Delayed Evaluation of Lexical Rules

Gertjan van Noord and Gosse Bouma
Alfa-Informatica and BCN
Postbus 716 NL 9700 AS Groningen
vannoord,gosse@let.rug.nl

ABSTRACT

1 Introduction

In the paper we review the proposals by [4, 1, 2, 5] in which a number of puzzling phenomena in Dutch syntax are solved using recursive 'lexical rules'. Lexical rules are modelled by *recursive constraints*. The processing of such recursive constraints crucially depends on delayed evaluation techniques.

Historically, lexical rules were not allowed to apply recursively, because it was unclear how such recursive applications were to be implemented. For example, if the effect of all lexical rules is compiled off-line, recursive lexical rules cannot be applied because an infinite lexicon might result. An alternative approach in which lexical rules are treated as non-branching syntax rules gives rise to a similar problem during parsing: non-branching derivations of unbounded length can easily surface.

The basic insight behind our implementation was that the problem is less severe: typically both the input *and* the output of the lexical rule applications is known. For example, consider a lexical rule that adds adjuncts to a subcat list. Given that there can be an unbounded number of adjuncts for any given verb, such a lexical rule must apply recursively. This does not neccessarily lead to problems, because, of course, in any given sentence there is a given number of adjuncts which determines the number of times the lexical rule can be applied. Using insights from logic programming (delayed evalution of complex constraints) it becomes feasible to implement this intuition.

In our previous work, complex constraints were produced (on the basis of a number of lexical rules) *by hand*, for two reasons. Firstly this enabled the use of delayed evaluation. Secondly, this allowed control to be added concerning the interaction of lexical rules (for example: to state that a lexical rule applies obligatorily, to reduce the number of spurious ambiguities, etc.).

Here we show that such complex constraints can be derived automatically. Thus, on the basis of a set of lexical rules (in a traditional format) an equivalent definite-clause program can be constructed in which simple annotations are provided to enable delayed evaluation. This result generalizes our previous work, and solves the first problem.

In order to solve the second problem, we allow an (optional) finite-state control over the applications of lexical rules. This can be incorporated in the compilation straightforwardly.

2 Lexical Rules as Binary Predicates

For concreteness we will assume that our grammar is a Definite Clause Grammar, in which lexical entries are introduced with the special predicate *lexicon(Word,Cat)*. This predicate in turn is defined as follows [1]:

lexicon(Word,Cat) :- stem(Word,Cat0), lexical_rules(Cat0,Cat).

The *lexical_rules* predicate is true if the underlying category, *Cat0*, can be related to the surface category, *Cat*, by the lexical rules machinery.

Lexical rules themselves are thought of as binary predicates that relate an 'incoming' category with an 'outgoing' category. Given that lexical rules supposedly can apply freely, the *lexical_rules/2* predicate thus reflects the reflexive and transitive closure of these lexical rules:[2]

lexical_rules(C,C).
lexical_rules(Lex,Surf) :- lr(Mid,Surf), lexical_rules(Lex,Mid).

[1] For simplicity we do not treat morphological alternations.

[2] In this paper, we will not be concerned with the way in which default information percolates between the 'in' and 'out' categories. We thus assume that all equations are explicitly defined.

3 Delayed Evaluation of Lexical Rules

This definition, as it stands, is not very helpful in applying delayed evaluation techniques. For example, if we delay the application of certain specific lexical rules, then the recursive predicate *lexical_rules/2* can be applied 'ad inf'. On the other hand, if we delay the application of *lexical_rules/2* itself, then we have no idea under what conditions application of this predicate should be allowed.

If we assume that for each lexical rule we know the conditions under which it can be fired, then we can solve this problem by automatic program transformations. The details of this transformation will be presented in the final version of the paper. The transformation is exemplified in the figures 1 and 2. We use the SICStus Prolog predicate `when(Cond,Goal)` to indicate under which conditions `Cond` a goal `Goal` can be evaluated.

```
lexical_rules(C,C).
lexical_rules(Lex,Surf) :-
    lr(Mid,Surf),lexical_rules(Lex,Mid).

lr( v(Pref*Post,Slash), v(Pref*[adv|Post],Slash) ).          % adjunct

lr( v(Pref*[Topic|Post],Slash), v(Pref*Post,[Topic|Slash]) ). % topic

lr( v(Sc,Slash), v([np|Sc],Slash)).                           % subject
```

Figure 1: Program before transformation, consisting of three lexical rules. The DCG formalism is extended with the 'append' constraint. The constraint $A * B$ stands for the value C such that the (standard) relation $append(A, B, C)$ is true. This relation is also evaluated in a lazy way.

```
lexical_rules(C,C).
lexical_rules(Lex,Surf) :- lr1(Surf,Lex).

lr1(v(Sc,Slash),Lex) :- when(nonvar(Sc),lr_adjunct(v(Sc,Slash),Lex)).
lr1(v(Sc,Slash),Lex) :- when(nonvar(Slash),lr_topic(v(Sc,Slash),Lex)).
lr1(v(Sc,Slash),Lex) :- when(nonvar(Sc),lr_subject(v(Sc,Slash),Lex)).

lr_adjunct( v(Pref*[adv|Post],Slash),Lex) :-                  % adjunct
    lexical_rules(Lex,v(Pref*Post,Slash)).

lr_topic( v(Pref*Post,[Topic|Slash]),Lex):-                   % topic
    lexical_rules(Lex,v(Pref*[Topic|Post],Slash)).

lr_subject(v([np|Sc],Slash),Lex):-                            % subject
    lexical_rules(Lex,v(Sc,Slash)).
```

Figure 2: Program after transformation.

4 Further issues

In the final version of the paper we will discuss how we can implement fine-grained control over the application of lexical rules. We allow regular expressions over lexical rule names. Implementing such a control extension is trivial. On the other hand, such a control extension seems to be vital for practical applications of the methods described, in order to limit spurious ambiguities, ad-hoc features, and the chances that the lexical rule component is not 'well-behaved', leading to non-termination (given the result in [3] it is not surprising that such cases exist).

References

[1] Gosse Bouma and Gertjan van Noord. Constraint-based categorial grammar. In *31th Annual Meeting of the Association for Computational Linguistics*, New Mexico, 1994.

[2] Gosse Bouma and Gertjan van Noord. A lexicalist account of the dutch verb cluster. In Gosse Bouma and Gertjan van Noord, editors, *CLIN IV, Papers from the Fourth Clin Meeting*, Groningen, 1994.

[3] Bob Carpenter. The generative power of categorial grammars and head-driven phrase structure grammars with lexical rules. *Computational Linguistics*, 17(3):301–313, 1991.

[4] Gertjan van Noord and Gosse Bouma. Adjuncts and the processing of lexical rules. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING)*, Kyoto, 1994.

[5] Gertjan van Noord and Gosse Bouma. Dutch verb clustering without verb clusters, 1995. Ms. Cf. http://www.let.rug.nl/~vannoord/papers/.