Syntactic pattern recognition in the HLP/PAS system

T. GYIMÓTHY AND J. TOCZKI

Abstract

In this paper a syntactic pattern recognition application of the HLP/PAS system is presented. The system has originally been developed for compiler generation. It can generate both one-pass and multi-pass compilers from attribute grammar specifications. The generated compilers use LL (1) or LALR (1) parsing methods. However, in many cases, patterns can be described only with ambiguous grammars. For this reason the HLP/PAS system was extended with a backtrack parser generator. The generated backtrack parsers use the LL (1) parsing tables to eliminate some of the unnecessary backtracks. Another characteristic of these parsers is that the parsing can be controled by the evaluated attributes. As an illustration, an attribute grammar description is presented for normal ECG waveforms.

key words: attribute grammars, syntactic pattern recognition, attribute evaluation.

1. Introduction

So far there have been several attempts for describing patterns by attribute grammars [8], [9], [11]. It is not surprising as both the context-free and the context-sensitive characteristics of the patterns can be described by attribute grammars. The numerical data of the patterns can conveniently be computed by semantic rules so attribute grammars can create a connection between syntactic and statistic methods of the pattern recognition.

While there are several complete compiler generator systems based on attribute grammars [4], [7], to our knowledge, there is no such a system for pattern recognition tasks. In a complete system a metalanguage is needed for the specification of the attribute grammars. It is practical if the primitives of the patterns can be described as lexical tokens in this metalanguage. The parser of the metalanguage has to check the formal correctness of the specifications e.g. the consistent using of the attributes. The system must generate a parser and an attribute evaluator, too. In contrary to the usual complier generators, in a pattern recognition system, the construction of backtrack parsers is needed. Therefore, the HLP/PAS system [5], [10], which has originally been developed for compiler generation, was extended with a backtrack parser generator. In this paper we give a description of this extended system. In more detail, section 2 gives a short description of the original system, section 3 contains the specification of the ECG grammar. In section 4 the structure of the generated backtrack parsers is described, while section 5 contains some observation about further research of this topics. Finally we give a short summary of the paper.

2. The HLP/PAS system

As it was already mentioned, the HLP/PAS system was originally developed for compiler generation. There are two metalanguages in the system for the lexical and syntactic-semantic descriptions of grammars. The lexical units (tokens) can be defined by regular expressions in the lexical metalanguage. In programming languages the usual tokens are identifiers, numbers etc. In the pattern descriptions the "primitives" [2] are the lexical units. The system generates finite automata to recognize these tokens. The generated lexical analyzer is a procedure of the complete compiler. The specification of an attribute grammar can be described in the syntactic-semantic metalanguage of the system. The semantic assignments in the description of an attribute grammar are Pascal-like expressions and procedure callings as the generated compilers are complete Pascal programs. An attribute grammar definition in the HLP/PAS system begins with the declaration of these procedures. After this the names and the types of the synthesized and inherited attributes can be defined. Both standard Pascal and user defined types can be used as the types of these attributes. Then the nonterminal declaration part follows, in which the nonterminals and the names of the attributes associated with them are described. After this the tokens and the terminals of the grammar are defined. Finally, in the last part of the specification, the syntactic rules and the semantic assignments are described. There are conditional statements which can be associated with the rules of the grammar. These statements can be applied to send messages during the compilation by means of evaluted attributes and are also used to control the generated backtrack parsers (see 4). The code generator statements generate the target code in the constructed compilers. Of course these statements also use the evaluated attributes. The evaluation time of a conditional statement is determined only by attribute dependencies while the evaluation sequence of the code-generator statements can be prescribed by the user. The system contains a simple error-recovery method which can be influenced by the definition of the grammar. If a set of terminal or token symbols (SKIP-set) is connected to a nonterminal and there is a syntactic error in the "subtree" rooted in this nonterminal during the parsing then the parser reads the input until it finds an element of the SKIP set. This symbol will be the next input symbol and the corresponding subtree is deleted (Panic method). The parser of the metalanguage always checks the formal correctness of a specification e.g. the name conflicts, the existence of superflous nonterminals, the consistence of the attribute assignments etc. The system can automatically generate so called copy rules for the simple transport of attribute values if the assignment is determined unambiguously. Finally, from a correct specification the parser of the metalanguage constructs files for other moduls of the system. As we mentioned earlier, both one-pass and multi-pass compliers can be generated. The one-pass compilers use LL (1) parsing method and L-attribute evaluation strategy [1]. In the multi-pass compilers LALR (1) parsing method and a modified version, MOAG [4] of the OAG [6] attribute evaluation method are applied.

3. An attribute grammar for normal ECG waveforms

On the basis of [9] an attribute grammar is presented for the description of normal ECG waveforms in the HLP/PAS system. This grammar is used to illustrate the backtrack parsing in the system. The first step in a description of a pattern is to determine the set of the primitives. These primitives are the terminal symbols of the grammar. First an ECG waveform is approximated with line segments [3]. The line segments are partitioned into pieces nearly of the same size (these are the primitives). This partition is carried out by using a UNIT segment (see Figure 1). A slope symbol is associated with each primitive as follows:

if $\nu_H < \varphi_P \leqq \nu_S$ then SP
if $\nu_S < \varphi_P \leqq \nu_I$ then IP
if $\varphi_P > \nu_I$ then LP
if $-\nu_H > \varphi_P \geqq -\nu_S$ then SN
if $-\nu_S > \varphi_P \geqq -\nu_I$ then IN
if $\varphi_P < -\nu_I$ then LN
if $-\nu_H < \varphi_P \leqq \nu_H$ then HP

were φ_P is the angle of the line segment S_P with the horizontal axis and ν_H, ν_S; ν_I are predefined constant angles. Each primitive in a segment has the same size and if the UNIT not too large then there is not large difference between the size of the primitives of different segments. Moreover each primitive has a duration which is the projection of the primitive to the time axis (Figure 1.).

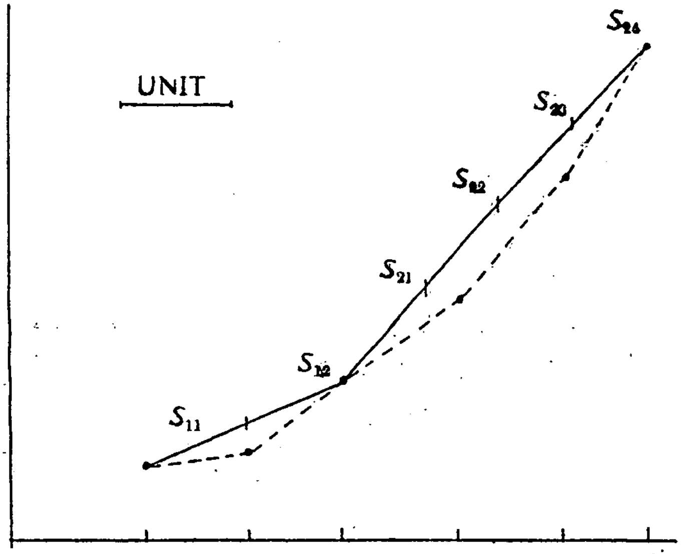

Figure 1

In Figure 1 the dashed lines indicate the pieces of a waveform and the solid lines are the segments. We can see that the duration of the primitives S_{11}, S_{12} is 1 and 3/4 of the primitives S_{21}, S_{22}, S_{23}, S_{24}. If the $v_H=10°$, $v_S=30°$ angle constants are used then the waveform can be coded as follows

$$(\mathrm{SP}, 1)(\mathrm{SP}, 1)(\mathrm{IP}, 0.75)(\mathrm{IP}, 0.75)(\mathrm{IP}, 0.75)(\mathrm{IP}, 0.75).$$

In Appendix A a grammar is given for the description of normal ECG. In the description X^n denotes $X \ldots Xn$ times. The grammar is ambiguous. For example consider the rules

11. $\mathrm{T} \to \mathrm{FGH}$; 12. $\mathrm{F} \to \mathrm{K}^4|\mathrm{K}^3|\mathrm{K}^2$; 13. $\mathrm{G} \to \mathrm{I}^3|\mathrm{I}^2|\mathrm{I}|\varepsilon$;

16. $\mathrm{K} \to \neq \mathrm{IP} \neq \mathrm{DIG}| \neq \mathrm{SP} \neq \mathrm{DIG}| \neq \mathrm{HP} \neq \mathrm{DIG}$ and

17. $\mathrm{I} \to \neq \mathrm{HP} \neq \mathrm{DIG}| \neq \mathrm{SP} \neq \mathrm{DIG}| \neq \mathrm{SN} \neq \mathrm{DIG}$;

Starting from the nonterminal T both the

$$\mathrm{T} \to \mathrm{FGH} \to \mathrm{K}^4\mathrm{GH} \to \mathrm{K}^4\mathrm{IH} \ldots \quad \text{and the}$$

$$\mathrm{T} \to \mathrm{FGH} \to \mathrm{K}^3\mathrm{GH} \to \mathrm{K}^3\mathrm{I}^2\mathrm{H} \ldots$$

derivation leads to the $(\neq\mathrm{HP}\neq\mathrm{DIG})^5$ string. The grammar in Appendix A is augmented with attributes and semantic assignments. These assignments compute the durations of the cardiac cycles from that of the primitives and determine the maximal durations of cycles (maxdur, mindur). An ECG is normal if $\frac{\text{maxdur-mindur}}{\text{maxdur}} > 0.1$. In Appendix B the description of the augmented ECG grammar is presented in the metalanguage of the HLP/PAS system. The description does not contain the complete grammar, only the most important parts of the specification are given. Of course using more attributes in the description further characterictics of ECG waveforms can be analysed.

4. The generated backtrack parsers

As it was already mentioned, the one-pass compiler generator part of the HLP/PAS was extended with a backtrack parser generator. First the structure of the one-pass compilers generated originally is outlined. For each nonterminal of the grammar a Pascal procedure is constructed. The inherited and synthesized attributes of a nonterminal are the input and output parameters of the procedure corresponding to this nonterminal. For example consider the following rules:

$$\begin{aligned} X_0 &\to X_{11} X_{12} \ldots X_{1n_1} \\ &\vdots \\ X_0 &\to X_{q1} X_{q2} \ldots X_{qn_q} \end{aligned}$$

The structure of the generated procedure is:

```
    procedure $X_0\$(I(X_0);$ var $S(X_0));$
      record $X_{11}$ declaration of $A(X_{11})$; end
        ⋮
      record $X_{qn_q}$ declaration of $A(X_{qn_q})$; end
      begin
      if $SY\$ \in S_1$ then begin
                       eval $(I(X_{11}))$; $X_{11}\ \$(I(X_{11}); S(X_{11}))$;
                       ⋮
                       eval $(I(X_{1n_1}))$; $X_{1n_1}\ \$(I(X_{1n_1}); S(X_{1n_1}))$; end else
                       ⋮
      if $SY\$ \in S_q$ then begin
                       eval $(I(X_{q1}))$; $X_{q1}\ \$(I(X_{q1}); S(X_{q1}))$;
                       ⋮
                       eval $(I(X_{qn_q}))$; $X_{qn_q}\ \$(I(X_{qn_q}); S(X_{qn_q}))$;
        end else error;
    eval $(S(X_0))$;
end of procedure $X_0\$$;
```

where $I(X_{ij})$, $S(X_{ij})$, $A(X_{ij})$ denote the inherited, synthesized and the all attributes of the nonterminal X_{ij}, respectively. For each different right-hand side nonterminal a record structure is generated. The variable $SY\$$ contains the current input symbol. The corresponding alternative is determined by the condition $SY\$ \in S_i$, where $S_i=$ $=\mathrm{FIRST}_1(X_{i1}, \ldots, X_{in_i})\oplus_1\mathrm{FOLLOW}_1(X_0)$. In the blocks of the alternativies, eval $(I(X_{ij}))$ denotes the evalution of the inherited attributes of the nonterminal X_{ij}. The places of the conditional and code generator statements in the alternativies are determined by attribute dependencies and the prescriptions of the user (see 2. section). The callings of the lexical procedure are also in the blocks of the alternatives. Instead of building the parse tree, only recursive procedure callings are executed during the parsing.

In the backtrack version of the generated compilers the instances of the procedures in a calling sequence are numbered (see Figure 2.).

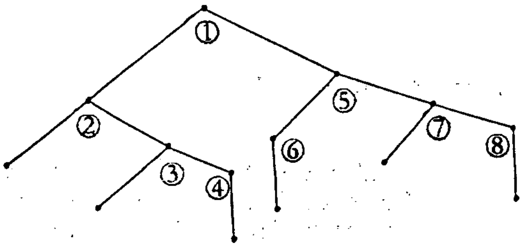

Figure 2

The greatest sequence number of the instances is stored in the global variable NUM$. In each procedure there is a local variable (NUMX) to store the number of the actual instance in the calling sequence. The TRUE value of the global Boolean variable BTRACK denotes that the parser is in backtracking mode. During the parsing a global stack is handled. The I-th element of this stack gives the number of the alternative chosen in the I-th instance. A flag denotes if another alternative with greater number can be chosen in this instance. Of course only those alternatives are considered for which the condition $SY\$\in S_i$ is true. In the global variable LPOINT the number of an instance is stored. In backtracking mode the new alternative will be chosen from this instance. Finally in each procedure there is a pointer (PT) to denote the position of the current input symbol at the entry of the corresponding instance. If there is an error in the K-th instance then BTRACK=TRUE and this instance is terminated. In the recursive calling structure the procedure instances are terminated until the condition NUMX>LPOINT is true. If NUMX<LPOINT then using the NUMX, NUMX+1, ..., LPOINT−1 elements of the stack and the pointers PT the necessary part of the parsing is reconstructed. In the instance indicated by LPOINT the new alternative is chosen. The assignments NUM$=NUMX, BTRACK=FALSE are executed and in the variable LPOINT the new backtracking point is stored. In [8] a backtrack parser was presented for pattern recognition. The main advance of the parser presented in this paper against that of [8] is that using the LL (1) conditions a lot of useless backtracks can be eliminated. Of course there is a cost of the computation of the LL (1) tables but this computation happens only ones in meta-compiling time.

To illustrate the backtrack parser consider the following structure of the ECG grammar: $ST\rightarrow I^{10}|I^{9}|I^{8}|I^{7}|I^{6}$.

It can be described with the following three rules:

```
  i) ST=I_LIST;
      DO
      I_LIST.length:=0;
      END
 ii) I_LIST=I  I_LIST;
      DO
      dur:=I.dur+I_LIST:dur;
      I_LIST.length:=length+1;
      COND
      if I_LIST.length>10 then BACKTRACK;
      END
iii) I_LIST=I;
      COND
      if length<6 then BACKTRACK;
      END
```

The inherited attribute length is used to count the I elements. The backtrack is controled by this attribute. For example if the rule ii) was applied ten times then a backtrack is executed for the nonterminal I_LIST and the alternative iii) is chosen instead of ii). On the other hand if in the rule iii) the condition length <6 is true then after several backtracks the instance of the nonterminal ST is terminated in backtracking mode. These redundant steps can also be eliminated if the number of I elements is stored in a synthesized attribute of ST and the condition length <6 is applied in the rule i). This solution can be seen in Appendix C.

5. Further research

The backtrack parsers presented in this paper use L-attribute evaluation method. This method can be applied to languages the elements of which depend on their left-hand side environments. It often holds in the case of programming languages but not in the case of pattern descriptions. A subpattern usually depends on both its left- and right hand side environments. Hence multi-pass attribute evaluators are needed. In such type parsers, attributed parsing trees are constructed to store the value of the evaluated attributes and the structure of the parsing. As we mentioned it earlier, in many cases the patterns can conveniently be described only by ambiguous grammars. Therefore the development of a multi-pass, backtrack parser generator in the HLP/PAS system would be needful. Because such type parsers work usually very slowly, in our opinion, a combination of the pass-directed and the dinamic attribute evaluation strategies is needed. When backtrack, some attribute values have to recompute. In these cases the appliement of the dinamic attribute evaluation method is efficient. Only those attributes must be recomputed the values of which are changed during the backtrack. In the other part of the grammar (and usually it is the larger part) a pass-directed evaluation method can be used e.g. MOAG [4].

6. Conclusions

In this paper a syntactic pattern recognition system was presented. The input of the system is a complete description of a pattern by attribute grammar. From this specification the recognizer of the pattern is generated. In the description of patterns ambiguous grammars can also be used. The generated parsers use the LL (1) tables so a lot of redundant backtracks can be eliminated. Further characteristic of the generated parsers is that the parsing can be influenced by the evaluated attributes. Calling the start symbol of an ambiguous grammar repeatedly the all possible derivations of the grammar can be constructed for a given input. The complete system was implemented on Pascal language on IBM—370 and IBM XT compatible computers.

Acknowledgements

We wish to thank Árpád Makay and Zoltán Fülöp for their constructive comments on this paper.

Appendix A

1. S=NORMAL_ECG
2. NORMAL_ECG=CARDIAC_CYCLE NORMAL_ECG
3. NORMAL_ECG=R
4. CARDIAC_CYCLE=RS ST T TP P PR Q

5. $R=C\ D$
6. $RS=C\ D\ E$
7. $C= \neq LP \neq$ DIG $C| \neq LP \neq$ DIG
8. $D= \neq LM \neq$ DIG $D| \neq LM \neq$ DIG
9. $E= \neq LP \neq$ DIG $E| \neq IP \neq$ DIG $E| \neq LP \neq$ DIG$| \neq IP \neq$ DIG$|\varepsilon$
10. $ST=I^{10}|I^{9}|I^{8}|I^{7}|I^{6}$
11. $T=F\ G\ H$
12. $F=K^{4}|K^{3}|K^{2}$
13. $G=I^{3}|I^{2}|I|\varepsilon$
14. $H=M^{4}|M^{3}|M^{2}|M|\varepsilon$
15. $M= \neq IM \neq$ DIG$| \neq SM \neq$ DIG
16. $K= \neq IP \neq$ DIG$| \neq SP \neq$ DIG$| \neq HP \neq$DIG
17. $I= \neq HP \neq$ DIG$| \neq SP \neq$ DIG$| \neq SM \neq$DIG
18. $TP=I^{14}|I^{13}|I^{12}|I^{11}|I^{10}|I^{9}|I^{8}$
19. $P=T$
20. $PR=I^{4}|I^{3}|I^{2}|I|\varepsilon$
30. $Q=L^{3}|L^{2}|L|\varepsilon$
31. $L= \neq IM \neq$DIG$| \neq LM \neq$DIG
32. DIG = NUMBER

Appendix B

```
ATTRIBUTE GRAMMAR ECG
(* B+  BACKTRACK OPTION IS ON *)
PASCAL DECLARATIONS ARE
  PROCEDURE BF (a, b: INTEGER; VAR c: BOOLEAN);
  BEGIN
  IF (a-b)/a> 0.1 THEN c:=TRUE ELSE c:=FALSE;
  END;
  PROCEDURE MAXF (VAR a:INTEGER; c, b: INTEGER);
  BEGIN IF b> c THEN a:=b ELSE a:=c; END;
  PROCEDURE MIN (VAR a: INTEGER; b, c: INTEGER);
  BEGIN
  IF b>c THEN a:=c ELSE a:=b;
  END;
SYNTHESIZED ATTRIBUTES ARE
  maxdur, mindur, dur :INTEGER;
  fl: BOOLEAN; val:INTEGER;
INHERITED ATTRIBUTES ARE
  length: INTEGER;
NONTERMINALS ARE
  ECG HAS fl;
  NORMAL_ECG HAS maxdur, mindur;
  CARDIAC_CYCLE, R, RS, ST,T,TP, P, PR, Q, C, D, E, DIG, F, G, H HAVE
  dur;
  LP_PAIR, LM_PAIR, IP_PAIR, IM_PAIR, HP_PAIR, SP_PAIR, SM-PAIR
    HAVE dur;
```

```
  I_SET, K_SET, M_SET, L_SET HAVE dur;
  I1_LIST, I2_LIST, I3_LIST, I4_LIST, L_LIST, K_LIST, M_LIST HAVE
    length, dur;
  TOKENS ARE
  NUMBER HAS val;
  TERMINALS ARE
    "LP", "LM", "IP", "IM", "HP", "SP", "SM";
PRODUCTIONS ARE
ECG= NORMAL_ECG;
  DO
  fl< -BF(NORMAL_ECG.maxdur, NORMAL_ECG.mindur, fl);
  END
NORMAL_ECG=CARDIAC_CYCLE NORMAL_ECG;
  DO
    maxdur< -MAXF(maxdur, CARDIAC_CYCLE.dur, NORMAL_ECG.
    maxdur);
    mindur< -MINF (mindur, CARDIAC_CYCLE.dur, NORMAL_ECG.
    mindur);
  END
NORMAL_ECG=R;
  DO
     maxdur :=0;
     mindur :=0;
  END
    CARDIAC_CYCLE=RS  ST  T TP  P PR Q;
  DO
    dur := RS.dur+ST.dur+T.dur+TP.dur+P.dur+PR.dur+Q.dur;
  END
     :
```

Appendix C

```
 i) ST=I_LIST
      DO
      I_LIST.length:=0;
      COND
      IF I_LIST.slength <6 THEN BACKTRACK;
      END

ii) I_LIST=I I_LIST;
      DO
      dur:= I.dur+I_LIST.dur;
      I_LIST.length:=length+1;
      slength=I_LIST.slength+1;
      COND
      IF I_LIST.length > 10 THEN BACKTRACK;
      END
```

```
iii) I_LIST=I;
     DO
     slength:=1;
     END
```

RESEARCH GROUP ON THEORY OF AUTOMATA
HUNGARIAN ACADEMY OF SCIENCES
SOMOGYI U. 7.
SZEGED, HUNGARY
H—6720

References

[1] BOCHMANN, G. V., Semantic evaluation from left to right, Commun. Ass. Comput. Mach., vol 19, pp. 55—62, Feb. 1976.
[2] FU, K. S., (ed), Syntactic pattern recognition, applications, Springer-Verlag, New York/Berlin 1977.
[3] GRITZALI, F. and G. PAPAKONSTANTINOU, A fast piecewise linear approximation algorithm, Signal Processing, vol. 5, pp. 221—227, 1983.
[4] GYIMÓTHY, T., E. SIMON and Á. MAKAY, An implementation of the HLP, Acta Cybernetica, Tom 6, Fasc. 3, pp. 315—327.
[5] GYIMÓTHY, T., KOCSIS, F., MAKAY, Á., SIMON, E. and TOCZKI, J., The compiler generator HLP/PAS, Computer and Automation Institute, Hungarian Academy of Sciences, Report, to be published.
[6] KASTENS, U., Ordered Attribute Grammars, Acta Informatica 13, 229—256, 1980.
[7] KOSKIMIES, K. and PAAKKI, J., HLP84-Semantic metalanguage and its implementation, University of Helsinki, Report C—1983—69.
[8] PAPAKONSTANTINOU, G., An interpreter of attribute grammars and its application to the waveform analysis, IEEE Transactions on Software Engineering, vol. SE-7, No. 3, pp. 279—283, May 1981.
[9] SKORDALAKIS, E. and PAPAKONSTANTINOU, G., Toward an attribute grammar for the description of ECG waveforms, 7-th International Conference on Pattern Recognition, 1984.
[10] TOCZKI, J., et al., On the Pascal implementation of the HLP, Proc. of 4th Hungarian Computer Science Conference, Győr, 1985, 12 pp.
[11] YOU, K. C. and FU, K. S., A syntactic approach to shape recognition using attributed grammars, IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-9, No. 6, pp. 334—345, 1979.

(Received May 6, 1986)