QPATR and Constraint Threading

James Kilbury
Seminar für Allgemeine Sprachwissenschaft
Universität Düsseldorf, Universitätsstr. 1
D-4000 Düsseldorf 1, Fed. Rep. of Germany
e-mail: Kilbury@DD0RUD81.BITNET

Abstract

QPATR is an MS-DOS Arity/PROLOG implementation of the PATR-II formalism for unification grammar. The formalism has been extended to include the constraints of LFG as well as negation and disjunction, which are implemented with the disjunction and negation-as-failure of PROLOG itself. A technique of *constraint threading* is employed to collect negative and constraining conditions in PROLOG difference lists. The parser of QPATR uses a left-corner algorithm for context-free grammars and includes a facility for identifying new lexical items in input on the basis of contextual information.

1 Introduction

QPATR ("Quick PATR") is an MS-DOS Arity/PROLOG implementation of the PATR-II formalism (cf Shieber et al. 1983, Shieber 1986) with certain logical extensions. The name was chosen to reflect the fact that the prototype system was developed in a short period of time but nevertheless runs quickly enough for practical use. QPATR was developed at the University of Düsseldorf within the research project "Simulation of Lexical Acquisition", which is funded by the Deutsche Forschungsgemeinschaft.

In contrast to most existing PATR implementations such as D-PATR (cf Karttunen 1986a, 1986b), QPATR runs under MS-DOS and thus makes minimal hardware demands. Like ProP (cf Carpenter 1989) QPATR is implemented in PROLOG but uses both the negation and disjunction of PROLOG in the extended PATR formalism; moreover, it employs a left-corner parser with a "linking" relation and PROLOG backtracking rather than a pure bottom-up chart parser.

The system comprises the following components: (1) grammar compiler, (2) unification, (3) left-corner parser, (4) lexical look-up, (5) input/output, (6) testing off-line input, and (7) tracing. The grammar compiler (1) transforms syntax rules and lexical entries from their external notation to an internal form; at the same time partial *feature structure matrices* (FSMs) are constructed and the linking relation (see below) is constructed. The unification package (2) uses techniques introduced by Eisele and Dörre (1986) and described by Gazdar and Mellish (1989) to implement the unification of FSMs with the term unification of PROLOG. A facility of *prediction* is included in the input/output package that allows new lexical items in input to be identified on the basis of contextual information. While QPATR uses a full-form lexicon at present, a package for morphological analysis is being developed.

Since QPATR is distributed in a compiled version, knowledge of PROLOG is only needed in order to write *macros* (see below) but not to write grammars or to run the system. Thus, QPATR can also be used in instruction with students who have no background in PROLOG programming.

2 Descriptions of FSMs

The formalism of PATR-II has been adopted for QPATR and will not be introduced here. As presented by Shieber (1986: 21) rules consist of a context-free skeleton introducing variables for FSMs and a conjunction of path equations that describe the FSMs, e.g.:

$$
\begin{array}{l}
X_0 \ \text{-->}\ X_1 \ X_2 \\
\qquad \langle X_0 \text{ cat}\rangle = \text{s} \\
\qquad \langle X_1 \text{ cat}\rangle = \text{np} \\
\qquad \langle X_2 \text{ cat}\rangle = \text{vp} \\
\qquad \langle X_0 \text{ head}\rangle = \langle X_2 \text{ head}\rangle \\
\qquad \langle X_0 \text{ head subject}\rangle = \langle X_1 \text{ head}\rangle
\end{array}
$$

where *cat*, *head*, and *subject* are attributes. Such path equations are written with "*=" in QPATR, which is implemented with the normal ("destructive") PROLOG unification. Furthermore, QPATR provides for pseudo-constraints written with "*==" in the path equations, which capture the expressiveness of constraining schemata in LFG (cf Kaplan/Bresnan 1982: 213) and allow the grammar writer to specify that some attribute must *not* receive a value unifiable with the indicated value. These are implemented with the "==" unification of PROLOG.

FSMs are described in QPATR with a logic generally based on that developed by Kasper and Rounds (1986). The presentation of the logical description language here is parallel to that of Carpenter (1989).

Atomic *well-formed formulas* (wffs) of this logic consist of the two types of equations just introduced as well as *macro* heads (see below); heads of macros defined in terms of constraints are prefixed with the operator "@" in atomic wffs. Equations contain two *designators*, which are atoms or FSM variables, implemented with PROLOG atoms and variables, respectively, or else paths. The latter are defined recursively and may contain atoms or paths as attribute expressions. The evaluation of embedded paths must yield an atom.

All derived wffs of the logic are built from atomic descriptions with conjunction ",", disjunction ";", and negation "not"; parentheses may be simplified in the customary manner. Disjunction and negation are not directly reflected in the FSMs generated in QPATR. Disjunctions are implemented with PROLOG backtracking, while negations are treated like pseudo-constraints, which are executed as tests *after* the complete FSM of an input phrase has been constructed by the parser. The "negation" employed here is thus the *negation-as-failure* of PROLOG.

FSMs themselves are represented internally as a PROLOG list of feature-value pairs with a variable remainder list (cf Eisele/Dörre 1986: 551; Gazdar/Mellish 1989: 228). Since FSMs are described rather than directly represented in the grammar and lexicon, these internal PROLOG representations normally are neither constructed nor seen by the user.

The syntax of the logical description language is defined here in Backus-Naur form:

```
well-formed formula
<wff> ::= <awff> |
          '(' <wff> ',' <wff> ')' |        conjunction
          '(' <wff> ';' <wff> ')' |        disjunction
          '(' 'not' <wff> ')'              negation

atomic wff
<awff> ::= <descr> |
           <cdescr> |
           <macro-head> |              see below
           '(' '@' <macro-head> ')'   constraining macro

FSM description
<descr> ::= '(' <desig> '*=' <desig> ')'

constraining FSM description
<cdescr> ::= '(' <desig> '*==' <desig> ')'

designator
<desig> ::= <atom> | <fsm-variable> | <path>

path
<path> ::= <fsm-variable> '/' <attr-exprs>

attribute expressions
<attr-exprs> ::= <attr-expr> | <attr-expr> '/' <attr-exprs>

attribute expression
<attr-expr> ::= <atom> | '{' <path> '}'
```

3 Macros

Macros (or *templates*; cf Shieber 1986: 51) may be employed in QPATR to reduced redundancy in syntax rules and lexical entries and thereby to capture generalizations. In the present version of QPATR macros are defined as *conjunctions* of other macros and FSM descriptions with "*=" and "*=="; they may not contain disjunctions or negations. Furthermore, macros may not be defined recursively as this would lead to nonterminating loops.

Since macros are ultimately defined in terms of FSM descriptions with "*=" and "*==", which themselves are implemented as executable PROLOG goals, macros are represented in the present QPATR version simply as PROLOG inference rules with a head consisting of the macro name as its predicate and the variables for FSMs referred to as its arguments. This is the only part of the system that requires elementary PROLOG programming in order to write grammars in the formalism.

A special representation language for the definition of macros is being developed and will be included in new versions of QPATR.

4 Rules and Lexical Entries

Syntax rules are indexed with an integer which is used by the linking relation constructed during compilation of the grammar into its internal form (see below). The numbering of rules is arbitrary and need not be consecutive or ordered.

Category descriptions are macro heads. In principle, a single dummy macro name *cat* can be used for all categories so that all information about the FSMs contained in a rule is put in the description wff of the right-hand side; however, the linking relation would then lose its value for the parser. In order to modularize the grammatical description, the wffs of rules and entries may be defined exclusively in terms of macros.

The syntax of rules and lexical entries is defined as follows:

```
rule
<rule> ::= <integer> '#' <cat> '--->' <rhs> '.'

right-hand side
<rhs> ::= <cats> | <cats> '::' <wff>

categories
<cats> ::= <cat> | <cat> ',' <cats>

category
<cat> ::= <macro-head>

lexical entry
entry ::= <atom> 'lex' <lrhs> '.'

lexical right-hand side
<lrhs> ::= <cat> | <cat> '::' <wff>
```

Orthographic word forms are represented as PROLOG atoms.

5 Constraint Threading

By convention, the wffs of rules and lexical entries are written in conjunctive normal form as a list of atomic wffs, disjunctions, and negations. When a rule or entry is compiled the list representing its wff is sorted into lists of atomic wffs (except constraints), disjunctions, and constraints (including negations) whose members are executed as PROLOG goals before, during, and after parsing, respectively. The execution of the atomic wffs without constraints builds partial FSMs which contribute to the information encoded in the linking relation (see below). In their compiled form rules and entries thus contain partial FSMs associated with lists of disjunctions and negations that apply to them.

Disjunctions are executed *during* parsing and make use of the normal backtracking mechanism of PROLOG while constraints and negations are executed *after* parsing to test whether a FSM in fact fulfills all conditions of the original wff. During parsing the constraints and negations contributing to the complete description of the FSM associated with the input must be collected. In order to accomplish this a technique of *constraint threading* is introduced based on the difference lists used by Pereira and Shieber (1987) for gap threading. The PROLOG term associated with a syntactic constituent contains difference lists of constraints associated with the constituent before and after it has been parsed. The first difference list for an entire input phrase is the empty list, while the second is instantiated with the complete list of constraints and negations after parsing is completed.

A complication arises from the fact that constraints and negations may be embedded in disjunctions and that their execution must be deferred. This can be dealt with by "percolating" such embedded constraints up into the difference lists for constraint threading when the disjunction is solved. The following program implements the execution of disjunctions during parsing:

```
% solve_disjunctions(
%     <disjunctions>,<constraints0>,<constraints>)

solve_disjunctions([], C, C).

solve_disjunctions([D|Ds], C0, C) :-
      dsolve(D, C0, C1),
      solve_disjunctions(Ds, C1, C).

dsolve((Wff ; Wffs), C0, C):-
      !, (dsolve(Wff,C0,C) ; dsolve(Wffs,C0,C)).

dsolve((Wff , Wffs), C0, C) :-
      !, dsolve(Wff,C0,C1), dsolve(Wffs,C1,C).

dsolve((not Wff), C, [(not Wff)|C]) :- !.

dsolve((@ Wff), C, [Wff|C]) :- !.

dsolve(Wff, C, C) :- call(Wff).
```

6 The Parser of QPATR

The parser is based on a left-corner algorithm with backtracking for context-free grammars (cf Kilbury 1988 and Pereira/Shieber 1987: 179ff). The efficiency of the parser is improved with top-down filtering in the form of a *linking* relation (cf Pereira/Shieber 1987: 182). This ordinarily is a transitive binary relation over categories represented as PROLOG atoms or terms with atomic category labels as functors. The PATR formalism requires a modified technique since the syntax rules contain FSMs, whose unification is more costly than that of atomic category lables. QPATR therefore uses numbered syntax rules and then defines the filter with a binary relation over the rule indices. If the grammar contains some rules

$i \# F_{i0} \dashrightarrow F_{i1}, \ldots , F_{im}$

$j \# F_{j0} \dashrightarrow F_{j1}, \ldots , F_{jn}$

where the subscripted *F*'s are FSMs, then we have *dlink(i,j)* iff F_{j1} subsumes F_{i0}, i.e. if F_{i0} is an immediate left corner of F_{j0}. Then *link(i,j)* is the reflexive and transitive closure of *dlink(i,j)*.

7 Lexical Prediction

QPATR includes a facility of *prediction* whereby FSMs are proposed for new lexical items encountered in input but not contained in the lexicon. Predictions are made on the basis of contextual information collected during the analysis of input. A fundamental distinction is made between *open* and *closed lexical categories*, and this information must be represented with definitions of corresponding macros in the grammar. These definitions may refer to semantic as well as syntactic categorial information. A prediction is blocked if the proposed FSM does not match an open lexical class or if it is described by an entry already in the lexicon, but FSMs may be constructed for new lexical items having homonyms in the lexicon. The definition of *open* is not used actively to propose an FSM but rather passively to test the admissibility of an FSM already constructed from the context.

References

Carpenter, Bob (1989) *Prop Documentation*. Computational Linguistics Program, Carnegie Mellon University.

Eisele, Andreas / Dörre, Jochen (1986) A Lexical Functional Grammar System in PROLOG, *Proceedings of COLING-86*, 551-3.

Gazdar, Gerald / Mellish, Chris (1989) *Natural Language Processing in PROLOG*. Wokingham, England et al.: Addison-Wesley.

Kaplan, Ronald M. / Bresnan, Joan (1982) Lexical-Functional Grammar: A System for Grammatical Representation, in *The Mental Representation of Grammatical Relations* (Joan Bresnan, ed.). Cambridge, Mass. / London: MIT Press.

Karttunen, Lauri (1986a) D-PATR: A Development Environment for Unification-Based Grammars, *Proceedings of COLING-86*, 74-80.

Karttunen, Lauri (1986b) *D-PATR: A Development Environment for Unification-Based Grammars* (= CSLI Report No. 86-61). Stanford, Calif.: CSLI.

Kasper, Robert T. / Round, William C. (1986) A Logical Semantics for Feature Structures, *Proceedingsof the 24th Annual Conference of the ACL*, 235-242.

Kilbury, James (1988) Parsing with Category Cooccurrence Restrictions, *Proceedings of COLING-88*, 324-327.

Pereira, Fernando C. N. / Shieber, Stuart M. (1987) *Prolog and Natural-Language Analysis* (= CSLI Lecture Notes, 10). Stanford, Calif.: University of Chicago Press.

Shieber, Stuart M. (1986) *An Introduction to Unification-Based Approaches to Grammar* (= CSLI Lecture Notes, 4). Stanford, Calif.: University of Chicago Press.

Shieber, Stuart M. et al. (1983) The Structure and Implementation of PATR-II, *Research on Interactive Acquisition and Use of Knowledge*, 39-93. Menlo Park, Calif.: SRI International