UNDERDETERMINED FEATURE SETS

Allan Ramsay

1987

Cognitive Studies Research Papers

Serial No. CSRP.068

The University of Sussex
Cognitive Studies Programme
School of Social Sciences
Falmer, Brighton BN1 9QN

UNDERDETERMINED FEATURE SETS

Allan Ramsay,
Cognitive Studies Program, University of Sussex
Falmer BN1 9QN ENGLAND

Abstract
Many words in English have underdetermined syntactic features and subcategorisation frames. Dealing with this by providing separate lexical entries for each variant of each word leads to unacceptably inefficient parsing algorithms. We present here a unification algorithm designed for such underspecified feature sets.

Keywords: parsing, lexical ambiguity, underspecification. unification

The problem

Many English words are ambiguous with respect to lexical category, a fact which can have significant effects on simple-minded parsing algorithms. This is particularly apparent when we start to use complex feature sets, rather than simple labels, as in the recent work on "unification grammars* (Gazdar et al. 1985, Bresnan & Kaplan 1982, Kay 1985, Pereira & Warren 1981). The problem intensifies further when we try to compress the rule set making up our grammar by allowing lexical entries to contain information which can be used to flesh out and constrain a minimal set of rather skeletal rules. For reasons of space we consider two particular cases in this paper, though we believe that what we say has in fact very wide application indeed.

The two special cases are for auxiliary sequences and for verbs taking sentential complements. We would like to deal with these by allowing one very simple rule for each, and then specifying in the lexical entry for each auxiliary or complement-taking verb how the rule should be applied. We would like, in essence, the following two rules:

```
(1) VP -> aux VP

(2) VP -> verb S
```

about the nature of the various components and the constraints that are imposed on them. The facilities of unification grammar enable us to do this by providing complex feature trees in place of the atomic symbols in (1) and (2), with 'logical variables' or some similar notion employed to propagate constraints between the components of a rule. In the notation used in this paper, we can replace (1) and (2) with the following pair of rules:

```
(3) vseq=(subject:SUBJ, head:HEAD, auxseq:AUX,
          mv=(v=(vform:AUXFORM, trans:TRANS, root:ROOT)),
          mood:M, slash=null) →
    vseq=(subject:SUBJ, head:HEAD, mood:M,
          mv=(v=(trans=(aux:AUXFORM), vform=(voice=active)))):AUX,
    v=(vform:AUXFORM, trans:TRANS, root:ROOT)

(4) VP=(pred:PRED, comp=VP:COMP, slash:SLASH) →
    vseq=(subject:SUBJ,
          mv=(v=(vform=(voice=active),
                 trans=(comp=(compsubj:COMPSUBJ,
                              finite:FINITE))))):PRED,
    VP=(pred=(subject:COMPSUBJ,
              head=(v=(vform=(finite:FINITE)))), slash:SLASH):COMP
```

The essential points about the notation are that expressions like `f1=(f2=(f3, f4), f5)` are specifications of complex features, where `f2` and `f5` are sub-features of `f1`, and `f3` and `f4` are sub-features of `f2`; and that the role of logical variables is played by names separated from the feature specification which they name by a colon, as in `f1=(f2:NAME)`. No significance should be read into the use of upper and lower cases: something is a name if and only if it follows a colon. This notation is similar to that used in Warren and Pereira's definite clause grammar, and differs from some of the later unification grammars in that the constraints must be specified explicitly by the grammar writer - the grammar does not make use of any universal principles about constraints on features.

The essential point about the rules themselves is that, apart from a couple of minor technical changes, they are the same as (1) and (2), except that they rely on the lexical entries for the words involved to fill in the details about what is actually replacement of the distinction between a sentence and a verb phrase by the use of a feature, subject, which records whether the verb phrase has been allocated a subject and if so what it is. We use vseq as a category on the grounds that the complement-taking behaviour of a verb depends on the local context in which it appears, notably on the particular configuration of auxiliaries that precede it. and that it is therefore preferable to consider the verb and auxiliary sequence as a unit when trying to describe the way it fits into a verb phrase or sentence. The notion that a sentence is simply a VP with the feature subject specified is not original to this paper, and we will say no more about it here other than to note that it enables us to use one rule for sentential complements rather than two - the difference between verbs which require a full sentence and ones which just require a VP is encoded in the value for the feature compsubj.

The major change is the detail provided for the feature specifications. (3), for instance, now says that the auxiliary which is the main verb of the initial verb sequence must have as the value of the feature trans a sub-specification whose major label is aux. but which is itself constrained to match the form of the following verb. Consider the following entry in the lexicon for a particular auxiliary!

```
(5)  am  v«(trans-(aux-(finite-(nonfinite«(participle-past)),
                        voice-passive, agree, case)),
           root-be,
           vform-(finite-(tensed-present), case-subject, voi ce-acti ve,
                  agree«(other-fi rst)))
```

This contains all the information needed to use (3) for handling passivizing uses of *am.* The sub-features of aux provide all the required constraints on the form of the following verb, namely that it must be a passive, past participle form, with a occurring items, *am,* for instance also occurs with present participle forms, and with nominal complements, *see* accepts a wide, but not universal, range of sentential complements (*/ saw him do it, I saw him doing it, I can see he does it,* but not / *saw him to do it)* as well as ordinary objects (/ *saw it)* and even no complement at all (/ *see).* The obvious solution is to provide separate lexical entries for each form, for instance (6) and (7) as alternative forms for *am.*

```
(6)  am  v«(trans-(aux«(finite»(nonfinite«(participle-present)),
                         voice-active, agree, case)),
          root-be,
          vform-(finite-(tensed-present), case-subject, voice-active,
                 agree-(other-first)))

(7)  am  v-(trans-copula),
          root-be,
          vform*(finite-(tensed-present), case*subject, voice-active,
                 agree-(other-first)))
```

This is an obvious and easy solution. It is also a very poor one. The problem is that each lexical entry will start a new path of analysis when we come to try to parse text containing such words. This leads to wasted effort, since at most one of the options is going to turn out to have been useful at the end of the analysis. And it leads to highly repetitive work. Suppose for instance that we wanted to analyse a sentence which started with the string / *would have seen* .... There are a number of ways we might go about this task, but it is very hard to see how we could avoid *either* combining identical analyses of the initial sequences / *would have* or *would have* with all the variants of *see, or* misguidedly trying out a passive analysis of *seen.* Similar problems occur in a large number of different situations, and the moral is clear: it is a bad idea to have separate lexical entries for minor variants in sub-categorisation.

Disjunctive feature sets

It would seem better to have a single lexical entry for words like *am* and *see,* and to deal with the uncertainties by allowing optional values for features. Thus the

```
(8) am v=(trans=(aux=(finite=(nonfinite=(participle=past)),
                          voice=passive, agree, case),
                  aux=(finite=(nonfinite=(participle=present)),
                          voice=active, agree, case),
                  copula),
          root=be,
          vform=(finite=(tensed=present), case=subject, voice=active,
                  agree=(other=first)))
```

This seems to be what we want - a single entry with all the options about its complement or following verb sequence present. The problem now is that it is no longer clear how to unify constituents of rules with putative constituents of an analysis. The standard notion of unification does not seem suitable, since the presence of different values for the same feature (as in the two appearances of `aux` in (8)) would seem to lead to an undefined or ill-specified unifier. The essential point of the current paper is that we give a coherent interpretation of what unification means when the objects being unified contain disjunctive feature sets (feature sets where features may have alternative values). For this we need the following definitions:

Definition 1: a pattern 'accepts' a target if and only if for every distinct feature in the pattern there is a corresponding entry in the target such that either one of them has no sub-features, or every sub-feature of the pattern accepts some sub-feature of the target. In other words, we take a liberal view of fine detail - if it's present in the pattern but not the target, or present in the target but not the pattern, then we accept the match on the grounds that we have no evidence to rule it out. But if both elements of the match do contain further detail, then the target must at least contain everything the pattern does.

Definition 2: the acceptance of a target by a pattern provides a 'binding' for the names that appear in the pattern. The binding for a name is the set of all values which were accepted for the feature the name is associated with.

of rules are taken as patterns their function is clearly to constrain what items may occur in adjacent positions. However, if a binding has been established by matching an under-specified feature against a pattern element, then any restriction established by that binding may contain alternatives for some features. In such a case, we have to view the pattern as a conjunction of disjunctions - every *distinct* feature must be matched by some element of the target, but not every alternative must be matched. To see what all this means in concrete terms, consider what happens when rule (3) is triggered in a bottom-up left→right parser by a verb sequence containing *am* as its sole constituent. (9) below is the description of such a verb sequence with inessential detail edited out to save space:

```
(9) (vseq=(head=<identical to mv, so edited for space>,
          mood=decl,
          mv=(v=(root=be,
                 trans=(aux=(agree,
                             case,
                             finite=(nonfinite=(participle=past)),
                             voice=passive),
                        aux=(agree,
                             case,
                             finite=(nonfinite=(participle=present)),
                             voice=active),
                        copula),
                 vform=(agree=(other=first),
                        case=subject,
                        finite=(tensed=present),
                        voice=active))),
          subject=null,
          vform=<detail edited>)))
```

This object will clearly be accepted by the first constituent of the right-hand side of (3), with bindings for SUBJ, HEAD, M, AUXFORM and AUX becoming established. When the next component of the rule is restricted by these bindings, we find that the rule requires to be completed by something which would be accepted by:

```
(10) (v=(root:ROOT, trans:TR,
        vform=(agree, case,
               finite=(nonfinite=(participle=past)),
               voice=passive):AUXFORM,
        vform=(agree, case,
               finite=(nonfinite=(participle=present)),
```

This is a pattern, but is underspecified with respect to vform. It will accept either passive-voiced past participles, as in *am eaten*, or active-voiced present participles, as in *am eating*. (10) is slightly more constrained than the final component of (3), which simply required vform to have *some* value. When a particular lexical entry, such as *eating* or *eaten*, is offered for acceptance by (10), the result of the match will bind AUXFORM to one or other of the candidates, so that the restriction of the left hand side of the rule will be marked as active or passive as appropriate.

Conclusions

By providing an interpretation for unification when applied to disjunctive feature sets, we can develop very concise rule sets whilst avoiding the introduction of large numbers of lexical entries. Given the number of words that have numerous marginally different subcategorisation frames, this should lead to much faster syntactic analysis, since we are now considering fewer rules and fewer potential instantiations of each rule. We do, however, have to pay a price, namely that we have a more complex notion of unification. The current implementation takes advantage of the fact that the order in which features appear in rules and in analyses is totally unimportant - feature sets are *sets*, not ordered lists. Since the order has no significance to the grammar, we are free to impose an order which will speed up the unification process if we can find one. By ensuring that all elements of a feature set are ordered by the alphabetic order of their main features, the unification algorithm can be made to run in O(n) rather than O(n**2) time. This is a very simple trick, but it does have the important consequence that the complex unification algorithm we are using does not greatly degrade performance.

It is notoriously difficult to compare the performance of programs for syntactic analysis - there are so many factors which can affect performance, from the speed is a far more serious problem for parsing strategies than is generally acknowledged, so that mechanisms for lessening its impact must be a priority. The status of the work described here is that we had a bottom-up chart parser for a unification grammar, the full details and code of which are given in (Ramsay & Barrett 1987), without disjunctive feature sets. This initial implementation performs very satisfactorily with a non-trivial grammar (order of a second per word for ten word sentences, with a grammar contained rules for unbounded dependencies via slash categories, conjunction, various forms of sentential complement, relative clauses including 'whiz-deleted' forms, ...) The version with disjunctive feature sets has a grammar which covers more cases with fewer rules, and performs rather better, especially in the face of words with underdetermined sub-categorisation frames. Some of the performance is due to the speed of the machine (a SUN-3 workstation), some to the basic design of the chart parser, and some to the denseness of representation achieved by using disjunctive feature sets.

References

Bresnan, J.W. & Kaplan, R. (1982): Lexical functional grammar; a formal system for grammatical representation. In: *The mental representation of grammatical relations*. Bresnan, J.W., ed. Cambridge, Mass.

Gazdar, G., Klein, E., Pullum, G.K. & Sag, I.A (1985): *Generalised phrase structure grammar*. Oxford.

Kay, M. (1985): Parsing in functional unification grammar. In: *Natural language parsing*. Dowty, D.R., Karttunen L. & Zwicky A.M., eds., Cambridge, 251-278.

Pereira, F.C.N. & Warren, D.H.D. (1980): Definite clause grammars for language analysis-a survey of the formalism and a comparison with augmented transition networks. In: *Artificial Intelligence* 13, 231-278.

Ramsay, A.M. & Barrett, M.R. (1987): *AI in practice: examples in POP-11*, Chichester.