A Programming System for Resolution Theorem Prover

Masato YAMAZAKI* and Akira YAMAMOTO**

Abstract

This paper describes a programming system for resolution theorem prover. Users can easily implement various kinds of strategies which are previously known or newly developed, therefore the system is effective in evaluations or comparisons of the strategies. High descriptive ability of the system enables users to implement complicated strategies such as the semantic resolution.

The system is composed of five groups of LISP functions: (1) extracting a pair of clauses from lists of clauses, (2) handling a list of clauses, (3) executing resolution, factoring etc., (4) subsidiary functions which are used together with functions in (2) or (3), and (5) miscellaneous functions.

1. Introduction

Resolution theorem prover has a demerit of inefficiency compared to other heuristic ones. To remedy this, many strategies have been developed. These strategies are classified into three groups: (1) deletion strategies; deleting subsumed clauses, tautology clauses, etc., (2) preference strategies; such as the unit preference strategy, (3) restrictive strategies; the semantic resolution, the set of support strategy, etc. The programming system is developed for easy implementation of various kinds of strategies because each strategy is effective only for a limited class of problems. First we extract basic operations of the three groups of strategies. Then we take five groups of LISP functions and a data array named CLIST as the main components of the system. This system is intended to have high descriptive ability to accept complicated strategies. Main features of the system are given and clarified with some examples.

This paper first appeared in Japanese in Joho—Shori (Journal of the Information Processing Society of Japan), Vol. 17, No. 5 (1976), pp. 410~416.

* Automatic Control Division, Electrotechnical Laboratory

** Data Processing System Division, OKI Electric Industry Co., Ltd.

2. The programming system

2.1 Configuration

Table 1 shows the basic system functions. All the clauses are processed on a work area named CLIST. CLIST is a one dimensional array of size eight and each element consists of a list of clauses.

This system has a merit of high descriptive ability compared to other system(4). This ability depends on the following features: (1) conditional test on an extracted clause-pair is possible because the function for resolution is separated from the function for extracting a pair; (2) users can explicitly assign a clause to any location in CLIST; (3) various kinds of preference strategies can be implemented uniformly by making use of the subsidiary functions.

group	function name	feature
(1)	getpair	generates a clause pair. Sec 2.2
(2)	addclist getclist delclist	handles CLIST. See 2.3.
(3)	resolve p-resolve factor	executes resolutions, factorings, etc. See 2.4.
(4)	clength cdepth model lorder llast	will be used together with (2) or (3) functions. See 2.5.
(5)	intlz p-reset gocv	miscellaneous functions. See 2.6.

Table 1. Basic system functions

*GPOINT and *FPOINT are used as the pointers of CLIST in order to extract a clause-pair or a clause exactly once, respectively. *HIST stores the histories of all the clauses which are initially given or generated later. Temporary variables such as PL, CL, LB, GTOP1 through GTOP8 are also used.

2.2 Function for extracting a clause-pair

getpair [(i j); l1; l2]

Getpair sets PL to a clause-pair extracted from CLISTi and CLISTj, then sets LB to l1. If there exists no new pair, it sets PL to NOPAIR, then LB to l2. In order to prevent extracting a clause-pair more than once, four pointers (GCTOPi, GCTOPj, GCURRi and GCURRj) in *GPOINT and two variables GTOPi and GTOPj are used.

n	GTOPi	GCTOPi	GCURRi	PAIR	GCURRj	GCTOPj	GTOPj
0	3	1	0	----	0	1	4
1	3	1	1	(1 1)	0	1	4
2	3	2	0	(2 1)	1	1	4
3	3	2	1	(1 2)	0	2	4
4	3	2	2	(2 2)	0	2	4
5	3	3	0	(3 1)	1	2	4
6	3	3	0	(3 2)	2	2	4
7	3	3	1	(1 3)	0	3	4
8	3	3	2	(2 3)	0	3	4
9	3	3	3	(3 3)	0	3	4
10	3	3	1	(1 4)	0	4	4
11	3	3	2	(2 4)	0	4	4
12	3	3	3	(3 4)	0	4	4
13	3	3	3	NOPAIR	0	4	4

Fig. 1. Clause pairs extracted by the successive calls of getpair

GTOPi indicates the number of clauses in CLISTi. Fig. 1 shows the pairs extracted by the successive calls of getpair. In this case CLISTi and CLISTj contain three and four clauses, respectively.

2.3 List-handling functions

addclist [l; i]

Addclist adds l (a clause list) at the end of CLISTi. If the second argument is a function fn, fn is applied to the clause list; and this value will be considered as the second argument.

getclist [i; n]

Getclist sets CL to the n-th clause of CLISTi. The value of getclist is this clause.

delclist [i; n]

Delclist deletes the n-th clause from CLISTi.

Functions in this group will automatically update the pointers *GPOINT, *FPOINT, and GTOP1 through GTOP8.

2.4 Functions for resolution

Functions in this group perform resolution, factoring etc.

resolve [pl; fn; ls; lt; lf]

Resolve generates a list of all the possible resolvents of a clause-pair pl, then it sets CL to this list and adds the history of each resolvent to *HIST. In case the empty clause, some other clauses, or no clause is generated, LB is set to ls, lt, or lf, respectively. Fn is applied to the clauses; then the maximum and the minimum values are put on VH and VL, respectively.

2.5 Subsidiary functions

These functions are used accompanying the main functions which belong to the group (2) or (3). Each subsidiary function has certain effects on its main function.

clength [c]

The value of clength is the number of literals in a clause c. If the value is greater than VMAX or VH, or less than VL, then these values will be updated. VMAX is initially set to zero by the function intlz, and will be updated from that time to indicate the maximum value of all the subsidiary functions. VH and VL are set to zero when the main function is called. They indicate the maximum and the minimum values of the subsidiary function in the scope of the main function.

model [c]

If a clause c is false in a given interpretation, then the value of model is one; else the value is two. The interpretation will be given by the user as shown in the example in 3.3.

2.6 Miscellaneous functions

Functions for initialization, functions for conditional expressions, arithmetic functions, logical functions etc. are included in this group.

3. Examples

3.1 The unit preference strategy(1)

This strategy chooses shorter clauses (consisting of fewer literals) as the parent clauses in every resolution. Fig. 2 shows an implementation.

```
(UNIT-PREFERENCE (LAMBDA (S) (PROG (L H J)
     (INTLZ)
     (SETQ L S)
     (ADDCLIST L (FUNCTION CLENGTH) )
     (SETQ J 1)
P1   (GETPAIR (1 J) P1S P1F)
     (GOEV LB)
P1F  (COND ((GREATERP J VMAX) (GO NU) )
           (T (ADD 1 J)(GO P1) )        )
P1S  (RESOLVE PL (FUNCTION CLENGTH) SUC P11 P1)
     (GOEV LB)
P11  (ADDCLIST CL (FUNCTION CLENGTH) )
     (SETQ J VL)
     (GO P1)
NU   (SETQ J 2)
P2   (FACTOR J NIL P2S P2F)
     (GOEV LB)
P2F  (COND ((EQ J 2) (ADD 1 J)(GO P2) )
           (T (GO NU1) )                )
P2S  (ADDCLIST CL (FUNCTION CLENGTH) )
     (SETQ J VL)
     (GO P1)
NU1  (SETQ H 2)
     (SUB 1 J)
P3   (GETPAIR (H J) P3S P4)
     (GOEV LB)
P3S  (RESOLVE PL (FUNCTION CLENGTH) SUC P31 P3)
     (GOEV LB)
P31  (ADDCLIST CL (FUNCTION CLENGTH) )
     (SETQ J VL)
     (GO P1)
P4   (COND ((EQ H VMAX) (GO FAIL) )
           ((GEP (PLUS 1 H) J) (ADD H J)(GO P2))
           (T (ADD 1 H)(SUB 1 J)(GO P3))        )
SUC  (PRINT (QUOTE SUCCESS))
     (RETURN (PRINT *HIST))
FAIL (RETURN (PRINT (QUOTE FAIL)))
)))
```

Fig.2. The unit preference strategy

3.2 The set of support strategy(2)

A subset T of a set S of clauses is called a set of support of S if S-T is satisfiable. The set of support strategy assures that at least one of the parent clauses is chosen from T or the descendant of T. Fig. 3 shows a simple implementation where the set SS corresponds to the set S-T and ST to T.

```
(SET-OF-SUPPORT (LAMBDA (ST SS) (PROG (J)
     (INITZ)
     (ADDCLIST ST 1)
     (ADDCLIST (FACTOR 1 NIL) 1)
     (ADDCLIST SS 2)
     (ADDCLIST (FACTOR 2 NIL) 2)
P0   (SETQ J 0)
P1   (COND ((EQ J 2) (GO FAIL)) )
     (ADD 1 J)
P2   (GETPAIR (1 J) P2S P1 )
     (GOEV LB)
P2S  (RESOLVE PL NIL SUC P21 P2 )
     (GOEV LB)
P21  (ADDCLIST CL 1)
     (ADDCLIST (FACTOR 1 NIL) 1)
     (GO P0)
SUC  (PRINT (QUOTE SUCCESS) )
     (RETURN (PRINT *HIST) )
FAIL (RETURN (PRINT (QUOTE FAIL)) )
)))
```

Fig.3. The set of support strategy

3.3 The semantic resolution(3)(5)

This is an effective but complicated restrictive strategy. The main points of this strategy are the following two: (1) it divides all the clauses into two classes, and prevents resolution in the same class, (2) it tries resolution of only a clash not to generate a redundant or an intermediate clause. Fig. 4 shows an implementation and Fig. 5 an interpretation and a problem. To obtain resolvents of a clash, the algorithm by Chang and Lee(6) is used.

4. Conclusion

A programming system for resolution theorem prover is described above. This kind of system is effective in evaluations and comparisons of many strategies. The merit of this system is high descriptive ability which enables (1) conditional test on an extracted clause-pair, (2) explicit assignment of a clause to any location in CLIST, and (3) uniform implementations of various kinds of preference strategies.

```
(SEMANTIC-RESOLUTION (LAMBDA (L PORDER) (PROG (M N J I)
      (INITZ)
 S1   (SETQ J 1)
 S2   (ADDCLIST L (FUNCTION MODEL))
      (SETQ M (GETCLIST 1 NIL))
      (ADDCLIST M 3)
      (ADDCLIST (FACTOR 3 NIL) 3)
      (SETQ N (GETCLIST 2 NIL))
 S3   (SETQ I 0)
 S4   (COND ((MEMBER (QUOTE SUC)(GETCLIST 1 NIL))(GO SUC)))
 S5   (COND ((NULL (GETCLIST 2 NIL))(GO S8) ))
 S6   (CLEARCLIST 4)
      (ADDCLIST 2 4)
      (CLEARCLIST 1) (CLEARCLIST 2)
 P1   (GETPAIR (3 4) P2 S7)
      (GOEV LB)
 P2   (P-RESOLVE PL NIL (FUNCTION LORDER)(FUNCTION LLAST)
                                          P2S P2S P1)
      (GOEV LB)
 P2S  (ADDCLIST CL (FUNCTION MODEL))
      (ADDCLIST 1 5)
      (GO P1)
 S7   (ADD 1 J)
      (GO S4)
 S8   (ADDCLIST 5 6)
      (ADDCLIST 6 7)
 S9   (ADD 1 J)
 S10  (ADDCLIST (FACTOR 6 NIL) 6)
      (ADDCLIST N 8)
      (CLEARCLIST 1)(CLEARCLIST 2)
 P3   (GETPAIR (6 8) P4 S11)
      (GOEV LB)
 P4   (P-RESOLVE PL NIL (FUNCTION LORDER) NIL P4S P4S S11)
      (GOEV LB)
 P4S  (ADDCLIST CL (FUNCTION MODEL))
 S11  (GO S3)
 SUC  (PRINT (QUOTE SUCCESS))
      (RETURN (PRINT *HIST))
)))
```

Fig.4. The semantic resolution

```
DEFINE ((
 (F (LAMBDA (X Y Z)
     (COND ((EQ X (QUOTE A))(EQ Y Z))
           ((EQ Y (QUOTE A))(EQ X Z))
           (T (EQ Z (QUOTE A)) )     ))
 (G (LAMBDA ( X Y)
     (COND ((EQ X Y)(QUOTE A))
           (T (QUOTE B))      ) ))
 (H (LAMBDA (X Y)
     (COND ((EQ X Y)(QUOTE A))
           (T (QUOTE B))      ) ))
 (K (LAMBDA (X) X       ))
))

SEMANTIC-RESOLUTION (
  (   (1 (X Y) ((P (G X Y) X Y)))
      (2 (X Y) ((P X (H X Y) Y)))
      (3 (U V W X Y Z) ((# P U Z V)(# P X V W)
                         (# P X Z V)(P U Z W)  ))
      (4 (X) ((# P (K X) X (K X) )))              )
  (    P    )
                    )
```

Fig.5. Examples of an interpretation and a problem

Acknowledgement

The authors thank Dr. T. Nagata of Electrotechnical Laboratory and Prof. M. Mukaidono of Meiji University for their discussions and valuable comments on the manuscript.

References

1) L. Wos, D. Carson and G. Robinson: The unit preference strategy in theorem proving, Proc. FJCC, (1964).

2) L. Wos, G. Robinson and D. F. Carson: Efficiency and completeness of the set of support strategy in theorem proving, JACM, Vol. 12, No. 4, (1965).

3) J. R. Slagle: Automatic theorem proving with renamable and semantic resolution, JACM, Vol. 14, No. 4, (1967).

4) L. Henshen, R. Overbeek and L. Wos: A theorem proving language for experimentation, CACM, Vol. 17, No. 6, (1974).

5) J. A. Robinson: A review of automatic theorem proving, Proc. Symp. App. Math., (1967).

6) C. L. Chang and R. C. T. Lee: Symbolic logic and mechanical theorem proving, Academic Press, (1973).