J. Indian Inst. Sci. 62 (A), Mar 1980, Pp. 71–81

A computational algorithm for the verification of tautologies in propositional calculus

S. V. RANGASWAMY*, N. CHAKRAPANI* AND V. G. TIKEKAR**
Indian Institute of Science, Bangalore 560 012, India.

Received on March 7, 1980.

Abstract

A computational algorithm (based on Smullyan's analytic tableau method) that verifies whether a given well-formed formula in propositional calculus is a tautology or not has been implemented on a DEC System 10. The stepwise refinement approach of program development used for this implementation forms the subject matter of this paper. The top-down design has resulted in a modular and reliable program package. This computational algorithm compares favourably with the algorithm based on the well-known resolution principle used in theorem provers.

Key words : Tautology, propositional calculus, analytic tableau, top-down design, stepwise refinement, Pascal.

1. Introduction

A proof procedure, due to Smullyan[1], that tests whether a given well-formed formula (wff) in Propositional Calculus (PC) is a tautology is known as the analytic tableau method. This paper presents the design and implementation details of a computational algorithm based on the analytic tableau method. The program is developed using the top-down stepwise refinement technique[2, 3, 4] and is coded in Pascal[5]. The program is implemented on a DEC System 10. It comprises 15 procedures and has about 600 lines of Pascal code with a memory requirement of 5K words. The readability and modularity of the program have been greatly enhanced due to the structured programming approach followed in its design.

* School of Automation
** Department of Applied Mathematics

2. Tableau method

The analytic tableau method, a variant of the semantic tableau method of Beth[1], is an elegant and efficient proof procedure for PC. The analytic tableau for a given wff is an ordered dyadic tree, whose root is the negation of the given wff which is to be tested for its validity. For explaining the method of generation of the tableau for a wff requires the notions of subformulas, conjugates,and signed formulas. We briefly present the definitions of these notions in the sequel.

2.1. *Definitions*

(*a*) *well-formed formula (wff)*

A wff in PC is defined recursively as follows :

(i) A propositional variable is a wff, also referred to as an atomic wff.

(ii) If A is a wff, so is $\sim A$.

(iii) If A, B are wffs, so is (AbB), where b is a binary connective.

(*b*) *immediate subformula*

(i) Atomic formulas have no immediate subformulas.

(ii) $\sim X$ has X as an immediate subformula and no others.

(iii) XbY, where b is a binary connective, has X, and Y as immediate subformulas and no others.

(*c*) *subformula*

Y is a subformula of Z if and only if there exists a finite sequence starting with Z and ending with Y such that each term of the sequence except the first is an immediate subformula of the preceding term.

(*d*) *signed formula*

A signed formula is an expression TX or FX, where X is a formula. We normally read TX as 'X is true' and FX as 'X is false'.

(*e*) *conjugate*

The conjugate of a signed formula is the formula obtained by changing the sign. Thus the conjugate of TX is FX, and the conjugate of FX is TX.

2.2. *Analytic tableau*

An analytic tableau for a wff X is an ordered dyadic tree, whose nodes are formulas and whose root is X. The tableau is generated using one of the following four pairs of

rules :

1 (i) If $\sim X$ is true, then X is false.

(ii) If $\sim X$ is false, then X is true.

2 (i) If $X \wedge Y$ is true, then both X and Y are true.

(ii) If $X \wedge Y$ is false, then either X is false or Y is false.

3 (i) If $X \vee Y$ is true, then either X is true or Y is true.

(ii) If $X \vee Y$ is false, then both X and Y are false.

4 (i) If $X \Rightarrow Y$ is true, then either X is false or Y is true.

(ii) If $X \Rightarrow Y$ is false, then simultaneously X is true and Y is false.

The above mentioned rules may be represented in signed formula notation as follows :

1. (i) $\dfrac{T \sim X}{FX}$ (ii) $\dfrac{F \sim X}{TX}$

2. (i) $\dfrac{TX \wedge Y}{\begin{array}{l} TX \\ TY \end{array}}$ (ii) $\dfrac{FX \wedge Y}{FX \mid FY}$

3. (i) $\dfrac{TX \vee Y}{TX \mid TY}$ (ii) $\dfrac{FX \vee Y}{\begin{array}{l} FX \\ FY \end{array}}$

4. (i) $\dfrac{TX \Rightarrow Y}{FX \mid TY}$ (ii) $\dfrac{FX \Rightarrow Y}{\begin{array}{l} TX \\ FY \end{array}}$.

If at any point in the tableau, a formula of the form given in the numerator of any rule appears, then the tableau can be extended on that particular branch by the formula(s) shown in the denominator. In case the denominator contains the '|' symbol, then it is an indication of the fact that the tableau has a branch at that point. The tableau is extended by repeated applications of the rules until no more extensions are possible. A branch of a tableau is closed if and only if it contains some signed formula and its conjugate. The tableau is said to be closed if and only if every branch in it is closed. A proof of an unsigned wff X in the system corresponds to showing that there exists a closed tableau for FX.

The method of proof employing the analytic tableau is shown to be both consistent and complete by Smullyan[1]. The system is consistent since any formula provable by the tableau method is a tautology and the root of any closed tableau is unsatisfiable. The system is complete since for every tautology X there exists a closed tableau with root FX.

3. Program implementation

The program is developed using the top-down stepwise refinement technique[2-4]. The program logic and the various data structures used in these procedures are presented in the sequel*. The problem statement is identified as step 1.

Step 1

Develop a program in Pascal to test whether a given wff in PC is a tautology, using the analytic tableau method.

A given formula has to be tested for its well-formedness, before we can verify whether it is a tautology or not; it may contain operators other than *and, or, not,* and *implies* and hence we need to rewrite the formula into an equivalent form containing only these four operators. These two requirements suggest the step 2 as a refinement of step 1.

Step 2

begin

2.1 Preprocess the input to enable further processing

2.2 Analyse the given wff using the tableau method to verify its validity.

end

The method of analytic tableau successively splits a given formula into its subformulas in an effort to come up with a contradiction. At every stage of computation, it becomes necessary to locate the primary operator in the formula in order to split the given formula into its constituent subformulas. For ease and elegance of computation, the well-known postfix Polish notation is better suited than the conventional infix notation.[9] This consideration along with the need to check the well-formedness of the formula corresponds to the actions in step 2.1. Step 3 presents these details as a refinement of step 2.

Step 3

begin

3.1 Accept a formula from input device.

3.2 *If* operators other than *and, or, not,* and *implies* appear in input *then* reduce the formula into an equivalent formula containing only these four operators.

3.3 convert the formula into postfix Polish notation and check for the well-formedness of input.

* The notation used in stepwise refinement utilizes Pascal-like constructs for depicting control flow.

3.4 Generate the tableau with the negation of the given formula at its root;
if all branches close
then given input wff is a tautology
else given input wff is not a tautology;

end

The steps 3.1 through 3.4 correspond to the preprocessing step and the specifications for these steps are complete. Each of these steps has been coded as a Pascal procedure.

The procedure for input accepts a given wff from a terminal as a string of characters. Every character read from the terminal is appended to a string named ' formula ' in the procedure. The end of input is signalled by a blank character. The input wff is restricted to a maximum length of 80 characters. Any error conditions detected during input are reported to the user.

The input wff in the array ' formula ' is passed on to the procedure that tests whether the given formula contains the *equivalence*, *nand*, or *nor* operators. The computations in this procedure are presented in step 4 which is a refinement of step 3.2†.

Step 4 {refinement of step 3.2}

begin

for every operator in the formula *do*
case operator *of*

$\Leftrightarrow$: rewrite $(\sim \text{opd1} \vee \text{opd2}) \wedge (\sim \text{opd2} \vee \text{opd1})$;
$\uparrow$: rewrite $\sim (\text{opd1} \wedge \text{opd2})$;
$\downarrow$: rewrite $\sim (\text{opd1} \vee \text{opd2})$;
others : skip

end
end

The wff equivalent to the given wff generated in the previous step is processed by the postfix procedure to yield the postfix version of the wff. Conversion to postfix from infix notation is done using the ' shunting yard model ' algorithm due to Dijkstra. This postfix version of the wff is tested to ensure the well-formedness of the given input. The logic of these procedures is presented in step 5.

† The stepwise refinement steps (step 1, step 2, . . .) correspond to completely refined versions. In our presentation, to avoid writing down the same details, in successive steps, we provide only the newly refined section. *E.g.*, step 4 is a refinement of step 3·2; this means that steps 3·1, 3·3 and 3·4 remain unchanged.

```
Step 5 {refinement of step 3.3}
    begin
      for every character in the formula do
        begin if (character = operator)
          then begin
               while topstack operator priority ≥ current operator priority
               do push topstack operator to output;
               push current operator into stack
               end
          else begin
               if (character = '(')
               then increase priority of all operators by the standard value
               else
               if (character = ')')
               then decrease priority of all operators by the standard value
               else push in ouput area
          end;
          if stack not empty
          then pop contents of stack and push them into output
        end;
        for every character in the formula do
        case character of
            Operands : associate weight – 1;
            Unary operator : associate weight 0;
            Binary operator : associate weight 1
            end;
        if sum of weights = – 1
        then formula is well-formed
        else formula is not well-formed;
    end.
```

This completes the preprocessing of input. Step 3.4 pertains to the processing based on the tableau method. Before we embark on the stepwise refinement of this step further, we need to decide about the data structures and control flow organization. The tableau is an ordered dyadic tree and a path in the tree consists of an ordered set of formulas. The method requires the testing of every path to find out whether it is closed and hence there is no need to maintain the complete tableau in the memory. For reasons of efficient core utilisation, only one path of the tableau is maintained at a time in the memory. An array is used to hold all the formulas in a path in contiguous locations as the processing on these formulas is strictly sequential. An array of pointers to this 'formula array' is maintained to identify the beginning of every formula in this data structure. To enable the generation of a path in the tableau in a systematic fashion, an array indicating the parent formula of a given sub-

formula is also maintained. The rules defining the construction of the tableau are coded as individual procedures. The main procedure is recursively invoked to enable the complete generation of a path. Once a path is completely constructed, another procedure scans all the signed atomic formulas to determine whether the path is closed. The main procedure provides the necessary pointers to access atomic formulas in the array, thereby avoiding searches to locate them. If a path is closed, the main procedure tests whether there are any unexplored paths in the tableau and if so, proceeds with the next path. A stack of branch pointers is maintained so that the various paths in the tableau are scanned in a left-to-right manner. If any open path (a path that is not closed) is detected, the processing is terminated with a message that the given formula is not a tautology. If all the paths are closed in the tableau, then the program declares that the given formula is a tautology. With the above mentioned control strategy and data structure organization as the base, we present below the refinement of step 3.4 in steps 7 and 8.

```
Step 7
   begin prefix an F to the wff in postfix notation.
    repeat
      repeat
        for every formula do
          case primary operator of
            not : change sign of formula;
            and : separate left operand (lhs) and right operand (rhs); 'andprocess';
            or : separate lhs and rhs; 'orprocess';
            implies : separate lhs and rhs; 'implyprocess'
          end;

      until path is complete;
      collect all atomic formulas in the path;
      if both TX and FX exist in this collection
      then declare path closed
      else declare path is open and set flag;
    until nomorepaths or flagset;
          if noflagset
          then declare given wff is a tautology
  end

Step 8 {refinement of 'andprocess'}
   begin
     if sign is T
       then extend current path with two subformulas T(lhs) and T(rhs)
       else begin
            set a flag to indicate branch at that point and stack the pointer;
            if first traversal then extend current path with F(lhs)
```

```
                    else begin
                          extend current path with F(rhs);
                          pop up branch stack
                         end
            end
    end
```

The refinements of steps ' orprocess ' and ' implyprocess ' are similar to step 8.

4. Illustrative examples

We provide below sample outputs generated by the program for two specific inputs. Every path is listed starting from the origin, for easy perusal; the actual implementation does not generate every branch from the origin. The first input wff is a tautology and the second input wff is not a tautology†.

EXAMPLE 1

input : (PA (QKR)) C ((PAQ)K(PAR))

output :

```
GIVEN FORMULA IS WELL-FORMED
    FPQRKAPQAPRAKC
    FPQAPRAK
    TPQRKA
    FPQA
    TP
    FP
    FQ

BRANCH CLOSED
    FPQRKAPQAPRAKC
    FPQAPRAK
    TPQRKA
    FPQA
    TQRK
    FQ
    FP
    TQ
    TR
```

† For purposes of computer implementation A, K, N, C are used to represent the operators *or*, *and*, *not*, and *implication* respectively.

```
BRANCH CLOSED
   FPQRKAPQAPRAKC
   FPQAPRAK
   TPQRKA
   FPRA
   TP
   FP
   FR

BRANCH CLOSED
   FPQRKAPQAPRAKC
   FPQAPRAK
   TPQRKA
   FPRA
   TQRK
   FP
   FR
   TQ
   TR

BRANCH CLOSED
GIVEN FORMULA IS A TAUTOLOGY
EXAMPLE 2
input :  (PAQ)C(PKQ)
output :
GIVEN FORMULA IS WELL-FORMED
   FPQAPQKC
   FPQK
   TPQA
   FP
   TP
BRANCH CLOSED
   FPQAPQKC
   FPQK
   TPQA
   FP
   TQ
BRANCH OPEN
HENCE NOT A TAUTOLOGY
```

5. Remarks and conclusions

This computational algorithm has not so far been used in theorem-provers; most of the well-known implementations of theorem-provers make use of the resolution

principle[7]. The resolution-based systems basically require that the wff be presented in clause form; normally a wff is not available in clause form and the task of converting a wff into clause form is by itself of exponential complexity. The analytic tableau method does not have any such special input requirements.

The resolution method requires that the empty clause be generated from the given clauses by successively resolving them. As there is no algorithmic way to decide which of the given clauses yield the empty clause by resolving, refinements to resolution by means of heuristics aim at reducing the searches through blind alleys. In contrast, the analytic tableau method, essentially breaks a formula into its constituents (subformulas); the process is a systematic one and terminates much faster compared to the resolution technique.

The method of Davis and Putnam[8] is another method used in one of the efficient automatic theorem-provers. This method is based on the four rules—tautology rule, one-literal rule, pure literal rule and splitting rule. This method also requires that the given formula be in conjunctive normal form so that any of the four rules may be applied. In the method of analytic tableau there is no need for a given wff to be in conjunctive normal form. Also at each step of the generation of the tableau, the principal connective of the signed formula of a node enables the determination as to which of the eight rules is to be applied; in the method of Davis and Putnam an analogous determination as to which of the four rules is to be applied is not possible.

The implementation of the tableau method ensures that the number of branches of the generated tableau is minimum by judiciously ordering the subformulas generated at each step. The depth-first approach of generating each path in the tableau, in the left-to-right order, results in reduced main storage requirements for the program. Each rule of this method has been coded as a procedure resulting in a neatly structured program enhancing its readability and modularity.

6. Acknowledgements

We express our gratitude to Professor R. Narasimhan, Director, National Centre for Software Development and Computing Techniques, Bombay, for providing the necessary computational facilities to enable this implementation.

References

1. Smullyan, R. M. *First-order logic*, Springer-Verlag, 1968.

2. Wirth, N. Program development by stepwise refinement, *CACM*, **14**, 4, 1971, 221–227.

3. Dijkstra, E. W. Notes on structured programming, in *Structred programming* by Dahl, Dijkstra, and Hoare, Academic Press, 1972.

4. KIEBURTZ, R. B. *Structured programming and problem solving with Algol–W*, Prentice-Hall, 1975.

5. JENSEN, K. AND WIRTH, N. Pascal : User Manual and Report, *Lecture Notes in CS*, **18**, Springer-Verlag, 1974.

6. BETH, E. W. *The foundations of mathematics*, North Holland, 1959.

7. ROBINSON, J. A. A machine-oriented logic based on the resolution principle, *JACM*, **12**, **1**, 23–41.

8. DAVIS, M. AND PUTNAM, H. A computing procedure for quantification theory, *JACM*, **7**, 1960, 201–215.