Jan Štěpán

Automated theorem proving in monadic predicate calculus

Acta Universitatis Palackianae Olomucensis. Facultas Rerum Naturalium. Mathematica, Vol. 30 (1991), No. 1, 273--283

Persistent URL: http://dml.cz/dmlcz/120263

ACTA UNIVERSITATIS PALACKIANAE OLOMUCENSIS
FACULTAS RERUM NATURALIUM
1991 MATHEMATICA XXX VOL. 100

Katedra výpočetní techniky
přírodovědecké fakulty Univerzity Palackého v Olomouci
Vedoucí katedry: PhDr.RNDr.Jan Štěpán, CSc.

AUTOMATED THEOREM PROVING IN MONADIC PREDICATE CALCULUS

JAN ŠTĚPÁN

(Received January 31, 1990)

Abstract: In this paper a modification of the method of analytical tables is described which makes possible to construct an effective logic program for automated theorem proving in monadic predicate calculus. The logic program is enclosed and its function is demonstrated.

Key words: Monadic predicate calculus, analytical table, logic program.

MS Classification: 68G15

In the sphere of logic programming there exists a great number of programs for proving of theorems of the propositional calculus (see for example [1], [2]). That is to say - highly effective algorithms of the matematical logic were developed which make the construction of such programs possible. For the proving of theorems of the predicate calculus there exists a very strong universal instrument - resolution principle. But this method is usually very slow in practice. Therefore it is

more convenient to pay attention on the solving of these questions in some more narrow or special field of problems. In this case such a narrow field seems to be the monadic predicate calculus. By monadic predicate calculus we mean the logic with maximally unary predicate constants and (specially here) only with Ø-ary (individual) function constants. For this calculus there exist algorithms, by which total decidability of provability is guaranteed. One of these algorithms is the method of analytical tables (acc. [3]).

Denotation: symbols ~ , & , # , => , <=> denote negation, conjuction, disjunction, implication and equivalence respectively, symbols "all" and "ex" denote universal and existential quantifier respectively.

I. Proof construction

The method of analytical tables is based on decomposition of formula to simpler components - subformulas of considered formula. The models of the decomposition are the rules for construction of tables. The analytical table of the formula X is taken as a binary tree (graph), the nodes of which are occurences of the formulas, and which is constructed as following - by the help of four-type rules:

Rule A: $\frac{C}{C1 \\ C2}$ (conjuctive) Rule B: $\frac{D}{D1 \mid D2}$ (disjunctive)

Rule C (universal): $\frac{A}{A(a)}$, where a is an arbitrary parameter.

Rule D (existential): $\frac{E}{E(a)}$, where a is a new parameter.

Rules of the type A and B deal with propositional connectives, rules of the type C and D serve for elimination of quantifiers.

Construction process:

1. the root of the tree is formula X;
2. let formula Y be terminal node of the given tree
   - if there - on the path from X to Y - occurs a formula C, then any of formulas C1 or C2 as the only successor of node Y can be added - we usually and step by step firstly

C1, secondly C2 (the tree in considered branch develops linearly);

- if there - on the path from X to Y - occurs a formula D, then formula D1 can be added as left successor and D2 as right successor of formula Y (in the node Y the tree develops into two branches);
- if there - on the path from X to Y - occurs a formula A, then formula A(a) can be added;
- if there - on the path from X to Y - occurs a formula E, then formula E(a) can be added, but a-parameter choice is limited by reservation.

The branch of given tree is said to be closed, if it contains a formula and its negation. Analytical table (tree) is called to be closed, if every of its branches is closed. The proof of the formula X is then understood as a closed table for formula ~X. Such accepted proof seems to demonstrate that every branch of decomposition of formula ~X forms inconsistent set of formulas. That is why the formula ~X is inconsistent, hence formula X is tautology or theorem.

Decompositional rules, which may be used in above mentioned process, are according types:

- rules A with two successors

R1: $\dfrac{X \,\&\, Y}{\begin{array}{c} X \\ Y \end{array}}$ R2: $\dfrac{\sim(X \,\#\, Y)}{\begin{array}{c} \sim X \\ \sim Y \end{array}}$ R3: $\dfrac{\sim(X \Rightarrow Y)}{\begin{array}{c} X \\ \sim Y \end{array}}$

- rules A with one successor

R4: $\dfrac{\sim\sim X}{X}$ R5: $\dfrac{X \Leftrightarrow Y}{(X \Rightarrow Y)\,\&\,(Y \Rightarrow X)}$ R6: $\dfrac{\sim(X \Leftrightarrow Y)}{\sim X \Leftrightarrow Y}$

- rules B

R7: $\dfrac{X \,\#\, Y}{X \mid Y}$ R8: $\dfrac{\sim(X \,\&\, Y)}{\sim X \mid \sim Y}$ R9: $\dfrac{X \Rightarrow Y}{\sim X \mid Y}$

- rules C

R10: $\dfrac{\text{all}\,(x,A)}{A(x/a)}$ R11: $\dfrac{\sim\text{ex}(x,A)}{\sim A(x/a)}$

- rules D (with parameter choice reservation)

R12: $\dfrac{\text{ex}(x,E)}{E(x/a)}$ R13: $\dfrac{\sim\text{all}(x,E)}{\sim E(x/a)}$

Note to rule D: If we prove during an argumentation (for example in mathematics), that there exists an element x holding a property p, i.e. that ex(x,p(x)) is valid, then we can affirm "let a be such x", i.e. p(a) is valid. If we later prove, that ex(x,q(x)) is valid for some property q, then it is not possible to affirm "let a be such x", because a had been related to property p, therefore we take a new parameter b, we say "let b is such x" and we write q(b). So that is the reason for reservation in rule D.

If we apply that reservation in proof is sufficient to restrict the application on considered branch of proof. That means we choose such parameter, that in given branch had not been used in a rule of the type D. This simplification gives a possibility to shorten some proofs, but the main merit is a possibility to avoid multiple applications of rules of the type C, which is visible from following example.

Example: proof of formula

all(x,p(x) => q(x)) => (all(x,p(x)) => all(x,q(x)))

1. ~(all(x,p(x) => q(x)) => (all(x,p(x)) => all(x,q(x))))
2. all (x,p(x) => q(x)) R3(1.)
3. ~(all(x,p(x)) => all(x,q(x))) R3(1.)
4. p(a) => q(a) R10(2.)
5. all(x,p(x)) R3(3.)
6. ~all(x,q(x)) R3(3.)
7. ~p(a) 8. q(a) R9(4.)
9. p(a) 10. p(a) R10(5.)
* 11. ~q(b) R13(6.)
12. p(b) => q(b) R10(2.)
13. ~p(b) 14. q(b) R9(12.)
15. p(b) * R10(5.)
*

The symbol * indicates a closed branch of proof.

The length of a proof depends on order of applications of the rules. The most effective proof can be obtained by priority application the rules of type A and if it not possible more, then we use other rules in order D, C, B.

Let us consider the formula used in example and let us construct the proof by mentioned process:

1. ~(all(x,p(x) => q(x)) => (all(x,p(x)) => all(x,q(x))))
2. all(x,p(x) => q(x)) R3(1.)
3. ~(all(x,p(x)) => all(x,q(x))) R3(1.)
4. all(x,p(x)) R3(3.)
5. ~all(x,q(x)) R3(3.)
6. ~q(a) R13(5.)
7. p(a) R10(4.)
8. p(a) = q(a) R10(2.)
9. ~p(a) 10. q(a) R9(8.)

The basic proof algorithm is not convenient for logic program construction, because multiple applications of rules of the type C on the same formula can account to an infinite computation. Above described optimized process is not convenient as well, because choice of rules by the types lengthens the computation nearly twice.

So the algorithm will be used in such modification, that it eliminates mentioned defects. But the modification is effective for monadic predicate calculus only - in full predicate calculus it would not be decision procedure yet. By point of view of effective computation the modification is a compromise between the length of a proof (a number of formulas - steps of proof is not minimal) and the length of computation The modification is based on these principles:

- every subformula of starting formula (i.e. negation of given formula) is decomposed step by step into atomic formulas or their negations
- this is performed in order, which is given by the rules

- only under this sequence (branch of proof) there is joined the sequence, that is relevant to paralel non-analysed formula
- rules of the type C are always applied with constant a
- rules of the type D respect branching of proof and apply succesively constants a, b, c,...,o.

Further we demand these limitations of the formulas form:

- bound variables are denoted by lower case characters x, y, z; they can be indexed
- formulas do not contain free variables, i.e. for example expression p(x) can occur as subformula only in expressions all(x,p(x)) or ex(x,p(x)); but it is not necessary, that in the scope of quantifier the variable quantified by them occurs, for example in expressions all(x,A) or ex(x,A) the subformula A does not have to contain x
- constants are denoted by lower case characters a,b,...,o
- if given formula contains constants, they occur there in above mentioned order form the left to the right beginning with the letter a
- usage of constants in formulas is not limited, but it is recommended.

Following logic program holds these conditions in its construction and function.

II. Logic program

```
/* Operations - logical connectives */
:- op(900,xfy,<=>)                    /* equivalence */
:- op(800,xfy,=>)                     /* implication */
:- op(700,xfy,#)                      /* disjunction */
:- op(600,xfy,&)                      /* conjuction */
:- op(500,fy,~)                       /* negation */
/* Regarding, testing and proving of formula */
formula:- repeat,nl,nl,write('Formula:'),nl,
          read (F),(F==stop;
          (testar(F),theorem(~F),fail)).
```

```
/* Arity testing */
testar(F):- ((arity(F),!);
             (nl,write('formula is not monadic'),fail).
arity(X):- functor(X,F,N),test(X,F,N).
test(X,~,1):- arg(1,X,Y),arity(Y).
test(X,Y,2):- (Y== &;Y== #;Y== =>;Y== <=>),
              arg(1,X,X1),arg(2,X,X2),arity(X1),arity(X2).
test(X,Y,2):- (Y== all;Y== ex),arg(1,X,X1),arg(2,X,X2),
              test(X1,X1,Ø),arity(X2).
test(X,Y,1):- test(Y,Y,Ø).
test(X,X,Ø).
/* Control of proof */
theorem(T):- nl,write('Main branch:'),
             (anal([T],[T],a),!,nl,nl,write('formula is theorem');
             nl,nl,write('formula is not theorem')).
/* Construction of analytic table */
anal([X|Y],Z,C):- nl,write(X),fail.
/* Conjunctive rules */
anal([~~X1|X2],Y,C):- append([X1],Y,T),
                      append(X2,[X1],Z),!,
                      anal (Z,T,C).                    /* R4 */
anal([X1 & X2|X3],Y,C):- append([X1,X2],Y,T),
                         append(X3,[X1,X2],Z),!,
                         anal(Z,T,C).                  /* R1 */
anal([~(X1 # X2)|X3],Y,C):- append([~X1,~X2],Y,T),
                            append(X3,[~X1,~X2],Z),!,
                            anal(Z,T,C).               /* R2 */
anal([~(X1 => X2)|X3],Y,C):- append([X1,~X2],Y,T),
                             append(X3,[X1,~X2],Z),!,
                             anal(Z,T,C).              /* R3 */
anal([X1 <=> X2|X3],Y,C):- append([X1 => X2,X2 => X1],Y,T),
                           append(X3,[X1=>X2,X2=>X1],Z),!
                           anal(Z,T,C).                /* R5 */
anal([~(X1<=>X2)|X3],Y,C):- append([~X1 => X2,X2 => ~X1],Y,T),
                            append(X3,[~X1 => X2,X2 => ~X1],Z),
                            anal(Z,T,C).               /* R6 */
/* Disjunctive rules */
anal([~(X1 & X2)|X3],Y,C):- append([~X1],Y,T1),
                            append([~X2],Y,T2),!,
```

```
                              v1,anal([~X1|X3],T1,C),!,
                              v2,anal([~X2|X3],T2,C).     /* R8 */
anal([X1 =# X2|X3],Y,C):- append([X1],Y,T1),
                          append([X2],Y,T2),!,
                          v1,anal([X1|X3],T1,C),!,
                          v2,anal([X2|X3],T2,C).          /* R7 */
 nal([X1 => X2|X3],Y,C):- append([~X1],Y,T1),
                          append([X2],Y,T2),!,
                          v1,anal([~X1|X3],T1,C),!,
                          v2,anal([X2|X3],T2,C).          /* R9 */
/* Universal rules */
anal([all(X,A)|Z],Y,C):- subst(a,X,A,U),
                         append([U],Y,T),!
                         anal([U|Z],T,C).                 /* R10 */
anal([~ex(X,A)|Z],Y,C):- subst(a,X,~A,U),
                         append([U],Y,T),!,
                         anal([U|Z],T,C).                 /* R11 */
/* Existential rules */
anal([~all(X,A)|Z],Y,C):- subst(C,X,~A,U),
                          exch(C,K),
                          append([U],Y,T),!
                          anal([U|Z],T,K).                /* R13 */
anal([ex(X,A)|Z],Y,C):- subst(C,X,A,U),
                        exch(C,K),
                        append([U],Y,T),!,
                        anal([U|Z],T,K).                  /* R12 */
/* Atomic formula */
anal([_|X],Y,C):- anal(X,Y,C).
/* End of analysis */
anal([ ],X, ):- scontr(X,X).
/* Substitution */
subst(N,P,P,N):- !.
subst(N,P,H,H):- atomic(H),!.
subst(N,P,H,D):- H=..[F|A],sarg(N,P,A,B),D=..[F|B].
sarg))_,_,[ ],[ ]):- !.
sarg(N,P,[A|B],[C|D]):- subst(N,P,A,C),sarg(N,P,B,D).
/* Exchange */
exch(C,K):- const(X),exc(C,K,X).
exc(C,K,[C,K|_]).
```

```
exc(C,K,[_|X]):- exc(C,K,X).
const([a,b,c,d,e,f,g,h,i,j,k,l,m,n,o]).
append([ ],L,L).
append([H|T],L,[H|U]):- append(T,L,U).
/* Searching of contradiction in actual branch */
scontr([ ],_):- fail.
scontr([X|Y],Z):- (contr(X,Y),nl,write('branch closed'));
                  scontr(Y,Z).
contr(X,[~X|_]).
contr(~X,[X|_]).
contr(X,[_|Y]):- contr(X,Y).
v1:- nl,nl
     write('1. branch').
v2:- nl,nl
     write('2. branch').
```

IV. Execution examples

Mentioned logic program can prove self-evidently any theorem of propositional calculus. Examples of such proofs can be seen in [2]. Here we show only the proofs of formulas of monadic predicate calculus.

```
Formula:
ex(y,ex(x,p(x)) => p(y)).
Main branch:
~ex(y,ex(x,p(x)) => p(y))
~(ex(x,p(x)) => p(a))
ex(x,p(x))
p(a)
~p(a)
branch closed
     formula is theorem

Formula:
(all(x,p(x)) # all(x,q(x))) => all(x,p(x) # q(x)).
Main branch:
~((all(x,p(x) # all(x,q(x))) => all(x,p(x) # q(x)))
all(x,p(x)) # all(x,q(x))
```

```
1. branch
all(x,p(x))
p(a)
~all(x,p(x) =# q(x))
~(p(a) =# q(a))
~p(a)
~q(a)
branch closed
2. branch
all(x,q(x))
q(a)
~all(x,p(x) =# q(x))
~(p(a) =# q(a))
~p(a)
~q(a)
branch closed
     formula is theorem
```

These examples demonstrate objectively the function of the program. Now let us show another usage of program as decision procedure by example of converse implication of last formula.

```
Formula:
all(x,p(x) =# q(x)) => (all(x,p(x)) =# all(x,q(x))).
Main branch:
~(all(x,p(x) =# q(x)) => (all(x,p(x)) =# all(x,q(x))))
all(x,p(x) =# q(x))
p(a) =# q(a)
1. branch
p(a)
~(all(x,p(x) =# all(x,q(x)))
~all(x,p(x))
~p(a)
~all(x,q(x))
~q(b)
branch closed

2. branch
q(a)
~(all(x,p(x)) =# all(x,q(x)))
```

~all(x,p(x))
~p(a)
~all(x,q(x))
~q(b)
formula is not theorem

V. Evaluation of logic program

Logic program can be used as standard decision procedure. In the majority of cases it is difficult to watch the proof protocol (i.e. analytical table). Proofs of more complicated formulas are rather long, so they exceed the range of the screen and the program operates fast. To make possible the watching of proof protocol we can use the interruption of execution or to print the protocol. As it is evident from program and from examples the branches in protocol are signed only as the first one and the second one and it is not possible to demonstrate the branching visually. We remind (for easier orientation in listing), that corresponding assignment is realized on the LIFO-principle.

REFERENCES

[1] C o e l h o, H. and C o t t a, J.C.: Prolog by example, Springer-Verlag, Berlin Heidelberg, 1988.

[2] Š t ě p á n, J.: Propositional Calculus Proving Methods in Prolog, Acta UPO 97 (1990), (to appear).

[3] S m u l l y a n, R.M.: First Order Logic (Slovak), Alfa, Bratislava, 1979.

Department of Computer Science
Palacký University
Vídeňská 15, 771 46 Olomouc
Czechoslovakia

Acta UPO, Fac.rer.nat. 100, Mathematica XXX (1991), 273 - 283.