Yugoslav Journal of Operations Research
17 (2007), Number 2, 275-285
DOI: 10.2298/YUJOR0702275M

RESOLUTION METHODS IN PROVING THE PROGRAM CORRECTNESS

Branko MARKOSKI
Faculty of Technical Sciences,
University of Novi Sad, Serbia
markoni@uns.ns.ac.yu

Petar HOTOMSKI
Technical Faculty "Mihajlo Pupin", Zrenjanin
University of Novi Sad, Serbia
hotomski@ptt.yu

Dušan MALBAŠKI
Faculty of Technical Sciences,
University of Novi Sad, Serbia
malbaski@uns.ns.ac.yu

Danilo OBRADOVIĆ
Faculty of Technical Sciences
University of Novi Sad, Serbia
dacha@uns.ns.ac.yu

Received: September 2003 / Accepted: September 2007

Abstract: Program testing determines whether its behavior matches the specification, and also how it behaves in different exploitation conditions. Proving of program correctness is reduced to finding a proof for assertion that given sequence of formulas represents derivation within a formal theory of special predicted calculus. A well-known variant of this conception is described: correctness based on programming logic rules. It is shown that programming logic rules may be used in automatic resolution procedure. Illustrative examples are given, realized in prolog-like LP-language (with no restrictions to Horn's clauses and without the final failure). Basic information on LP-language are also given. It has been shown how a Pascal-program is being executed in LP-system proffer.

Keywords: Test, specification, resolution, program correctness.

1. INTRODUCTION

Derivations in formal theories are the general framework for development of deductive methods for proving program correctness. This framework gives two basic methods (refuting associated predicate formula, and use of programming logic rules), as well as their modifications.

Working with a formula associated to a given program requires the presence of additional axioms. Additional axioms describe properties of domain predicates and operations, giving necessary knowledge to present by deductive system.

Proving program correctness and designation of correct programs are related theoretical problems with large practical significance. The first is solved in frame of program analysis and the second in frame of synthesis of program, because of connection of analyses problematic and synthesis of program we can notice interlace between each other of these processes. However when we speak of automatic methods of proving correctness and about methods of automatic synthesis of program, difference between them is clear. In the N. Nilson's book (Nil, 1971), is described the initial possibility of automatic synthesis simple programs in assistance of resolution procedure of automatic proving theorem (ADT) or more exactly in assistance of resolution procedure of deduction answer on question. By proof that question in shape $(\exists x)W(x)$ is a logical consequence of axioms which define the predicated W and which define (elementary) program operators ensuring that variable x in answer gets value which represents composition of (elementary) operators, in fact demanded program.

In Manna [14, 15], resumed in [4], problems of program analysis and synthesis, using resolution procedure of proving and deduction of answer are discussed in detail. Different direction of investigation is axiomatic definition of programming language semantics (for example Pascal) in the form of special rules of programming logic derivation, as shown in [2], [5], [8], [9]. This approach enables deduction on the basis of programming logic rules, using accordance confirmation of concrete input-output predicates with deduced values. The marked laterals technique enables to find out conditions for refuting, which need not be known in advance. Although both presented approaches has common characteristic they are essentially different in concept. It is a deductive system on predicated language. In fact it is elaboration in special predicated calculation which is based on deduction in formal theory. In that way the problem of correctness of program is brought in connection with automatic revision (existed) proves of mathematics theorems. From that concept proceed both presented approaches as well as their modifications.

The first of these programs was LT (Logic Theory Machine), in 1957, written by Newell, Shaw, and Simon. This program was proving theorems of statement calculus and was tested on theorems from Mathematics principia. Only a year later, the Wang Hao program was able to prove all theorems from Mathematics principia in only 3 minutes. This program has proven about 200 theorems. The interactive program based on principle of automatic theorem proving, written by Green and Raphael (1968) represents further progress in this area. Other important use of resolution, connected to program correctness proving and with automatic programming, is present in papers by: Waldinger R. J., Lee C. T. (1969), Manna Z., Waldinger R. (1971) op. cit. [20].

Introduction of semantic information and creation of possibility to intervene outside in proving process enables further increase of proving programs' efficiency. In this area papers of Henschen (1974) were especially important. On the other hand, improvement of efficiency is accomplished by incorporation of specific properties of concrete theories into rules of derivation and unification algorithms. In this area, results of Slagle J. (1972, 1974) are especially important. Axioms of second order theory were used by J. L. Darlington (1968).

The aim of automatic theorem proving is to design and implement computer programs that can prove or help prove a theorem.

Programs for automatic proving of theorems could be divided as follows:

- Independent from human participation, or purely automatic ones: program is independently proving theorem, if and when it succeeds;
- Interactive proffers, where programs find parts of the proof.

In the beginning, programs for theorem proving were implemented only in mathematics. When it was realized that other problems could be presented as theorems which needs to be proven, application possibilities were found for areas such as program correctness, program generating, query languages over relational databases, electronic circuits design an so on.

As for formal presentation where theorem is being proven, it could be propositional logic, first-order predicate calculus, as well as higher-order logic. Theorems in propositional logic are easy to deal with for contemporary proffers, but propositional logic itself is not expressional enough. Higher-order logic is highly expressional, but it introduces a number of practical problems. Therefore, first-order predicate calculus appears to be the most appropriate when dealing with practical problems.

Regarding techniques of automatic theorems proving, most investigations have been done in resolution rules of derivation. Resolution is a very important derivation rule with the completeness property.

This paper is based on BASELOG –system, described in [3], which is used for automatic proving of program structure correctness. The BASELOG-system was implemented at Technical Faculty "Mihajlo Pupin", Zrenjanin. Basic property of BASELOG is that there is no limitation to Horn's clauses and presence of CWA-controllers. Linear resolution with marked literals is a deductive basis of this system, while CWA controller enables choice of working regime in closed, open or partially closed/open environment. This enables behavior of proving system as a classical Prolog-system (closed environment), or as classical ADT-system (open environment), or as combination of these, depending on declared CWA predicates list.

Management of operating mode was achieved at the predicate level, depending on semantic completeness of basis contents. Shortages of PROLOG and DATOLOG, manifesting in semantically incomplete bases, connected to negation problem and CWA-principle, were eliminated.

2. DEDUCTIVE TESTING OF PARTIAL CORRECTNESS

The original rules for manual design and manual confirmation of program correctness ([1], [6], [19]), do not imply the possibility of automatic (resolution) confirmation methods. Basic relation: $\{P\}S\{Q\}$represent specification of program S with next meaning: If predicated P in input is correct before execution of program S, then the predicated Q on exit is correct after execution of program S. If we wish to prove the correctness of program S, we must prove predicate {P}S{Q},defined as (if P is true and if S terminates, then Q becomes true) where the current values of program variables must satisfy the formula P and output values of variables must satisfy the formula Q. This defines so-called partial correctness of program S, since it is assumed that program S terminates. If we prove that S terminates and that predicate {P}S{Q} is true, we say that S is totally correct. Thus, designated principle of correctness is used for program design. Designation starts from specification {P}S{Q} with given precondition P and given resulting post condition Q. Process of projecting is divided on under specification type {Pi}Si{Qi} from component Si from which is program constructed. On that way, basic idea is that projecting of program goes in the same time with proof of correctness of published specifications. Then the special rules of deduction ensure proof that fulfillment of relation {Pi}Si{Qi} comes from fulfillment f relation {Pi}Si{Qi} for component program Si. Rules of execution are marked for simile and complicated programs operators. Formula {P}S{Q} is written as K(P, S, Q), where K is a predicate symbol and P,S,Q are variables of first-order predicate calculus.

$\{P^z{}_y\}$ z := y {P}	we represent by K(t(P,Z,Y), d(Z,Y), P)...
	where t,d are function symbols and P,Z,Y are variables;

Rules R(τ):

P1. $\dfrac{\{P\}S\{R\}\ ,\ R\Rightarrow Q}{\{P\}\ S\ \{Q\}.}$	we represent by K(P,S,R) ∧ Im(R,Q) ⇒ K(P,S,Q) where Im (implication) is a predicate symbol, and P, S, R, Q are variables;
P2 $\dfrac{R\Rightarrow P,\ \{P\}S\{Q\}}{\{R\}\ S\ \{Q\}.}$	we write Im(R,P) ∧ K(P,S,Q) ⇒ K(R,S,Q)
P3 $\dfrac{\{P\}S1\{R\}\ ,\ \{R\}S2\{Q\}}{\{P\}\ S1;\ S2\ \{Q\}}$	K(P,S1,R) ∧ K(R,S2,Q) ⇒ K(P,s(S1,S2),Q) where s is a function symbol, and P, S1, S2, R, q are variables
P4 $\dfrac{\{P\wedge B\}S1\{Q,\ \{P\wedge \sim B\}S2\{Q\}}{\{P\}\ \text{if B then S1 else S2}\ \{Q\}}$	K(k(P,B),S1,Q)∧K(k(P,n(B)),S2,Q) ⇒ K(P,ife(B,S1,S2),Q)
	where k, n, ife are function symbols
P5 $\dfrac{\{P\wedge B\}S\{Q\}\ ,\ P\wedge \sim B\Rightarrow Q}{\{P\}\ \text{if B then S}\{Q\}}$	K(k(P,B),S,Q) ∧ Im(k(P,n(B)),Q) ⇒ K(P,if(B,S),Q)
	where k, n, if are function symbols
P6 $\dfrac{\{P\wedge B\}\ S\quad \{P\ \}}{\{P\}\ \text{while B do S}\ \{P\wedge \sim B\}}$	K(k(P,B),S,P) ⇒ K(P,wh(B,S),k(P,n(B)))
	where k, n, wh are function symbols

P7 $\underline{\{P\}S\{Q\}\ ,\ Q\wedge\sim B \Rightarrow\ P}$ $K(P,S,Q) \wedge Im(k(Q,n(B)),P) \Rightarrow K(P,ru(S,B),k(Q,B))$

$\{P\}$ repeat S until B where k, n, ru are function symbols

$\{Q\wedge B\}$

Transcription of other programming logic rules is also possible.

Axiom $A(\tau)$:

A1 $K(t(P,Z,Y),d(Z.Y),P)$ assignment axiom

Formal theory τ is given by $(\alpha(\tau), F(\tau), A(\tau), R(\tau))$, where α is a set of symbols (alphabet) of theory τ, F is a set of formulae (correct words in alphabet α), A is a set of axioms for theory $\tau(A\subset F)$, R is a set of derivation rules for theory τ.B is a theorem within theory τ if and only if B can be derived within calculus k from set $R(\tau) \cup A(\tau)$ (k is a first-order predicate calculus). Let S be special predicate calculus (first-order theory) with it's own axioms $A(S) = R(\tau) \cup A(\tau)$. This means that derivation of theorem B within theory τ could be replaced by derivation within special predicate calculus S, with own axioms $A(S)= R(\tau) \cup A(\tau)$. Axioms of special predicate calculus S are: $A(S)= A(\tau) \cup R(\tau)$.We assume that s is a syntax unit whose (partial) correctness is being proven for certain input predicate U and output predicate V.

Within theory S is being proved $\vdash_S (\exists P)(\exists Q)K(P,s,Q)$ where S is a constant for presentation of a given program. Program is written using functional notation with symbols: s (sequence), d (assignment), ife (if-then-else), if (if-then), wh (while), ru (repeat-until). To the initial set of axioms A(S), negation of statement is added. The result of negation using resolution procedure is as follows: $/Im(X_\theta,Y_\theta,)\vee$ answer(P_θ,Q_θ), where $X_\theta,Y_\theta,P_\theta,Q_\theta$ are values for which successful negation of T_θ means that for these values a proof is found. But this does not mean that given program is partially correct. It is necessary to establish that input and output predicates U, V are in accordance with P_θ, Q_θ, and also that Im (X_θ,Y_θ) is really fulfilled for domain predicates ant terms. Accordance means confirmation that is valid. : $U \Rightarrow P_\theta,\ \wedge\ Q_\theta \Rightarrow V) \wedge (\ X_\theta \Rightarrow Y_\theta)$.There are two ways to establish the accordance: manually or by automatic resolution procedure but their realization is not possible within theory S. However, it becomes possible within the new theory, which is defined by predicates and terms that are part of the program s and input-output predicates U, V. Within this theory U, P, Q, V, X, Y are not variables, but formulae with domain variables, domain terms and domain predicates. This method is related to derivation within special predicate calculus based on deduction within the formal theory. Thus the program correctness problem is associated with automatic proving of (existing) proofs of mathematical theorems.

3. FORMALIZATION OF PROGRAMMING LOGIC RULES ON THE BASELOG SYSTEM

This section shows how programming logic rules look within BASELOG system: rules and axioms are translated in component form. These rules and axioms have a status of sequence elements of AKS within BASELOG system. Sequence BAKS (where BAKS represents elements of database) is empty, since it is not important for our needs, so the number is zero. CWA predicate is not needed, so their number is also zero. In this way, initial set of formulae for BASELOG system operation consists of:

- Components originating from statement negation, with their number stated
- Elements of AKS sequence
- Zero elements of BAKS sequence
- Zero elements of CWA predicates.

The P1 rule is an example.
K(P,S,R) $\wedge$ Im(R,Q) $\Rightarrow$ K(P,S,Q) is transformed into:
~K(P,S,R) $\vee$ ~IM(R,Q) $\vee$ K(S,P,R). Symbols P, S, R, and Q are replaced by symbols X1, Y1, Z1, X2 and a disjunction symbol is omitted. In this way, programming logic rules within BASELOG system are as follows:

P1. ~K(X1,Y1,Z1)~IM(Z1,X2)(K(X1,Y1,X2).

P2. ~IM(X2,Y1)~K(Y1,U0,V1)K(X2,U0,V1)& /consequence rule

P3. K(X1,Y1,U1)~K(U1,Y2,V1)K(X1,s(Y1,Y2),V1)& / sequential rule

P4. ~K(k(X1,Y1),Z1,U1)~K(k(X1,u(Y1)),Z2,U1)K(X1,ife(Y1,Z1,Z2),U1) /ife rule

P5. ~K(k(X1,Y1),Z1,U1)~IM(k(X1,n(Y1)),U1)K(X1,if(Y1,Z1),U1)& /if rule

P6. ~K(k(X1,V2),U0,X1)K(X1,w(V2,U0),k(X1,n(V2)))& / while rule

A1. Assigning axioma

K(t(X1,Y1,Z1),d(Y1,Z1),X1

If resolution procedure generate refutation then comes next result: /Im(X_θ,t(Y_θ,Z_θ,T_θ))$\vee$ Answer (P_θ,Q_θ),where X_θ,Y_θ,Z_θ,T_θ,P_θ,Q_θ are instanced values for which is realised refutation in fact for which the proof is found.

Marked literals which are related on prediction of implication IM collect in themselves instanced values of variables which are marked in process of searchinig of refutation. The number of these different literals also is not known before realisation of refutation. Because of that in this lecture is realised modification of metod of resolutions refutation. Meaning of that modification is t ensure collection of all necessary literals connected with implication at the beginning of resolvement. In that way refutation is ended when there is no more predicate K in resolvement (all the predicates IM are marked). That is cleared in the last step and the standard shape of resolutions refutation with empty connection on end is realised.

To avoid marking (in principe we don't know how many are of them) we have to execute change in program and in that way save all IM on ending level.

Because of that the change is executed in resolvent procedure. The sense of it is to enable resolution to predicate K and to save all IM predicates with instanced values for which have to be proved agreement. It means that during the proving of correctness of program IM have to be always on first place.

4. CONCLUSION

This paper describes general framework for investigation of program correctness investigation by resolution negation method. The basis for development of basic deduction methods and their modifications, used for establishing program correctness (use of programming logic rules and negation of associated predicate formula) are derivations within formal theories. One of the ways to solve problem of program correctness is to use programming logic rules. This enables deduction on this basis, as well as checking the accordance of concrete input-output predicates with deducted values. It is shown how programming logic rules look within BASELOG system. It is also shown how the rules and axioms are translated into component form. Marked literals technique enables insight in conditions for negation, which may not be known before negation. Efficiency of automatic procedures is especially important, as well as definition of complexion limits of programs for practical use of deductive concept. These results and examples should induce further theoretical and experimental investigations in this field.

Execution in formal theories represent the basic framework for development of deductive methods of checking of correctness of the program. Out of that frame come two basic methods (refutation associated predicated formulae) as well as their modification. Work with formulae which are associated to given program assumes the presence of additional axioms without which the refutation can not be realized. Additional axioms describe characteristics of domain predicates and operations which represent necessary knowledge which have to be shown in deductive system. Existed theoretical results we described assume that knowledge but in practical sense that appears as significant difficulty.

Another approach enables deduction based on the rule of logical program to check agreement of correct input-output predicates with deductive values. The techniques with marked literals enable us to learn conditions (which do not have to be known beforehand) for realization of refutation. There is a sense of investigation in both directions and looking for new modifications of basic ideas. We have demonstrated how the Pascal program is executed in resolution proofer of LP system. That is principal base for development of pro logicical languages of logical programming. It is especially an important question of efficiency of automatic procedures as well as determination of complexity of the program for practical use of deductive concept.

Presented results and examples should be considered as a support for further theoretical and experimental examinations in this important field of science.

5. REFERENCES

[1] Alagić, S., Arbib, M., *Proektirovanie korrektnyh strukturirovannyh programm*, Moskva, 1994.

[2] Basili, V.R., and Selby, R.W., *Comparing the Effectiveness of Software Testing Strategies, Components*, a publication of the IEEE computer society 1993.

[3] Berković, I., "The deductive bases for development of the descriptive language for the logical programming", Ph.D. Thesis, University of Novi Sad, 1997.

[4] Čubrilo, M., Mathematical logic for expert's systems, 1989

[5] Dahl, O.J., Dijkstra, E.W., and Hoare, C.A.R, *Structured Programming*, Academic Press, 1972, 83-174.

[6] Dijkstra, E.W., "A Discipline of Programming, Prentice-Hall", 1993.

[7] Glesner, S., "Using program checking to ensure the correctness of compiler imlementation", *Journal of University Comp. Sciences*, 9(3) (2003) 191-222.

[8] Hoare, C.A.R., "An axiomatic basis for computer programming", *Communications of the ACM 12* (1973) 576-583.

[9] Hoare, C.A.R., and Wirth, N., "An axiomatic definition of the programming language pascal", *Acta Informatica*, 2 (1973) 335-355.

[10] Hotomski, P., and Berković, I., "Symbolical execution of Pascal programs in theorem prover of Baselog system", Tara 2002.

[11] Jazayeri, M., Looos, R.G.K., and Musser, D.R., "Generic Programming", LNCS 1766, Springer 2000.

[12] Klein, G., Nipkow, T., "Verified Bytecode Verifiers", *Theorerical Computer Sciences*, 2003.

[13] Krishnamurthi, S., Fisler, H., and Greenberg, M., "Verifying aspect advice modularly ", *Foundations of Software Engineering*, 2004.

[14] Manna, Z., "The correctness of programs", *Journal of Computer and System Sciences*, 3 (2) 1969.

[15] Manna, Z., *Mathematical Theory of Computation*, McGraw-Hill, 1974.

[16] Markoski, B., "Program testing and deductive checking of their validaty", MS Thesis, University of Novi Sad, Faculty of Technical Sciences, Novi Sad, 2000.

[17] Mills, H., and Linger, R., "Cleanroom Software Engineering", *Encyclopedia of Software Engineering, 2nd (J. Marciniak, ed.).* New York: John Wiley & Sons, 2002.

[18] Nipkow, T., Paulson, L.C., Wenzel, M., "A proof assistant for higher-order logic", Springer, *Lecture Notes in Computer Science*, Vol 2283, 2002

[19] Pascual, J.I,. "On the Correctness of the Factoring Transformation", *FLOPS* 2002.

[20] Radulović, B., Hotomski, P., "Projecting Deductive Databases with CWA Management i Baselog system" Novi Sad J. Math Vol.30, No 2, 2000.

[21] Sihman, M., and Katz, S., "Model Cbeching application of aspect and superimposition ", In G.T. Leavents and C. Clicton (eds.), FOAL, 2003.

APPENDIX

Here is presented example which is realised in BASELOG system, without modification of resolution procedure for prooving correctness of program structure based on program's logic rules.

Example 1.

```
begin
max:=x[1];
i:=0;
while i<=n do
begin
        i:=i+1;
        if x[i]>max then max:=x[i];
end;
end.
```

Let predicate i<=n; be marked with constant b, and b1 predicate x[i]>max, by constant t1 term x[1]>, by constant t2 term x[i], , then it is written:

```
        s(s(d(max,t1),d(i,0)),w(b,s(d(i,i+1,if(b1,d(max,t2))))))
1
/IM1(X0,Y0)/IM(X2,Y2)O(X1,V1)~K(X1,s(h,g),Y1)~K(Y1,w(b,if(b1,e)),V1)&
11
~K(Y1,d(max,t1),V1)K(Y1,h,V1)& / reserve for shortening the entry length
~K(Y1,d(i,0),V1)K(Y1,g,V1)& / reserve for shortening the entry length
~K(Y1,d(max,t2),V1)K(Y1,e,V1)& / reserve for shortening the entry length
~K(X1,Y1,U1)~K(U1,Y2,V1)K(X1,s(Y1,Y2),V1)&/sequention rule
~K(k(X1,Y1),Z1,U1)~IM(k(X1,n(Y1)),U1)K(X1,if(Y1,Z1),U1)& /if rule
~K(k(X1,V2),U0,X1)K(X1,w(V2,U0),k(X1,ng(V2)))& / while rule
K(t(X1,Z1,Y1),d(Z1,Y1),X1)& /
~IM(X2,Y1)~K(Y1,U0,V1)K(X2,U0,V1)& / consequence rule
~IM(Y1,V1)~K(X1,U0,Y1)K(X1,U0,V1)& / consequence rule
 ~IM1(X0,Y0)IM(X0,Y0)&
~O(X1,Y1)&
0
0
Baselog  system generates the following negation
number of generated resolvents = 14
maximal reached level = 15
the level on which is generated empty  component = 15
LEVEL = 1; central  component
:/IM1(X0,Y0)/IM(X2,Y2)O(X1,V1)~K(X1,s(h,g),Y1)~K(Y1,w(b,if(b1,e)),V1)&
6.lateral, 2.literals :
   ~K(k(X1,V2),U0,X1)K(X1,w(V2,U0),k(X1,ng(V2)))&
LEVEL= 2; resolvents:
    /IM1(X0,Y0)/IM(X2,Y2)O(X1,k(X3,ng(b)))~K(X1,s(h,g),X3)/~K(X3,w(b,if(b1,e)),
    k(X3,ng(b)))~K(k(X3,b),if(b1,e),X3)&
```

```
5. lateral, 3.literals:
~K(k(X1,Y1),Z1,U1)~IM(k(X1,n(Y1)),U1)K(X1,if(Y1,Z1),U1)&
LEVEL= 3; resolvents:
/IM1(X0,Y0)/IM(X2,Y2)O(X1,k(U1,ng(b)))~K(X1,s(h,g),U1)/~K(U1,w(b,if(b1,e)),k(U1
,ng(b)))/~K(k(U1,b),if(b1,e),U1)~K(k(k(U1,b),b1),e,U1)~IM(k(k(U1,b),n(b1)),U1)&
LEVEL= 4; resolvents predecessor;
/IM1(X0,Y0)/IM(k(k(U1,b),n(b1)),U1)O(X1,k(U1,ng(b)))~K(X1,s(h,g),U1)/~K(U1,w(b,
if(b1,e)),k(U1,ng(b)))/~K(k(U1,b),if(b1,e),U1)~K(k(k(U1,b),b1),e,U1)&
3. lateral, 2.literals :
~K(Y1,d(max,t2),V1)K(Y1,e,V1)&
LEVEL= 5; resolvents:
/IM1(X0,Y0)/IM(k(k(V1,b),n(b1)),V1)O(X1,k(V1,ng(b)))~K(X1,s(h,g),V1)/~K(V1,w(b,if
(b1,e)),k(V1,ng(b)))/~K(k(V1,b),if(b1,e),V1)/~K(k(k(V1,b),b1),e,V1)~K(k(k(V1,b),b1),
d(max,t2),V1)&
8. lateral, 3.literals :
 ~IM(X2,Y1)~K(Y1,U0,V1)K(X2,U0,V1)&
LEVEL= 6; resolvents:
/IM1(X0,Y0)/IM(k(k(V0,b),n(b1)),V0)O(X1,k(V0,ng(b)))~K(X1,s(h,g),V0)/~K(V0,w(b,
if(b1,e)),k(V0,ng(b)))/~K(k(V0,b),if(b1,e),V0)/~K(k(k(V0,b),b1),e,V0)/~K(k(k(V0,b),b1
),d(max,t2),V0)~IM(k(k(V0,b),b1),Y1)~K(Y1,d(max,t2),V0)&
7. lateral, 1.literals :
    K(t(X1,Z1,Y1),d(Z1,Y1),X1)&
LEVEL= 7; resolvents:
/IM1(X0,Y0)/IM(k(k(X2,b),n(b1)),X2)O(X1,k(X2,ng(b)))~K(X1,s(h,g),X2)/~K(X2,w(b,
if(b1,e)),k(X2,ng(b)))/~K(k(X2,b),if(b1,e),X2)/~K(k(k(X2,b),b1),e,X2)/~K(k(k(X2,b),b1
),d(max,t2),X2)~IM(k(k(X2,b),b1),t(X2,max,t2))&
10. lateral, 2.literals :
 ~IM1(X0,Y0)IM(X0,Y0)&
LEVEL= 8; resolvents:

/IM1(X0,Y0)/IM(k(k(X2,b),n(b1)),X2)O(X1,k(X2,ng(b)))~K(X1,s(h,g),X2)/~K(X2,w(b,
if(b1,e)),k(X2,ng(b)))/~K(k(X2,b),if(b1,e),X2)/~K(k(k(X2,b),b1),e,X2)/~K(k(k(X2,b),b1
),d(max,t2),X2)/~IM(k(k(X2,b),b1),t(X2,max,t2))~IM1(k(k(X2,b),b1),t(X2,max,t2))&
LEVEL =9; resolvents predecessor;
/IM1(k(k(X2,b),b1),t(X2,max,t2))/IM(k(k(X2,b),n(b1)),X2)O(X1,k(X2,ng(b)))~K(X1,s(
h,g),X2)&
4. lateral, 3.literals :
 ~K(X1,Y1,U1)~K(U1,Y2,V1)K(X1,s(Y1,Y2),V1)&
LEVEL = 10; resolvents :
/IM1(k(k(V1,b),b1),t(V1,max,t2))/IM(k(k(V1,b),n(b1)),V1)O(X0,k(V1,ng(b)))/~K(X0,s(
h,g),V1)~K(X0,h,U1)~K(U1,g,V1)&
2. lateral, 2.literals :
    ~K(Y1,d(i,1),V1)K(Y1,g,V1)&
LEVEL = 11; resolvents:
/IM1(k(k(V0,b),b1),t(V0,max,t2))/IM(k(k(V0,b),n(b1)),V0)O(X0,k(V0,ng(b)))/~K(X0,s(
h,g),V0)~K(X0,h,Y1)/~K(Y1,g,V0)~K(Y1,d(i,1),V0)&
```

7. lateral, 1.literals :
K(t(X1,Z1,Y1),d(Z1,Y1),X1)&
LEVEL = 12; resolvents:
/IM1(k(k(X1,b),b1),t(X1,max,t2))/IM(k(k(X1,b),n(b1)),X1)O(X0,k(X1,ng(b)))/~K(X0,s(h,g),X1)~K(X0,h,t(X1,i,1))&

1. lateral i, 2.literals :
~K(Y1,d(max,t1),V1)K(Y1,h,V1)&
LEVEL = 13; resolvents:
/IM1(k(k(X1,b),b1),t(X1,max,t2))/IM(k(k(X1,b),n(b1)),X1)O(Y1,k(X1,ng(b)))/~K(Y1,s(h,g),X1)/~K(Y1,h,t(X1,i,1))~K(Y1,d(max,t1),t(X1,i,1))&
7. lateral, 1.literals :
K(t(X1,Z1,Y1),d(Z1,Y1),X1)&
LEVEL= 14; resolvents:

/IM1(k(k(X1,b),b1),t(X1,max,t2))/IM(k(k(X1,b),n(b1)),X1)O(t(t(X1,i,1),max,t1),k(X1,ng(b)))&
11. lateral, 1.literals :
~O(X1,Y1)&
LEVEL= 15; resolvents:
&
The proof is printed

Now we have to prove accordance, that is to confirm that
$(X_\theta \Rightarrow Y_\theta{}^{Z\theta}{}_{T\theta}) \wedge (U \Rightarrow P_\theta, \ \wedge \ Q_\theta \Rightarrow V)$ which is located on LEVEL= 14; resolvents:
/IM1(k(k(X1,b),b1),t(X1,max,t2))/IM(k(k(X1,b),n(b1)),X1)O(t(t(X1,i,1),max,t1),k(X1,ng(b)))&

Returning notation to domain level we have:
$(X1 \wedge \ (i<=n) \Rightarrow X1^{i}{}_{i+1} \ {}^{p}{}_{p/i})i \ (U \Rightarrow X1^{i}{}_{0} \, {}^{p}{}_{x}) \wedge (X1 \wedge \neg (i<=n) \Rightarrow V)$

Setting for X1: max=x(i) $\overset{i}{\underset{j=1}{\wedge}}(x(j) \geq x(i)), i \in [1..n]$ we obtain the following correct imlications:

$$\max = x(i+1) \overset{i+1}{\underset{j=1}{\wedge}}(x(j) \geq x(i))$$

$$=> \max = x(i+1) \overset{i}{\underset{j=1}{\wedge}}(x(j) \geq x(i)) \wedge (x(i+1) \geq x(i)$$

For i=1putting :

$$\max = x(1) \overset{1}{\underset{j=1}{\wedge}}(x(j) \geq x(i)) => \max = x(1)$$

Thus, the accordance is proved, which is sufficient for conclusion that the given program is partially correct.