fibre alt yarn>
  <alt fibre alt yarn alt fabric>==CROP:<alt fibre alt yarn alt
    fabric>.
```

Note the way in which equations that have paths on their RHS are also extended by the subpath used to extend the LHS. This characteristic of the DATR language plays a crucial role in our treatment of alternant meanings in the rest of this chapter.