A Theorem Proving System for Logic Design Verification

NAOYUKI YAMADA,* YASUHIRO KOBAYASHI* and TAKASHI KIGUCHI*

This paper describes a logic design verification system based on a heuristically guided theorem proving method. The system called HTPS consists of three main programs; meta-level program, base-level program and control program. In the meta-level program, domain specific knowledge expressed in the form of a rule is used to generate guidance for the proof procedure. According to this guidance, proof is performed in the base-level program. The control program regulates these procedures. In order to facilitate this hierarchical inference mechanism, a connection graph method is adopted as the proof strategy in the base-level program.

HTPS was successfully applied to verification of logic circuits and its effectiveness and usefulness were clearly demonstrated.

1. Introduction

Hardware verification that makes use of the theorem proving techniques has been receiving much attention. It is considered to be one alternative to numeric simulations for verification of the correctness of logic design [1]. Verification by theorem proving techniques, sometimes referred to as formal verification, is based on the following idea; if the proposed design is correct and meets its design specifications, the specifications can be logically proved as a theorem under the proposed design. Under a hierarchical design methodology of digital systems, there are several steps that can be viewed as levels in the design process. Usually, verification utilizes this hierarchical nature; i.e. verification is done by proving the desired behavior specified at one level of the design hierarchy under the proposed implementation at a level one lower.

One notable study with this verification method was performed by Wagner [2], who developed a non-procedural hardware description language RTS (Resister Transfer Statement) and used a theorem prover called FOL (First Order Logic) [3]. Although the proof of an 8 bit multiplier with 260 steps was excellent, it was entirely guided manually. Wojcik and his coworkers tried to automate a proof and used AURA (Argonne National Laboratory Automated Reasoning Assistant) as a theorem prover in their design verification system [4]. Recently, AURA has been replaced by a more portable system LMA (Logic Machine Architecture) [5]. They showed the feasibility of generating a proof automatically by verifying simple logic circuits [6].

In these works, existing general purpose theorem provers were employed to generate a proof, and in order for efficient verification, domain specific knowledge was utilized implicitly only in the selection of proof strategies they provide and/or in the way of grouping axioms which were supplied to them. Therefore, complexity of the proof process was still exceedingly high, even in the case of verification of simple logic circuits. Certainly, the efficiency of verification depends heavily upon the symbolic manipulation abilities that the theorem prover has. However, there still remains a possibility to improve the efficiency drastically by introducing domain specific knowledge explicitly into the control of a proof procedure. From the viewpoint of artificial intelligence, control of the proof procedure can be categorized as an inference mechanism known as meta-level reasoning.

This paper describes an efficient theorem proving system called HTPS (Heuristically guided Theorem Proving System), which is suitable for logic design verification. HTPS is a resolution-based theorem prover which embodies an introduction of domain specific knowledge explicitly into the control of the inference procedure. It adopts a connection graph method as an appropriate proof strategy in order to make the hierarchical inference easy to perform.

2. Hardware Verification by Theorem Proving Method

2.1 General Description

The logic design verification method by theorem proving techniques involves verifying a proposed design by proving the design specifications under the proposed design. As mentioned before, this technique is used under a hierarchical design method [1] and so the proposed design and its specifications are sometimes refer-

Energy Research Laboratory, Hitachi Ltd. 1168 Moriyamacho, Hitachi, Ibaraki 316, Japan.

A paper submitted to Journal of Information Processing for review by Information Processing Society of Japan.

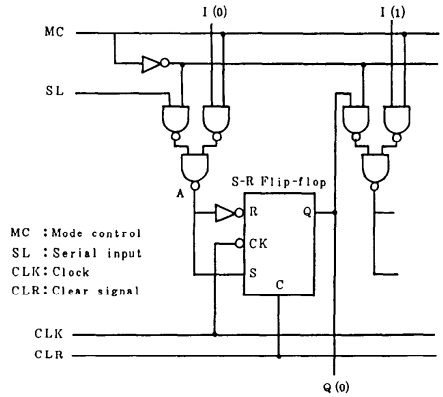


Fig. 1 Partial logic circuit diagram of the 4 bit shift register.

```
(Rt CLR (Rf Q 0 0) c0)                                          (2-1)
(Rt (And (Not CLR)(Td CLK)) (Rf Q 0 0) A)                       (2-2)
(Rt (And (Not CLR)(Not (Td CLK))) (Rf Q 0 0) (Rf Q 0 0)         (2-3)
(Rt cl (Or (And (Not MC) SL)(And MC (Rf I 0 0))))               (2-4)
```

Fig. 2 Part of the RTS expression of the 4 bit shift register.

```
(Eq (Conc (Rf $x 0 0)(Rf $x 1 $m)) (Rf $x 0 $m))        (3-1)
(Eq (Conc (Rf $x 1 1)(Rf $x 2 $m)) (Rf $x 1 $m))        (3-2)
(Eq (Conc (Rf $x 2 2)(Rf $x 3 $m)) (Rf $x 2 $m))        (3-3)
(if (and (Rt $x $a $c)(Rt $x $b $c))
    (Rt $x (Conc $a $b) $c))                            (3-4)
```

Fig. 3 Examples of axioms.

```
(not (Rt CLR (Rf Q 0 3) c0)                                          (4-1)
        ⇩ Paramodulation on (3-1)
(not (Rt CLR (Conc (Rf Q 0 0) (Rf Q 1 3)) c0))                       (4-2)
        ⇩ Resolution on (3-4)
(not (Rt CLR (Rf Q 0 0) c0)) V (not (Rt CLR (Rf Q 1 3) c0))          (4-3)
        ⇩ Resolution on (2-1)
(not (Rt CLR (Rf Q 1 3) c0))                                         (4-4)
```

Fig. 4 An example of proof process.

red to as the proposed implementation and the desired behavior, respectively.

In order to realize this verification, the following three key techniques are required; (a) the development of a flexible hardware description language which covers broader levels of design hierarchy, (b) the establishment of enough axioms which operate upon the theorem to be proved and form a proof process and (c) the development of an efficient theorem proving system with powerful symbolic manipulation ability to generate a proof. Usually, a non-procedural language is used as its description language. As for the theorem prover, a resolution-based one is adopted since it is suitable for mechanical theorem proving.

Table 1 RTS expression.

(Rt W A B)	: Register Transfer Statement
	W : Boolean condition
	A : Destination
	B : Source
	meaning : if W is true then B is transferred to A
(Rf A n m)	: Reference to register A, bit n through m
(Td A)	: Down transition for signal A
(Conc a b)	: Bit concatenation a and b
(And a b)	: Boolean operator
(Or a b)	: Boolean operator
(Not a)	: Boolean operator

In the following, the verification technique is outlined through a verification using a 4 bit shift register [4]. Figure 1 shows the partial logic circuit of the register and its description written in RTS (Register Transfer Statement) is given in Fig. 2. Figure 3 shows some axioms needed for this verification. Here, RTS is a non-procedural register transfer language developed by Wagner [2]. The present HTPS uses RTS as its description language. The RTS is summarized in Table 1 in a form convenient for HTPS.

Consider the verification of a design specification for which each bit of the output register is set to "0" by a clear signal. This specification is expressed in RTS as follows,

(Rt CLR (Rf Q 0 3) c0)

where, CLR and c0 denote the clear signal and signal value "0", respectively. A resolution-based theorem prover proves a theorem by extracting a contradiction from the clause set which includes the negation of the theorem to be proved. This resulting process is called a refutation process. In this example, the clause

(not (Rt CLR (Rf Q 0 3) c0))

is added to the clause set given in Figs. 2 and 3. In generating a proof, an inference rule that handles axioms expressed in a equality relation is indispensable since such axioms as Boolean algebra play an important role in the logic design verification. In this example, resolution and paramodulation are used and its proof process is shown in Fig. 4. Although the proof is given for only the first bit of the register, the proof for other bits can be obtained in the same manner.

The procedure outlined above was employed in previous works [4], [6] for the logic design verification method based on theorem proving technique, though demodulation was used for equality handling instead of paramodulation. However, as can be easily predicted from the above description, it is rather difficult to find a refutation process when this technique is applied to the verification of larger circuits. Therefore, development of an efficient control mechanism in theorem proving is required in order to make this verification technique practical.

2.2 The Principle of HTPS

The principle underlying HTPS is that in proving the desired behavior under the proposed implementation, the manipulation sequence of the desired behavior can be determined by the characteristics of the desired behavior, information about the proposed implementation, and the meaning of each axioms that can be applied to the desired behavior. From the viewpoint of a verification method by theorem proving, the proof procedure is considered to be composed of the following steps.

(1) The desired behavior is expanded into a description at a lower level by using axioms for expansion.

(2) The expanded behavior is modified by axioms for transformation.

(3) Then each modified behavior is refuted with the proposed implementation.

The characteristics of the desired behavior and the proposed implementation can be utilized to determine the proof strategy through steps (1) to (3). On the other hand, the meaning of each axiom can be used to generate detailed proof steps in step (2) under the determined strategy.

The use of this information (domain specific knowledge) explicitly in the control of the inference brings about big advantages in the verification efficiency. Firstly, it enables the system to execute a bottom-up inference as well as the usual top-down inference. By using the characteristics of a theorem to be proved and the proposed implementation, it is possible to set a potentially complimentary unifiable literal and modify it into a convenient expression. Secondly, it allows implementation of the "modification if-needed" idea. Namely, a modification is executed only to reduce discrepancies between the theorem to be proved and the nominated potentially complimentary unifiable literal. In logic design verification, it is necessary to handle many equalities and that produces an astronomically large search space. Therefore, the "modification if-needed" process cuts down the search space considerably.

3. Theorem Proving System HTPS

3.1 System Overview

3.1.1 Basic Approach

According to the principle mentioned above, HTPS is organized in the hierarchical structure; meta-level and base-level. The meta-level program controls the proof procedure by using the domain specific knowledge. Under this control, proof is executed by the base-level program. The domain specific knowledge includes characteristics of the desired behavior, information about the proposed implementation and knowledge about axioms usage.

In order to implement the meta-level program, the following requisites should be taken into consideration.

(1) The program should be flexible and modular so that its capability to determine the proof procedure can be easily enhanced.

(2) The overhead time caused by introduction of this level should be as small as possible.

Considering these items, the meta-level program is realized as a kind of production system. In this framework, the domain specific knowledge is organized hierarchically and represented in the form of if-then rules. In addition, each rule is expressed by using predefined functions in order to be easily converted to an executable program.

On the other hand, the following items should be taken into consideration in developing the base-level program.

(3) As an inference rule, a method to treat equalities, besides resolution, is needed.

(4) The proof strategy employed in resolution should be effective and suitable for clear separation of the meta-level and base-level.

Item (3) reflects the fact that theorem proving for logic design verification needs groups of axioms, for example, rules of Boolean algebra and rules for handling the concatenation of the bit level structure, and that most of them are represented in the form of equality literals. As for item (4), a proof strategy that it is itself effective and has specifications of inference under it which are simple enough, to ensure the total inference is effective. In order to meet these requisites, HTPS adopts the connection graph method for resolution and paramodulation [7], [8].

3.1.2 System Configuration

The system configuration of HTPS is shown in Fig. 5. The meta-level program and base-level program are combined through the control program. The meta-level program consists of two modules. The miscellaneous function module gives function definitions which are utilized in describing the knowledge in the knowledge base. The proof guidance generation module is an interpreter that operates upon the knowledge base to yield proof guidance.

In the base-level program, the resolution and paramodulation modules execute inference using the connection graph method. In this execution, the unification module is accessed frequently as a basic operation. In order to enhance the symbolic manipulation ability, a lexical ordering module is added which performs lexical ordering under the connection graph method. All interactions of these modules with the connection graph are performed through the connection graph management module.

As shown in Fig. 5, HTPS also has a preprocessing module. With this module the proposed design expressed in RTS and axioms are simplified, lexically ordered and converted into expression in the Conjunctive Normal Form (CNF). In the description of the propos-

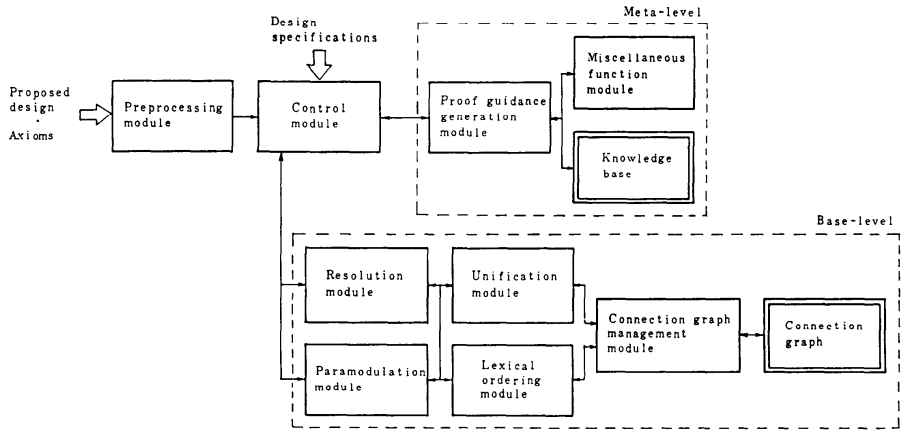


Fig. 5 The system configuration of HTPS.

ed design, there are many redundant and complex expressions including the signal value "0" and "1". In order to reduce the burden of theorem proving system, these expressions are simplified beforehand by applying simplification rules. Most of these simplification rules are from Boolean algebra. Canonicalization aids in the identification of equivalent expressions. In particular, in the case of unification of two expressions that contain functions that follow commutativity and associativity, canonicalization is needed. As an example of this module, the following expression,

(And a (And (And $x (Rf Q 2 2)) (Rf Q 1 1)))

is transformed to

(And $x (And a (And (Rf Q 1 1) (Rf Q 2 2)))),

where the symbol which begins with "$" denotes a variable. Note that the lexcial ordering in the theorem proving process plays an important role.

The major characteristics of HTPS as compared to the theorem proving techniques developed so far can be summarized as follows.

(1) A way to introduce domain specific knowledge to inference control is developed.

(2) The inheritance mechanism of paramodulation under the connection graph method is improved to complete the inheritance.

(3) A lexical ordering algorithm under the connection graph method is developed.

3.2 Metal-level Program

The meta-level program is constructed as a kind of production system. The knowledge base contains if-then rules that relate the characteristics of the theorems to be proved, information about other theorems and axioms which can be used in the proof procedure with proof guidance. These rules are obtained through investigation of the structure and the meaning of the circuit representation and the usage of axioms.

These rules are grouped into two categories, one is that used for generating global strategies (hereafter the term strategy is used for simplicity), and the other is used for finding concrete inference steps (hereafter the term tactics is used). This categorization is useful to keep the size of the search space for applicable knowledge manageable. On the other hand, in order to have fast processing of knowledge itself, the knowledge in each category is expressed in if-then rules and both if-part and then-part of the rules are written by using predefined functions. This representation method enables conversion of each rule into an executable function.

(a) Functions used for describing the if-part of strategy determination rules

```
*Target_literal (p)      : get the literal index of the current theorem
*Cond_part_of (p q)      : get the Boolean condition part of literal p, and set it to q
*Dest_part_of (p q)      : get the destination part of literal  p, and set it to q
*Sour_part_of (p q)      : get the source part of literal p, and set it to q
*Has_R_link (p q r)      : get the R-link index connecting literal p and the
                           literal which has the name q, and set it to r
*Has_P_link (p q r)      : get the P-link index connecting term p and the term
                           which has the name q, and set it to r
*Exist_apliteral (p q r s) : get the literal index which has the same q part
                             with p and different r part with p, and set it to s
Differ (p q r)           : check whether literals p and q have different r parts
```

(b) Functions used for describing the then-part od strategy determination rules

```
Set_tl (p)           : set the literal index p to TL
Set_tt (p)           : set the term index p to TT
Set_cl (p)           : set the literal index p to CL
Set_ct (p)           : set the term index p to CT
Set_strategy (p)     : set the strategy name p to STRATEGY
Set_link (p)         : set the link index p to LINK

Note : Functions for which the name begin with "*" expect return values
       and others do not.
```

Fig. 6 The functions used in describing strategy determination rules.

It is important to be able to add, modify, and delete knowledge easily in the meta-level program. This feature not only ensures the meta-level program flexibility, but also makes incremental development of the system possible.

3.2.1 Rules for Strategy Determination

Rules for strategy determination are evaluated to find the global proof strategy. Rules are therefore, represented by using the relations between characteristics of theorems to be proved and their environment, where environment means the status of the connection graph, or more precisely, the existence of

```
        Rule    form                                    Meanings
(Srule-1
   (If (and (*Target_literal $x)                        As against the theorem to be proved
            (*Exist_spliteral $x '(BC DT) '(ST) $y)     if there exists the literal which has
            (*Sour_part_of $x $w)                       the same BC, DT and different ST, then
            (*Sour_part_of $y $z))                      set the literal and its ST to CL and CT
       then (Set_cl $y)                                 and set the ST of the theorem to TT
            (Set_ct $z)                                 and set "Unite-ST" to STRATEGY.
            (Set_tt $w)
            (Set_strategy 'Unite-ST))
   6)

(Srule-2
   (If (and (*Target_literal $x)                        As againt the theorem to be proved
            (*Dest_part_of $x $y)                       if there exists the literal DT of which
            (*Has_P_linkz $y 'Expand $z)                has an P-link to the equality literal
       then (Set_link $z)                               "Expand", then set the link to LINK
            (Set_strategy 'Expand-DT))                  and set "Expand-DT" to STRATEGY.
   2)

        BC  : Boolean condition term,     R-link : Resolution link
        DT  : Destination term,           P-link : Paramodulation link
        ST  : Source term
```

Fig. 7 Examples of strategy determination rules.

Table 2 Global constants and their usage.

STRATEGY	Store the strategy name
LINK	Store the link index on which inference is performed
TL	Store the literal index which is to be proven
TT	Store the term index which is to be modified in TL
CL	Store the literal index which is nominated as a potentially complimentary unifiable literal of TL
CT	Store the term index which is to be modified in CL

literals that can be complementarily unifiable with the theorem. If it is difficult to find the appropriate literals, it is necessary to find an appropriate axiom to convert the theorem into a more desirable form. As a result, rules for strategy determination can be grouped into one of two functions; one is to set a target hypothesis (literal) and strategy name which is used to trigger the tactics determination rules and the other is to set a link directly.

Examples of standard functions to be used in these rules are shown in Fig. 6. Each rule is expressed in the if-then rule by using these functions. Typical examples of the strategy determination rules are shown in Fig. 7. As shown there, each rule consists of the rule index, rule body and its priority, where each priority is given to specify the order of evaluation.

In the HTPS, some global constants are set up to smooth the interaction between strategy determination and tactics determination, and also between the meta-level and base-level. These constants and their usage are summarized in Table 2.

3.2.2 Rules for Tactics Determination

Tactics determination rules are used to find detailed proof steps in the form of link indecies. These rules are basically expressed as relations between characteristics of both the theorems to be proved and the nominated

```
(a) Functions used for describing the if-part of tactics determination rules

*Has_P_linkx (p q r)   : get the P-link index connecting p and the term which
                        has the group name q, and set it to r
*Has_P_linky (p q r)   : get the P-link index connecting p and the term which
                        has the name q and the triggering direction of which
                        is normal, and set it to r
*Has_P_linkz (p q r)   : get the P-link index connection p and the term which
                        has the group name q and the triggering direction of
                        which is normal, and set it to r
*Get_elem (p q r)      : get the q-th element of p, and set it to r
Subsume (p q)          : check whether the elements of p subsume the elements
                        of q
*Subsume (p q r)       : if the elements of p subsume the elements of q, set
                        the extra elements of p to r

(b) Functions used for describing the then-part of tactics determination rules

Resolve (p q)          : resolve upon the link index q of literal p
Paramodulate (p q)     : paramodulate upon the link index q of literal q
Unite_terms (p)        : make the two terms given in the pair form p equal
                        by applying one or more paramodulations
Lex_part (p q r)       : place the elements q at the head of the term p and
                        lexically order the rest of the elements of p by
                        applying paramodulation
Lex_all (p)            : apply the lexical ordering to the elements of p

    Note : Functions for which the name begin with "*" expect return values
           and others do not.
```

Fig. 8 The functions used in describing tactics determination rules.

```
        Rule    form                                    Meanings
(Trule-1
   (If (Strategy Unite-ST)                              If STRATEGY is "Unite-ST" and TT has
       then (If (*Has_P_link TT 'Simplify $x)           P-link to the equality literal "Simplify"
                then                                    then paramodulate on the link and set
                     (Paramodulate TT $x)               the ST of the resulting literal to TT.
                     (*Sour_part_of TL $y)
                     (Set_tt $y)))
3)

(Trule-2
   (If (Strategy Unite-BC)                              If STRATEGY is "Unite-BC" and the function
       then (If (and (*Func_symbol TT $x)               of TT is "And" and the second term of TT
                     (Eq_symbol $x 'And)                subsumes CT, then find the R-link of TL
                     (*Get_elem TT 2 $y)                which connects to the literal "and_red2"
                     (Subsume $y CT))                   and resolve on the link and set the BC of
                then                                    the resulting literal to TT.
                     (*Find_R_link TL 'and_red2 $z)
                     (Resolve TL $z)
                     (*Cond_part_of TL $w)
                     (Set_tt $w)))
7)

        BC  : Boolean condition term,     R-link : Resolution link
        DT  : Destination term,           P-link : paramodulation link
        ST  : Source term,
```

Fig. 9 Examples of tactics determination rules.

potentially complimentary unifiable literal, and concrete proof guidance.

Simplification and canonicalization are necessary operations in mechanical theorem proving procedures. In the HTPS, these operations are also accessed by evaluating these rules, by which the important idea "if-needed paramodulation" is realized. In addition to these expression manipulation functions, the HTPS has a function like ETG (Equality Tree Generator) [7] to find a link sequence that makes two expressions equal by triggering plural equality literals.

Examples of standard functions which the HTPS has to represent these rules are shown in Fig. 8, and typical rules are shown in Fig. 9. Tactics determination rules have nested premises in their bodies. The first premise denotes the strategy name under which the rule is searched, and the second one is used as the premise in ordinary if-then type rules. Each priority included in the rules indicates the order for applying them under the specified strategy. It is important to note here that HTPS is free from the loop problem which may happen in applying such an equality axiom as Boolean distributivity [4], [6]. In each tactic, triggering directions of equality axioms can be specified if necessary. For example, the function *Has_P_linky in Fig. 8 is

used for this purpose, and some equality axioms are associated with normal triggering direction as shown in Fig. 16 later. Each rule of this category is also converted into an executable function.

3.3 Base-level Program

According to the proof guidance generated in the meta-level program, proof is performed in the base-level program. Therefore, the main part of the base-level program is mainly composed of the inference module and the unification module which is a fundamental procedure for inference. As mentioned earlier, the proof procedure in the HTPS base-level adopts the connection graph method as its strategy. As for the unification module, the HTPS uses Martelli's and Montanari's algorithm [9].

3.3.1 Connection Graph Method

The connection graph method was first developed by Kowalski [7], and extended by Siekmann and Wrightson [8] to handle equality. The proof procedure of the base-level program is based on the paramodulated connection graph method. Its proof procedure is made more compact than original connection graph method by introducing some refinements.

In order to gain notion of the connection graph method, the example proof process should be explained first. The example gives proof of the specification of a 4 bit shift register, corresponding to Fig. 4. In Fig. 10, the connection graph is represented by drawing R-links and P-links between literals and terms, the roots of which are located at the position of their predicates and function symbols. The R-link is a link which connects complementary literals, on the other hand, the P-link is one which connects paramodulable terms.

The upper most graph is the initial connection graph constructed by both the theorem to be proved (b), and other clauses shown in Figs. 2 and 3. In this graph, only the minimum set of clauses is displayed and the R-inks which connect two literals in the same clause (auto-links) in (d) are omitted for simplicity. The paramodulation on link ① yields the paramodulant (e) in the second graph. R-link ⑤ is obtained by the inheritance mechanism (mentioned later). Then resolution of link ⑤ yields the resolvent (f) in the third graph. Here, link ⑥ is also obtained by the inheritance mechanism. The final clause (g) which corresponds to the clause (4-4) in Fig. 4 is obtained by resolving link ⑥. Although the links attached to the clause (g) are not shown in Fig. 10, there are P-links to the omitted clause and it is possible to continue the proof.

As can be seen from the above example, the connection graph method is suitable for a theorem proving strategy that requires clear separation from its control.

A detailed proof procedure of the connection graph method is described in Fig. 11. In the procedure there, the inheritance check of R-links and P-links for resolvent and paramodulant is one of the distinct

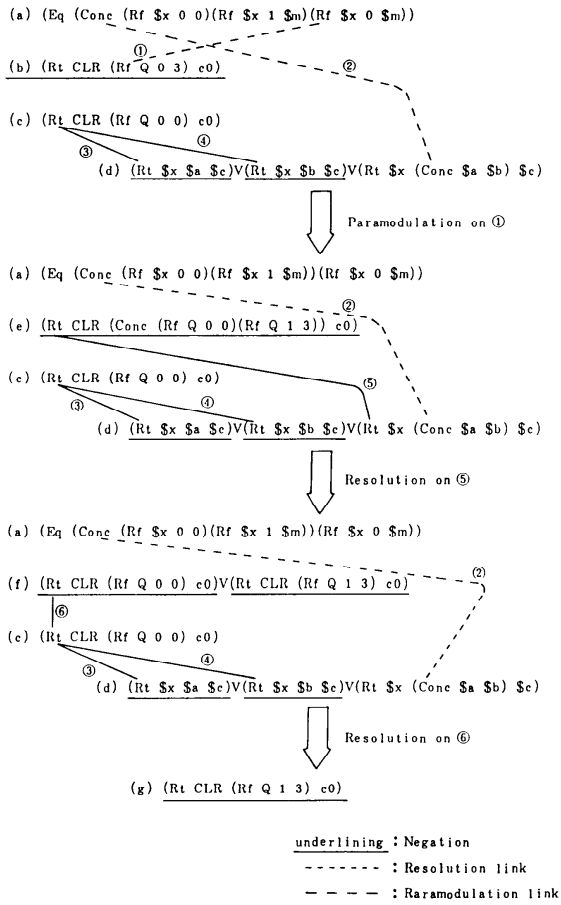


Fig. 10 An example proof process of connection graph method.

characteristics of the connection graph method. That is, in the proof process, the applicability of resolution and paramodulation of the resolvent and paramodulant is inherited from their parent clauses and no further searching for them is needed. The procedure for checking the inheritance of R-links is as follows.

For each literal L in R (resolvent),
  if L descends from L' in C or D (parent clause)
  and if L' is connected by an R-link to some
    literal K,
  and if L and K are unifiable,
  connect K and L by an R-link.

As for the inheritance of P-links, the mechanism is improved to complete the applicability of paramodulation. A newly introduced procedure checks for unifiability of the terms replaced in the paramodulant by its parent term in the equality literal. This procedure is indispensable for paramodulation in which equality literals with a self unifiable term are used, where a self unifiable term is one which contains a subterm unifiable with itself. The improved procedure for checking the inheritance of P-links consists of two steps as follows.

```
(1) Initially set up the connection graph <S>.
 (a) Generate and include the graph all factors of clauses in <S>.
 (b) For every pair of unifiable complementary literals in clauses
     in S, insert an R-link connecting them.
 (c) For every pair of literals, one is either equality or non-equality
     literal and the other is an equality literal in the clauses in S,
     if the term of either side of the latter literal is unifiable with
     some term in the former literal, insert a P-link connecting them.
 (d) For every equality literal, if any term on one side is unifiable
     with the term of the other side, insert a P-link connecting them.
(2) In order to process a connection graph,
    if <S> contains the empty clause, terminate successfully,
    else if <S> is itself empty, terminate with failure,
    else
 (a) Select a link l
 (b) If l is a R-link,
  (b-1) Add the resolvent R and its factor F to <S>,
  (b-2) For each literal in R and F, check the inheritance of R-links,
  (b-3) For each term in R and F, check the inheritance of P-links.
 (c) If l is a P-link,
  (c-1) Add the paramodulant P to <S>,
  (c-2) For each literal in P, check the inheritance of R-links,
  (c-3) For each term in P, check the inheritance of P-links,
  (c-4) For each new P-link connecting a term in a literal L in P to
        some other term in a literal K, if L and K are unifiable,
        erase P-link and insert R-link connecting L and K,
        else erase the P-link,
  (c-5) Add each factor F of P to <S>,
  (c-6) For each literal in F, check the inheritance of R-links,
  (c-7) For each term in F, check the inheritance of P-links.
 (d) Delete l from <S> and remove the tautology and pure clauses and all
     the links connecting them from <S> until <S> contains no tautology
     and pure clauses.
 (e) Return to (2).

        S    : Set of clauses
       <S>   : Connection graph constructed from S
      P,R,F  : Clause
      L,K    : Literal
```

Fig. 11 Proof procedure of connection graph method.

(1) For each literal L in paramodulant P,
For each term tL in L,
if L descends from L' in parent clauses C or D,
and if tL descends from tL' in L',
and if tL' is connected by a P-link to some term tK
in some literal K,
and if tL and tK are unifiable,
connect tL and tK by a P-link.

(2) For each subterm tL in tT in paramodulant P,
if tT descends from tT' in equality literal C,
and if tL and tT' is unifiable,
connect tL and tT' by a P-link.

In order to clarify the improved procedure, an example inference process is given in Fig. 12, along with a connection graph. This example shows the inference process by which the second term of (a) is lexically ordered using the commutative law (b) and associative law (c). The equality literal (b) permits a "one-sided" paramodulation, i.e. the right hand side of (b) is not connected and only the P-links in the left hand side are used for paramodulation. Paramodulation on ④ in the upper graph of Fig. 12 generates the new literal (d) and the connection graph is changed to the middle graph shown. In that graph, the newly inserted P-links ⑬, ⑭, ⑮, ⑯ and ⑰ are inherited from ⑥, ⑦, ②, ⑪ and ⑩ respectively. At this point the usual inheritance mechanism has no way to connect the left hand side of (b) and the term (And c (And d b)) in (d), which must be

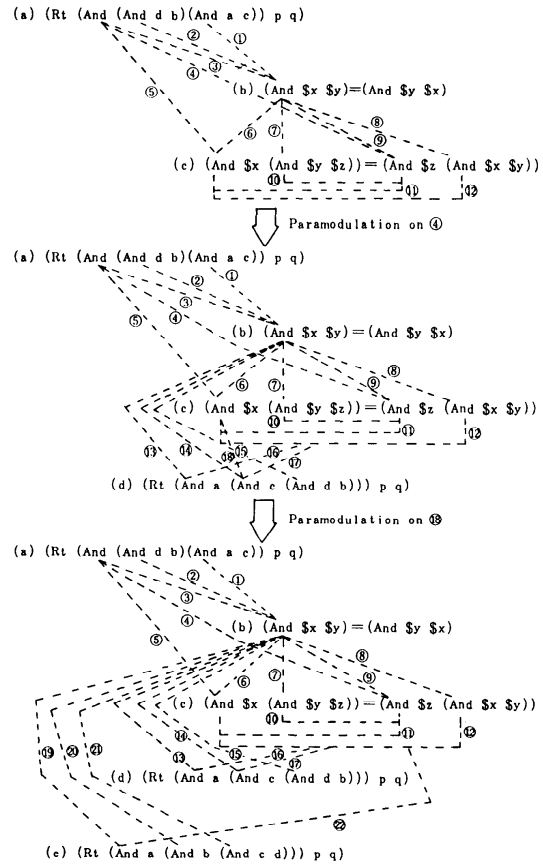


Fig. 12 An example proof process for connection graph method.

linked by a P-link. By introducing the new mechanism mentioned above, they can be connected as P-link ⑱. As a result of this method, the lexically ordered literal (e) is obtained by paramodulating on ⑱ (in the middle graph) as shown in the lower graph of Fig. 12.

Note that this inheritance mechanism of P-links does not destroy the main principle of the connection graph method, which is to eliminate unsuccessful searchs for unifiability.

3.3.2 Unification Method

Unification is a key operation of theorem proving and considered to be time consuming. Therefore, the unification efficiency dominates the global efficiency of the inference process. In the connection graph method, it is necessary to check the inheritance of links which the parent clauses have after each resolution and paramodulation. This means that the unification method should be suitable for retrieving link information from the resulting substitution, as well as being efficient. Taking accounts of these requisites, the HTPS implements Martelli's and Montanari's algorithm [9] by modifying its substitution generating mechanism.

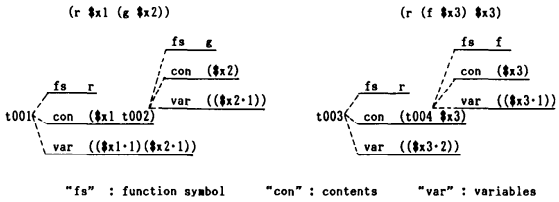


Fig. 13 Internal expression of HTPS.

```
U : { [0]($x)  =(t001 t003)
      [1]($x1)=φ                          Common part : (r $x1 $x3)
      [1]($x2)=φ              --->        Frontier    : (($x1)=(t004),
      [2]($x3)=φ                                        ($x3)=(t002))
      [2]($x3)=φ  }
T : ( )
                              (a)
U : { [0]($x1)=(t004)
      [1]($x2)=φ              --->        Common part :  t004
      [1]($x3)=(t002) }                   Frontier    : (($x3,$001)=φ)
T : (($x)=(r $x1 $x3))
                              (b)
U : { [1]($x2)=φ                          Common part :  t002
      [0]($x3,$001)=(t002) }  --->        Frontier    : (($x2,$002)=φ)
T : (($x)=(r $x1 $x3),($x1)=(t004))
                              (c)
U : { [0]($x2,$002)=φ }
T : (($x)=(r $x1 $x3),($x1)=(t004),($x3,$001)=(t002))
                              (d)
U : ( )
T : (($x)=(r $x1 $x3),($x1)=(t004),($x3,$001)=(t002),($x2,$002)=φ)
                              (e)
```

Fig. 14 Unification process.

Martelli's and Montanari's algorithm formulates the unification problem as the solution of equations and it involves no substitution procedure. Here, in order to show its appropriateness to be used in the connection graph method and also its modification, the unification procedure implemented in HTPS is shown with a simple example. Consider the unification of (*r* $x1 (*g* $x2)) and (*r* (*f* $x3) $x3). In HTPS, these terms are represented in the frame structure using term indexing t001 and t003 respectively as shown in Fig. 13. Each term has a pair designation composed of its variable name and its number for convenience in unification.

The unification process is shown stepwise in Fig. 14, where U and *T* denote a set of multiequations and a sequence of multiequations respectively, and other notations are consistent with Ref. [9]. As shown in Fig. 14, term indices are retained in overall procedure. From the end of the process in Fig. 14(e), we obtain the final substitution in this example,

{$002/$x2, t002/$x3, t004/$x1}.

The characteristics of this procedure are that the substituent for each variable is given by the term index and that it is facilitated by HTPS's internal implementation. Therefore, if we attach link information to each term index, we can retrieve them from the results of the unification. This process is appropriate to resolution and paramodulation under the connection graph method.

3.3.3 Lexical Ordering

HTPS has a built in lexical ordering mechanism for manipulating expressions. While the lexical ordering used in the preprocessing module does not use the paramodulation, it is indispensable for the lexical ordering in the base-level program. The ordering rules applied are as follows.

(a) Single terms are placed before nested terms.
(b) Variables are placed before non-variable terms.
(c) Non-variable terms are ordered alphabetically.
(d) The order of nested terms follows the order of their elements.

In order to realize these rules, the commutative and associative laws of function *f*,

(*f* $x $y)=(*f* $y $x)
(*f* $x (*f* $y $z))=(*f* $z (*f* $x $y))

are handled. In particular, the associative law is used from both sides. In the HTPS, its algorithm is developed which takes advantage of the fact that usage of these rules is determined uniquely by the position index of the objective term. The position index represents the position of the objective term in the expression using a sequence of symbols "1" and "*r*", where "1" denotes the term which is the first argument of its embracing function and "*r*" denotes the term which is the second one. For example, the position index of "a" in (And c (And (And b a) d)) is "rlr". Only the last two characters of this position index are used to determine the rule. The algorithm is shown in Fig. 15. The link inheritance mechanism after paramodulation described before completes the links needed for this algorithm.

3.4 The Proof Procedure

The proof is performed for each design specification (theorem), individually. The proof procedure of HTPS is the following non-deterministic algorithm.

(1) Input a theorem and add to the connection graph.
(2) Select one literal in the theorem and set it to TL.
  (a) If TL is empty then terminate successfully.
  (b) If TL has a link to a unit literal clause then resolve on that link, and go back to step (2).
  (c) Strategy determination
    (c-1) Select a strategy and set it to STRATEGY.
    (c-2) If STRATEGY is empty then enter interactive module 1.
    (c-3) If LINK exists then resolve or paramodulate on LINK, and go back to (2).
    (c-4) Tactics determination
      (c-4-1) Select a tactics and set it to TACTICS.
      (c-4-2) If TACTICS is empty then enter in-

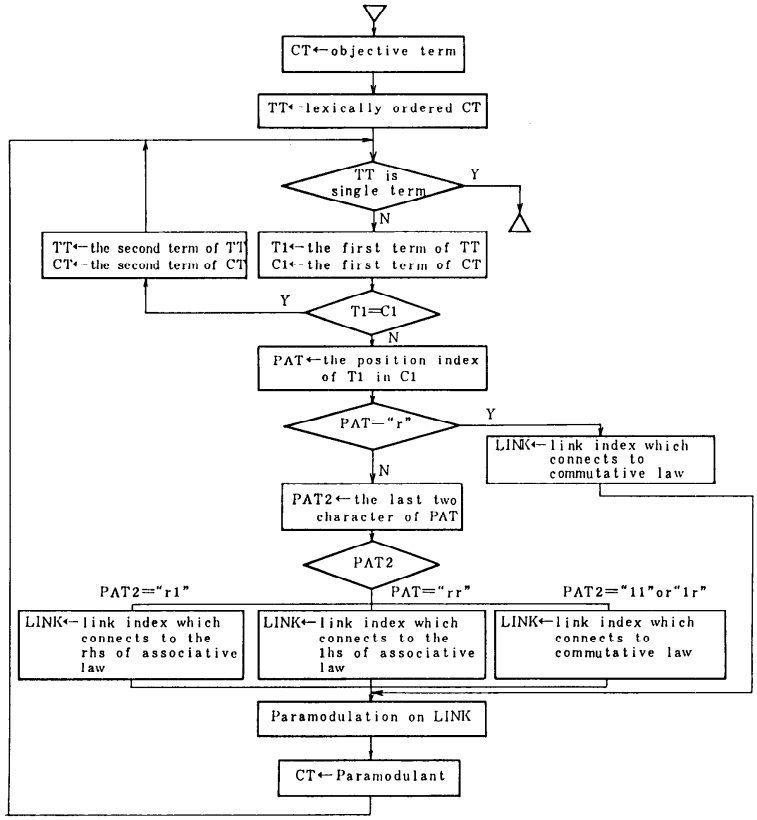


Fig. 15 Lexical ordering algorithm.

teractive module 2.

(c-4-3) Resolve or paramodulate according to TACTICS.

(c-4-4) If STRATEGY still exists go back to (c-4), else go back to (2).

Before starting a proof, the initial connection graph is constructed with the proposed design expressed in RTS and general axioms. Rules used for strategy and tactics determination are converted into executable functions and also loaded. Among the literals which constitute the theorem to be proved, one literal is selected in a depth-first manner in the first step of (2). Against this literal TL, all R-links are investigated to find the link that connects to a unit literal clause. If such a link exists, the number of literals in the theorem is certainly decreased by one, after the resolution on that link (b). In the process of strategy determination, if the link to be inferred is directly determined, the inference on that link is executed without searching tactics (c-3). As can be seen in the steps (c-4-1) through (C-4-4), tactics determination is performed under the selected strategy, and frequently the sequence of the tactics is selected and evaluated under the same strategy.

In the above procedure, if the strategy or tactics determination fails, interactive modules 1 or 2 are called. In these modules, information about the history of proof and the link connecting the current theorem, etc. are displayed on a demand basis. By selecting a link interactively, it is possible to continue the proof.

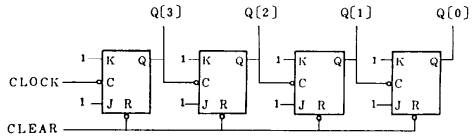
Currently, HTPS is implemented with Franzlisp and is run on the VAX11/750.

4. Verification Example Using HTPS

HTPS has been applied to the verification of some logic circuits. These examples include a shift register, synchronous counter and ripple counter. Since their verification is concerned mainly with the functional and logic level of design hierarchy, axioms for Boolean algebra and the related modification expressed in RTS are used. These axioms are shown in Fig. 16. They are cited from Refs. [1] and [3], with some of them modified slightly to be compatible with the expression employed. As shown in this figure, a unique name is associated with each axiom so that knowledge in the meta-level program can easily specify it. Additionally, a third term is added to some equality axioms in Fig. 16. These terms denote the normal direction from which the axiom is used, and also utilized in the evaluation of knowledge in the meta-level block. As for the

```
Expansion

((If (and (Rt (And $w (Not $x)) $a $y)
          (Rt (And $w $x) $a (Not $y)))
     (Rt $w $a (Eor $x $y)))                          st_expand)

((If (and (Rt $x $a $z) (Rt $y $a $z))
     (Rt (Or $x $y) $a $z))                           or_exp1)

Reduction

((If (Rt $x $a $z) (Rt (And $x $y) $a $z))            and_red1)

((If (Rt $y $a $z) (Rt (And $x $y) $a $z))            and_red2)

Concatenation

((Eq (Conc (Rf $x 0 0) (Rf $x 1 $m)) (Rf $x 0 $m))    concat_exp1  rhs)

((Eq (Conc (Rf $x 1 1) (Rf $x 2 $m)) (Rf $x 1 $m))    concat_exp2  rhs)

Increment

((Eq (Suc (Rf $x 0 3)) (Conc (Eor (Rf $x 0 0)
                                  (And (Rf $x 1 1) (And (Rf $x 2 2)
                                                        (Rf $x 3 3))))
                             (Suc (Rf $x 1 3))))    increment1    lhs)

Commutative law

((Eq (And $x $y) (And $y $x))                                  and_commutative)

((Eq (Or $x $y) (Or $y $x))                                    or_commutative)

Associative law

((Eq (And $x (And $y $z)) (And $z (And $x $y)))                and_associative1)

((Eq (Or $x (Or $y $z)) (Or $z (Or $x $y)))                    or_associative1)
```

Fig. 16 Examples of axioms used in the verification.



(a) Circuit diagram of ripple counter

```
(Rt (Not CLR) (Rf Q 3 3) c0)
(Rt (And CLR (Td CL)) (Rf Q 3 3) (Or (And c1 (Not (Rf Q 3 3)))
                                     (And (Not c1) (Rf Q 3 3))))
(Rt (And CLR (Not (Td CL))) (Rf Q 3 3) (Rf Q 3 3))

(Rt (Not CLR) (Rf Q 2 2) c0)
(Rt (And CLR (Td (Rf Q 3 3))) (Rf Q 2 2)
                              (Or (And c1 (Not (Rf Q 2 2)))
                                  (And (Not c1) (Rf Q 2 2))))
(Rt (And CLR (Not (Td (Rf Q 3 3)))) (Rf Q 2 2) (Rf Q 2 2))

(Rt (Not CLR) (Rf Q 1 1) c0)
(Rt (And CLR (Td (Rf Q 2 2))) (Rf Q 1 1)
                              (Or (And c1 (Not (Rf Q 1 1)))
                                  (And (Not c1) (Rf Q 1 1))))
(Rt (And CLR (Not (Td (Rf Q 2 2)))) (Rf Q 1 1) (Rf Q 1 1))

(Rt (Not CLR) (Rf Q 0 0) c0)
(Rt (And CLR (Td (Rf Q 1 1))) (Rf Q 0 0)
                              (Or (And c1 (Not (Rf Q 0 0)))
                                  (And (Not c1) (Rf Q 0 0))))
(Rt (And CLR (Not (Td (Rf Q 1 1)))) (Rf Q 0 0) (Rf Q 0 0))
```

(b) The proposed design

Fig. 17 Circuit diagram of ripple counter and the proposed design expression in RTS.

knowledge in the meta-level program itself, we implement 6 and 23 rules for strategy determination and tactics determination, respectively. Example of these rules were shown before in Figs. 7 and 9.

4.1 Verification of Ripple Counter

The circuit of the ripple counter and its proposed design expressed in RTS are shown in Fig. 17. The desired behavior of this circuit is represented with the following three theorems:

(Rt (Not CLR) (Rf Q 0 3) c0),
(Rt (And CLR (Td CL)) (Rf Q 0 3)
  (Suc (Rf Q 0 3))), and
(Rt (And CLR (Not (Td CL))) (Rf Q 0 3)
  (Rf Q 0 3)).

Although the circuit is simple, the proof is very complicated. This is because some of the proposed design have a transient expression of the first three bits in their condition part and these expressions must be replaced by ordinal expressions in order to get refutation.

In reference [1], this circuit was proved manually in 101 steps, excluding the process of simplification. The HTPS is applied to the verification of this circuit which can be proved fully and automatically in 200 steps. Paramodulation for simplification and canonicalization are involved. Part of this proof is shown in the appendix. The proof process for the second behavior, which requires 126 inference steps, depends mainly on the following strategies.

(a) Expansion of the theorem which has an expression of concatenation in its destination and source parts by using concatenation expansion rule (concat__expl).

(b) Replacement of the transition expression in the condition part of the theorem.

(c) Transformation of the condition part of the theorem to the Or-expression.

(d) Expansion of the theorem which has an Or-expression in its condition part (or__expl).

(e) Reduction of the theorem which has an And-expression in its condition part (and__red1, and__red2).

The name represented in parentheses denotes the name of the axiom used and the real expressions are found in Fig. 16. Strategy (a) is an expansion of the theorem to be proved, and so regarded as a top-down process. On the other hand, strategies (b), (c) and (d) are for transformation of the literal which is nominated as the potentially complimentary unifiable literal of the theorem and therefore regarded as a bottom-up process. In strategy (e), both top-down and bottom-up processes are combined to yield refutation.

4.2 Results

Through these verification examples, HTPS generates appropriate proof guidances and performs for each proof effectively. This effectiveness is realized through the following two factors.

(1) The number of clauses generated during proofs is kept small, and it is often the case that only the minimum number of clauses is selectively generated to establish the proof.

(2) The number of unifications needed for the proof is also kept small.

Concerning factor (1), the meta-level program makes the main contribution to this. Effective proof guidances

Table 3 Comparison of verification data for 4-bit shift register design.

	Number of Clauses generated	Number of Unifications
Proposed Method	55	3751
Conventional Method*	695	6496

*Data from the proof in Ref. [4].

are able to be generated with the use of knowledge which relates the characteristics of the theorem to be proved and the proposed design to proof direction explicitly coupled with knowledge about a role of each axiom.

As for (2), HTPS owes its efficiency not only to its meta-level mechanism but also to the connection graph method. In this method, once the initial graph is constructed all the information as to which literals are potentially resolvable is presented and no further searching for unifiable complimentary literals is needed.

In order to show the usefulness of HTPS, the verification result of 4-bit shift register design by the proposed method is compared with that by a conventional method in Ref. [4] and shown in Table 3. The figures in the table suggest that HTPS can save a considerable amount of search time in theorem proving and indirectly show the efficiency of HTPS.

Although examples so far are restricted to simple circuits, difficulties which arise from an increase of circuit complexity can be handled by an increase of rules in the meta-level program. Therefore, the inference mechanism realized in HTPS is considered to be both general and powerful.

5. Conclusions

A theorem proving system HTPS, for logic design verification has been developed. HTPS has a hierarchical structure, meta-level and base-level, and has an advantage of using domain specific knowledge explicitly in generating proof plans in the meta-level. The connection graph method was modified in its link inheritance mechanism and used as a proof strategy in HTPS. Adoption of this method was useful in making the hierarchical structure straightforward.

In application of the verification to simple circuits, HTPS demonstrated its usefulness and effectiveness.

Through these applications, we are convinced that the utilization of domain specific knowledge in the theorem proving process in the way HTPS does it will be a key technique to the logic design verification by the theorem proving method.

Acknowledgements

The authors are deeply indebted to Dr. H. Motoda of Advanced Research Laboratory, Hitachi, Ltd. for useful discussions.

References

1. H. G. BARROW. VERIFY: A Program for Proving Correctness of Digital Hardware Designs, *Artificial Intelligence* **24**, p. 437 (1984).
2. T. J. WAGNER. Hardware Verification, *Ph.D.* Dissertation, CSD, Standford Univ. No. STAN-CS-77-632 (Sept. 1977).
3. R. WEYHRAUGH. Prolegomena to a Theory of Mechanized Formal Reasoning, Artificial Intelligence, **13**, p. 133 (1980).
4. A. S. WOJCIK. Formal Design Verification of Digital System, *Proc. 20th DA Conf.* p. 228 (Jun. 1983).
5. E. L. LUSK, et al. Logic Machine Architecture Kernel Functions, *Lecture Notes in Computer Science*, 138, Springer Verlag (1982).
6. A. S. WOJCIK, et al. A Formal Design Verification System Based on an Automated Reasoning System, *Proc. 21th DA Conf.* p. 641 (Jun. 1984).
7. R. KOWALSKI. A Proof Procedure Using Connection Graphs, *JACM* **22** (1975).
8. J. SIEKMANN and G. WRIGHTSON. Paramodulated Connection Graphs, *Acta Informatica*, **13**, p. 67 (1980).
9. A. MARTELLI and U. MONTANARI. An Efficient Unification Algorithm, *ACM Trans. on Programming Lang. and Sys.*, **4**, 2 (1982).
10. J. B. MORRIS. E-Resolution—Extension of Resolution to Include the Equality Relation, in Automated Reasoning 2, *Classical Papers on Computational Logic* 1967–1970, Springer Verlag (1980).

(Received October 13, 1986; revised May 26, 1987)

Appendix

The proof process of the second specification of the ripple counter in section 4.1 is partially described in Fig. A1 through Fig. A3. In Fig. A1, the clause index c06295 is a unit literal clause which corresponds to the specification. In each proof step, the selected strategy and also tactics if they exist, are displayed followed by the inference rule applied and its arguments. In strategies S1, S2, S3, S4 and S5 the direct links has been determined, on the other hand, in strategies S6 and S7, more than two tactics are selected.

As mentioned in section 4.1, this proof is done in 126 steps.

```
-> c06295
                [STRATEGY]:Expand-DT                    (S1)
  << Paramodulation >>
     (Rf Q 0 3)
  :in:
   (not (Rt (And CLR (Td CL))(Rf Q 0 3)(Suc (Rf Q 0 3))))
  :with:
   (Eq (Conc (Rf $01106 0 0)(Rf $01106 1 3))(Rf $01106 0 3))

                [STRATEGY]:Expand-DT                    (S2)
  << Paramodulation >>
     (Rf Q 1 3)
  :in:
   (not (Rt (And CLR (Td CL)
            (Conc (Rf Q 0 0)(Rf Q 1 3))
            (Suc (Rf Q 0 3))))

  :with:
   (Eq (Conc (Rf $01127 1 1)(Rf $01127 2 3))(Rf $01127 1 3))

                [STRATEGY]:Expand-DT                    (S3)
  << Paramodulation >>
     (Rf Q 2 3)
  :in:
   (not (Rt (And CLR (Td CL))
            (Conc (Rf Q 0 0)(Conc (Rf Q 1 1)(Rf Q 2 3)))
            (Suc (Rf Q 0 3)) ))

  :with:
  (Eq (Conc (Rf $01155 2 2)(Rf $01155 3 3))(Rf $01155 2 3))

                [STRATEGY]:Expand-ST                    (S4)
  << Paramodulation >>
     (Suc (Rf Q 0 3))
  :in:
   (not (Rt (And CLR (Td CL))
            (Conc (Rf Q 0 0)
                  (Conc (Rf Q 1 1)(Conc (Rf Q 2 2)(Rf Q 3 3)))
            (Suc (Rf Q 0 3)))))

  :with:
   (Eq (Suc (Rf $01276 0 3))
       (Conc (Eor (Rf $01276 0 0)
                  (And (Rf $01276 1 1)
                       (And (Rf $0176 2 2)(Rf $01276 3 3))
            (Suc (Rf $01276 1 3))))))

                [STRATEGY]:Expand-TL1                   (S5)
  << Resolution >>
   (not (Rt (And CLR (Td CL))
            (Conc (Rf Q 0 0)
                  (Conc (Rf Q 1 1)(Conc (Rf Q 2 2)(Rf Q 3 3)))
            (Conc (Eor (Rf Q 0 0)
                       (And (Rf Q 1 1)(And (Rf Q 2 2)(Rf Q 3 3)))
                  (Suc (Rf Q 1 3)))))))

  :with:
   (Rt $00128 (Conc $00130 $00131)(Conc $00133 S00134))
   (not (Rt $00128 $00130 $00133))
   (not (Rt $00128 $00131 $00134))

                [STRATEGY]:Expand-TL2                   (S6)
                  [TACTICS]:Trule-7                     (T1)
  << Paramodulation >>
     (Eor (Rf Q 0 0)(And (Rf Q 1 1)(And (Rf Q 2 2)(Rf Q 3 3))))
  :in:
   (not (Rt (And CLR (Td CL))
            (Rf Q 0 0)
            (Eor (Rf Q 0 0)
                 (And (Rf Q 1 1)(And (Rf Q 2 2)(Rf Q 3 3))))))

  :with:
   (Eqa (Eor $04208 $04209)(Eor $04209 $04208))
```

Fig. A1 Verification process-1.

```
                [TACTICS]:Trule-8                       (T2)
  << Resolution >>
   (not (Rt (And CLR (Td CL))
            (Rf Q 0 0)
            (Eor (And (Rf Q 1 1)(And (Rf Q 2 2)(Rf Q 3 3)))
                 (Rf Q 0 0))))

   (not (Rt (And CLR (Td CL))
            (Conc (Rf Q 1 1)(Conc (Rf Q 2 2)(Rf Q 3 3)))
            (Suc (Rf Q 1 3))))

  :with:
   (Rt $00286 $00287 (Eor $00289 $00290))
   (not (Rt (And $00286 (Not $00289)) $00287 $00290))
   (not (Rt (And $00286 $00289) $00287 (Not $00290)))

              [STRATEGY]:Unite-BC
                [TACTICS]:Trule-10                      (T3)
  << Paramodulation >>
     (Td (Rf Q 1 1))
  :in:
   (Rt (And CLR (Not (Td (Rf Q 1 1))))(Rf Q 0 0)(Rf Q 0 0))
  :with:
   (Eq (Td (Rf Q 1 1))(And (Rf Q 1 1)(And CLR (Td (Rf Q 2 2)))))

                [TACTICS]:Trule-10                      (T4)
  << Paramodulation >>
     (Td (Rf Q 2 2))
  :in:
   (Rt (And CLR (Not (And (Rf Q 1 1)(And CLR (Td (Rf Q 2 2))))))
       (Rf Q 0 0)
       (Rf Q 0 0)  )

  :with:
   (Eq (Td (Rf Q 2 2))(And (Rf Q 2 2)(And CLR (Td (Rf Q 3 3)))))

                [TACTICS]:Trule-10                      (T5)
  << Paramodulation >>
     (Td (Rf Q 3 3))
  :in:
   (Rt (And CLR
            (Not (And (Rf Q 1 1)
                      (And CLR
                           (And (Rf Q 2 2)(And CLR (Td (Rf Q 3 3))))
       (Rf Q 0 0)
       (Rf Q 0 0)))))

  :with:
   (Eq (Td (Rf Q 3 3))(And (Rf Q 3 3)(And CLR (Td CL))))

                [TACTICS]:Trule-21                      (T6)
  << Paramodulation >>
     (And (Rf Q 1 1)
        (And CLR
             (And (Rf Q 2 2)
                  (And CLR (And (Rf Q 3 3)(And CLR (Td CL))))))))

  :in:
   (Rt (And CLR
            (Not (And (Rf Q 1 1)
                      (And CLR
                           (And (Rf Q 2 2)
                                (And CLR
                                     (And (Rf Q 3 3)
                                          (And CLR (Td CL))
       (Rf Q 0 0)
       (Rf Q 0 0)))))))))

  :with:
   (Eq (And $04246 (And $04248 $04249))
       (And $04249 (And $04246 $04248)))
   )
                .       .       .
```

Fig. A2 Verification process-2.

```
                [TACTICS]:Trule-16                              (T7)

  << Paramodulation >>
     (And (And CLR (Td CL))(Rf Q 3 3))
  :in:
   (not (Rt (And (And CLR (Td CL))(Rf Q 3 3))
            (Rf Q 2 2)
            (Not (Rf Q 2 2))))

  :with:
   (Eq (And (And $04556 $04557) $04558)
       (And (And $04557 $04558) $04556))

  << Paramodulation >>
     (And (And (Td CL)(Rf Q 3 3)) CLR)
  :in:
   (not (Rt (And (And (Td CL)(Rf Q 3 3)) CLR)
            (Rf Q 2 2)
            (Not (Rf Q 2 2))))

  :with:
   (Eqa (And $04033 $04034)(And $04034 $04033))

  << Paramodulation >>
     (And (Td CL)(Rf Q 3 3))
  :in:
   (not (Rt (And CLR (And (Td CL)(Rf Q 3 3)))
            (Rf Q 2 2)
            (Not (Rf Q 2 2))))

  :with:
   (Eqa (And $04033 $04034)(And $04034 $04033))

  << Paramodulation >>
     (And (Td CL)(Rf Q 3 3))
  :in:
   (Rt (And CLR (And (Td CL)(Rf Q 3 3)))
       (Rf Q 2 2)
       (Not (Rf Q 2 2)))

  :with:
   (Eqa (And $04033 $04034)(And $04034 $04033))
                [TACTICS]:Trule-23                              (T8)
  << Resolution >>
   (Rt (And CLR (And (Rf Q 3 3)(Td CL)))
       (Rf Q 2 2)
       (Not (Rf Q 2 2)))

  :with:
   (not (Rt (And CLR (And (Rf Q 3 3)(Td CL)))
            (Rf Q 2 2)
            (Not (Rf Q 2 2))))

   (not (Rt (And CLR (Td CL))(Rf Q 3 3)(Not (Rf Q 3 3))))

---- [ THE CURRENT THEOREM HAS AN R-LINK TO AN UNIT LITERAL CLAUSE ]

  << Resolution >>
   (Rt (And CLR (Td CL))(Rf Q 3 3)(Not (Rf Q 3 3)))
  :with:
   (not (Rt (And CLR (Td CL))(Rf Q 3 3)(Not (Rf Q 3 3))))

  *** Contradiction ***
          ---- [ THEOREM ] ----
    ********************************************
    (Rt (And CLR (Td CL))(Rf Q 0 3)(Suc (Rf Q 0 3)))
    ********************************************
              HAS BEEN PROVED.
  done
->
```

Fig. A3 Verification process-3.