Vol. 35 No. 12 Transactions of Information Processing Society of Japan Dec. 1994

Regular Paper

Design Verification Based on Theorem-Proving Technique for Sequential Control Circuits with Timing Coordination

NAOYUKI YAMADA,[†] YOSHIKATSU UEDA,[††] JUNKO ITO,[††]
TOMOHARU NAKAMURA,[†††] JUNICHI YOSHIZAWA[††††]
and SATOSHI MATSUDA[††††]

A design verification method for sequential control circuits with timing coordination has been developed. Unlike prevailing numeric simulations, this method is based on a theorem-proving technique. Sequential control circuits realize their target functions by coordinating the asynchronously inputted signals. In order to verify timing coordination efficiently, the control strategy which combines hyperresolution and a connection graph method with attached procedures for handling time variables symbolically has been developed. This method facilitates not only the verification of timing coordination but also the extraction of intended behaviours from proposed designs. The developed method was applied to verification of an auto-reclosing circuit in an electric power substation which contains about 40 components. The verification for each specification was executed correctly in about 1 minute on a mainframe computer. The developed verification method was judged to be useful and efficient for practical use.

1. Introduction

Verification is an indispensable process in engineering design. For electric power circuits, verification of the protection system, which is implemented by sequential control logics, is one of the central issues in preventing trouble. Generally sequential control circuits realize their functions by coordinating signals inputted asynchronously from the control environment, therefore verification of timing coordination is required. Verifications are most often done by human experts, although numeric simulations are applied in selected cases. Therefore, a verification method which is free from human errors and guarantees complete verification is strongly desired.

A design verification method based on theorem-proving appears to be the best approach to take. Unlike numeric simulations, which are widely used technique for verification in other fields, test data generation is not needed and a complete verification is guaranteed. However, verification based on theorem-proving is generally time-consuming[1] and does not easily deal with time varying signals easily. Recently, extensive research on applying temporal logic to hardware verification has been carried out,[2)-4)] especially the symbolic model checking method for temporal logic has been applied to the verification of practical design.[5] However, it is still not easy to handle time variable explicitly in their verification.

This paper describes a verification method, based on a theorem-proving technique in classical predicate logic, which allows handling of time varying signals of sequential control circuits. The main advantage of this method is that it can assist the verification even when the design specification does not contain sufficient timing conditions. This is performed by extracting timing relations between the input and output signals from the verification procedure of the proposed design. The next section explains the verification problem for sequential control circuits. Section 3 outlines the developed verification system. Section 4 describes the theorem-proving method in detail and section 5 discusses an application example.

† Energy Research Laboratory, Hitachi, Ltd.
7-2-1 Omika-cho, Hitachi, Ibaraki, 316
†† Systems Engineering Division, Hitachi, Ltd.
6 Kanda-surugadai 4 chome, Chiyoda-ku, Tokyo 101
††† Kokubu Works, Hitachi, Ltd.
1-1 Kokubu-cho, 1 chome, Hitachi, Ibaraki 316
†††† Computer & Communication Research Center, The Tokyo Electric Power Co.
1-4-10 Irifune, Chuo-ku, Tokyo 104

2. Design Verification of Sequential Control Circuits

2.1 Verification problems

In electric power circuits, most of the control and protection systems are implemented by sequential logics. Among them, almost all the logics except for operation interlock contain some constraints for timing coordination. These constraints include controls on waiting time for confirmation of phenomena and controls which take account of setting or resetting time.

As a typical example of such sequential control logics, **Fig. 1** shows a design diagram for an auto-reclosing circuit in an electric power substation. The circuit contains a number of memories and timers as well as ordinal AND and OR elements. This circuit automatically restores the transmission line to some preset duration after a trouble occurrence on it. In practice, four design specifications are given to designers in order to realize this function. One of the main specifications is as follows.

If the "reclosing start" signal is on
when the "reclosing in service" is on
and the "main protection is service" is on
and the "CB 3 pole closed" is on
and the "normal air pressure" is on
and the "loop condition" is on
then generate the "reclose command" after 200 milliseconds.

With the specifications, expert designers design the circuits by referring to design knowledge and their expertise. These circuits are then converted into programs and installed into micro computers.

The design verification problem here is to verify the proposed design (Fig. 1) against each of the specifications. This verification is effective for reducing the work in the design process, which otherwise requires a trial and error procedure, as well as enhancing design reliability.

2.2 Requirements for design verification

The design verification method should consider the following requirements.

(1) A necessity to handle time varying data explicitly.

One of the main characteristics of sequential control circuits is that the input signals to the circuits change asynchronously according to changes in the control environment. Therefore, time varying data should be explicitly represented and processed in the verification.

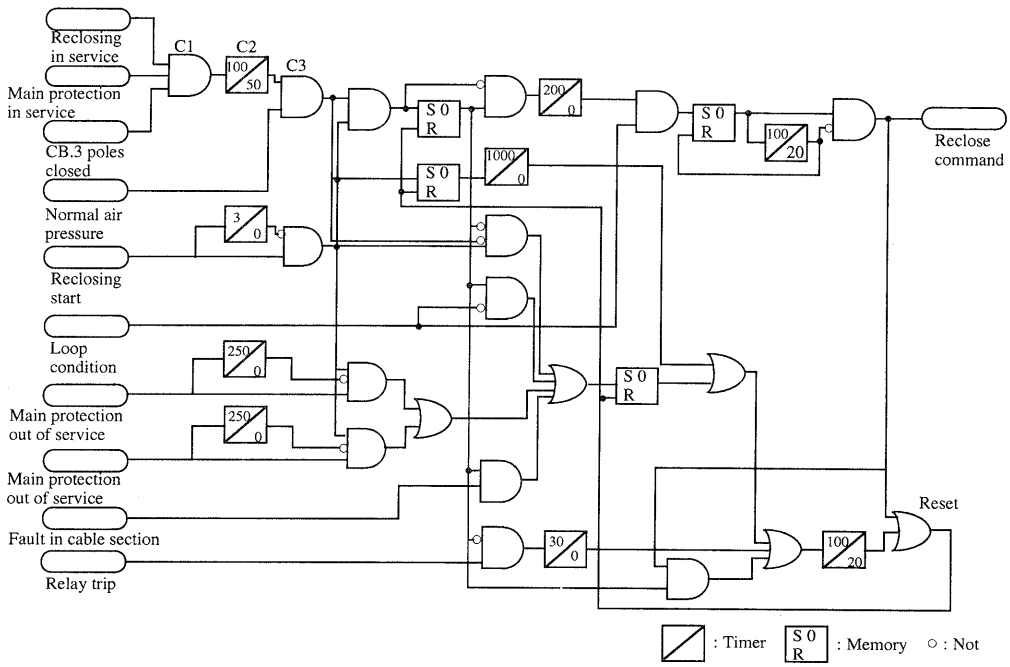


Fig. 1 Auto-reclosing circuit.

(2) A necessity to support the verification when not all the timing constraints are provided.

As mentioned above, the input signals to the sequential control circuits change asynchronously. Therefore, the design specifications except for those which prescribe the primary functions do not contain sufficient timing constraints. Thus, it is necessary not only to verify a specification with complete timing constraints, but also to support verification of a specification with insufficient timing constraints. For this purpose, the verification method should be able to extract the behaviour from the proposed design and show it to the designers as well as to decide the correctness of the proposed design.

The design verification method exploited in this paper meets these requirements.

3. Design Verification System

3.1 Problem formulation

The fundamental idea of design verification based on the theorem-proving method is that if the proposed design is correct, the design specifications can be logically derived from that design.

In the verification problem of sequential control circuits which is discussed in this paper, the design specifications are described as relations between the primary input and output signals of the whole circuit. On the other hand, the proposed design is a designed diagram, which is specified by type description of each component and their connectivities. Then, behaviour rules of each component are used as the axioms which relate the design specifications and the proposed design. Thus, the verification is formulated as a theorem-proving problem which proves the design specification under the axioms of the proposed design and behaviour rules of the components.

In order to mechanize this problem, a representation scheme of signals, efficient description of axioms, and efficient control of theorem-proving should be developed, taking account of the requirements mentioned above.

3.2 Signal representation

A signal representation scheme for sequential control circuits should offer an ability to express not only instants of time but also intervals of time, since the circuits involve timers and memories as well as the usual logical-AND and logical-OR elements. For this reason, a signal representation scheme which describes signals as events is adopted. The notion of event taken here is rather informal. It is used to represent state changes with both instant and some time periods. Then, all the signals which may occur in sequential control circuits can be expressed by fundamental three events; no event, step, and pulse, and their combinations. **Figure 2** shows a description of these fundamental events and the corresponding time charts. Note that this representation scheme can save the number of behaviour rules.

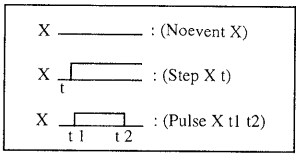


Fig. 2 Event descriptions.

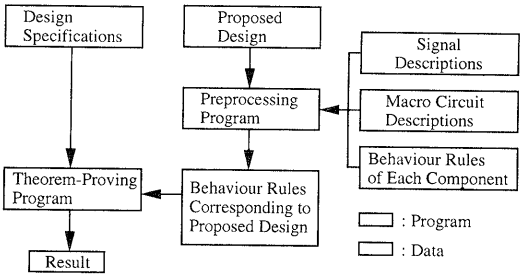


Fig. 3 Configuration of verification system.

3.3 System organization

The configuration of the verification system is shown in **Fig. 3**. Design specifications are represented in a formula of predicate calculus by using the signal representation scheme mentioned above and given by the user. A proposed design is described by its component specifications and their connectivities. Their description formats are as follows.

Component:

(**Comp-type** component-ID its-type)

Connectivity:

(**Conn** (**Outport** port-ID component-ID)
(**Inport** port-ID component-ID))

Each component is declared by the predicate "**Comp-type**" followed by a component identification and its type. Connectivity relations are described by using the predicate

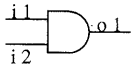
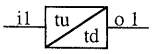
"**Conn**" and a pair of port specifications. Part of the proposed design of the auto-reclosing circuit in Fig. 1 is shown in **Fig. 4**. These data are transferred from a CAD system data base.

The behaviour rules of each component are represented as an event transformation. Examples of behaviour rules are also shown in **Table 1**, where a symbol which starts with "$" denotes a variable. Each component has more than one behaviour rule according to its input signals and logical function, and these rules are used as axioms in the verification. Currently, the system contains only behaviour rules which relate fundamental events.

```
Component types :
        (Comp-type P1 panel)
        (Comp-type C1 and30)
        (Comp-type C2 tr)
Connectivity :
        (Conn (Outport 1 P1) (Inport 1 C1))
        (Conn (Outport 1 C1) (Inport 1 C2))
        panel : control panel
        and30: 3-input And
            tr : timer
```

Fig. 4 Example design descriptions.

Table 1 Examples of behaviour rules.

Type	Symbol	Behaviour rule
and20		(if (and (Step i1 $t1) (Step i2 $t2) (≧ $t1 $t2)) (Step o1 $t1) (if (and (Step i1 $t1) (Pulse i2 $t21 $t22) (≧ $t1 $t22)) (noevent o1)
tr		(if (and (Step i1 $t1) (= $tx (+ $t1 tu))) (Step o1 $tx)) (if (and (Pulse i1 $t11 $t12) (= $tx (- $t12 $t11)) (< $tx tu)) (noevent o1))

tu : Delay time in transition OFF->ON
td : Delay time in transition ON->OFF

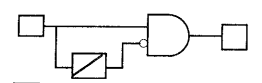
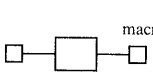
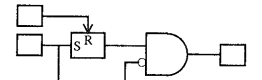
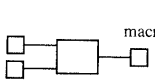
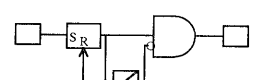
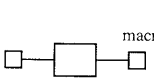
In order to keep the search space of theorem-proving small, it is desirable to reduce the number of axioms. In the developed system, two main techniques for reduction of the number of axioms are realized. One is to use a macro circuit description extensively. Generally, macro circuits are used in designing sequential control circuits to realize special behaviours. A pulse generation circuit, for example, is one such circuit. Therefore, a considerable reduction of the number of axioms is expected, by using the behaviour rules of such macro circuits directly, instead of those for primitive components. Preceding to the verification, these macro circuits are identified by their configurations in the proposed design and the corresponding descriptions are replaced with those of macro circuits. Currently, the system prepares three macro circuits, which are shown in **Table 2**.

The other technique to reduce the axioms is not to use the connectivity rules explicitly. When the connectivities are to be used directly in the verification, the following connectivity rule is needed to propagate a signal :

(If (and (**Step** (**Outport** 1 C1) $t)
(**Conn** (**Outport** 1 C1) (**Inport** 1 C2)))
(**Step** (**Inport** 1 C2) $t))

Thus, the theorem-proving program has to spend

Table 2 Macro circuit definitions.

	Circuit description	Macro component	Function
1			Pulse generation
2			Delay according to the input signal
3			Pulse generation

much time to follow trivial signal propagation. One way to make the connectivity rules of no use is to use line names. The signals of both the ports which are declared to be connected with the connectivity relation can be represented by a single expression, i.e. a signal name on the line. Accordingly, the port descriptions in the behaviour rules are replaced with the corresponding line names. As a result, the proposed design and behaviour rule of each component are combined, resulting in behaviour rules which correspond to the proposed design. The preprocessing program provides this conversion as illustrated in Fig. 3. Here, signal descriptions are those which specify the port attributes, primary inputs and primary outputs; specifications of primary inputs and outputs are used to specify the boundaries of the proposed design to be verified and port attributes provide the specifications of intermediate points which may be given special meanings in the circuit. If the circuit can be functionally decomposed into several partial circuits, the correspondence to each outport is also specified through these descriptions. In that case, hierarchical verification is performed.

4. Design Verification Method

4.1 Theorem-proving method

It is well known that for efficient theorem-proving, the control strategy should reflect the characteristics of the problem to be proved. From the description above, the theorem to be proved, i.e. each design specification, is an event transformation between the primary input and output signals; the axioms used for the proof are also an event transformation between the input and output signals of each component. This suggests that the control process of theorem-proving becomes a chaining of axioms in the same representational level, i.e. the same kind of behaviour rules; additionally the chaining of axioms implies signal propagations. Therefore, hyperresolution which provides an efficient forward chaining mechanism is adopted as a main control strategy. Furthermore, its efficiency is enhanced by combining it with a connection graph method.[6),7)]

Each behaviour rule may contain equality and/or inequality literals to express the timing relation of each event as shown in Table 1. These relations should be processed by the attached procedures in the theorem-proving procedure, since it is impractical to prepare all the timing relations needed for a proof. An attached procedure mechanism facilitates the symbolic manipulation of timing relations as well as their numeric calculation. Furthermore, this mechanism coupled with the above mentioned control strategies allows theorem-proving to be done under multiple hypotheses, which is indispensable for the verifications with insufficient timing constraints as explained later.

(1) Control strategy

Hyperresolution is a kind of semantic resolution, which introduces an interpretation for each atomic formula and decomposes all the clauses used for a proof into two groups.[1)] In particular, positive hyperresolution uses an interpretation in which all the literals are negative decomposing the clauses into positive and non-positive clauses, where a positive clause is a clause which consists of only affirmative, i.e. positive literals. Then, only the resolution between the clauses of different groups is allowed; as such it restricts the possible resolutions. The resulting resolution procedure is similar to forward chaining in production systems.

Although hyperresolution is efficient, considerable steps are used in finding complementary literals from the two groups of clauses. Some of this inefficiency can be eliminated by combining it with a connection graph method as the second control strategy. The connection graph method does not require a search for complementary literals at each proof step, instead it inherits the resolvabilities from the parent clauses.

The theorem-proving process in the verification system is shown in **Fig. 5**. After producing the initial connection graph, all the clauses in the graph are grouped into positive and non-positive clauses. Then, resolution is applied in two ways; the left half procedures, i.e. ③, ④, ⑤ in Fig. 5 handle the resolution between all of the positive clauses generated up to that time including the primary input positive clauses and non-positive clauses generated in the proving process and the right half procedures, i.e. ⑥, ⑦, ⑧, and ⑨ perform the resolutions between the generated positive clauses and the primary input non-positive clauses. The number of resolutions in ③ and ⑦ depends on the

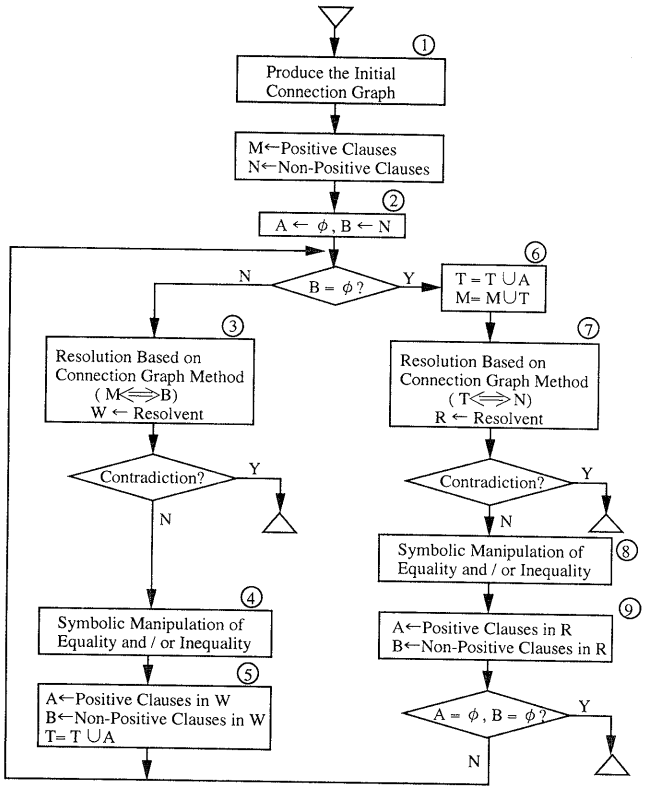


Fig. 5 Theorem-proving process.

problem, however, the right half procedure is executed only once after all the resolutions between M and B (③) are performed, since the newly generated positive clauses are added to M in ⑥. The resulting procedure is able to cover all the possible resolutions. From the viewpoint of signal propagation, the right hand side procedures correspond to picking up the components which can propagate the generated signals. The left hand side procedures checks all the conditions for propagating the signals and actually generates new signals.

Usually, hyperresolution is coupled with a predicate ordering method for further efficiency. The predicate ordering method restricts a resolution to the literal which has a high priority. In the current system, the following predicate ordering is used for convenience.

Pulse > Step > No event > Arithmetic expression

As described above, hyperresolution controls the theorem-proving globally, whereas the connection graph method controls it locally. After each resolution, symbolic manipulation of equality and inequality literals is invoked for handling timing relations.

(2) Equality and inequality manipulation

As shown in Table 1, each axiom, i.e. behaviour rule contains equality and/or inequality literals. Equality literals are used to assign new variables to the result of a simple arithmetic calculation of already specified variables. On the other hand, inequalities prescribe greater and smaller relations between already specified variables.

In the theorem-proving procedure for design verification explained above, there are three cases in which these literals are manipulated.

(a) Expression with rigid numerals

Examples (=tl (+2.0 10.0)), (>5.0 0.5)

(b) Symbolic expressions with sufficient constraints

Example (≧tl 10.0), with (≧tl 15.0)

(c) Symbolic expressions with insufficient

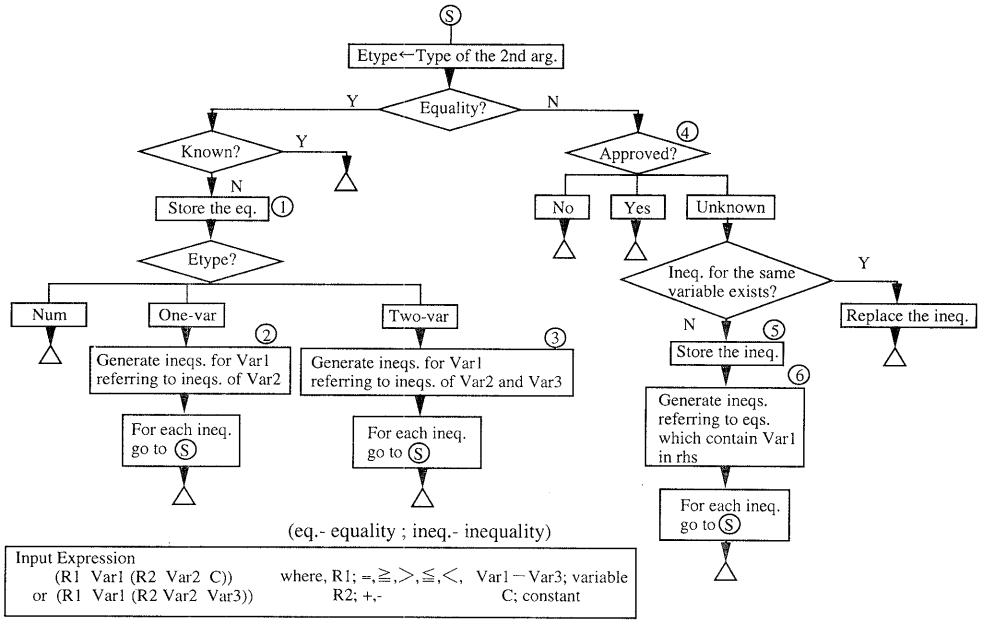


Fig. 6 Flowchart of equality/inequality storing.

constraints

Example (≧t1 (+t2 0.5))

Case (a) corresponds to the verification in which the design specifications are fully specified numerically. On the other hand, the specifications in case (b) are represented by using symbolic time constants with constraints between them. In case (c), the specifications are represented just like in (b), but with insufficient constraints.

From the viewpoint of theorem-proving, both cases (a) and (b) can be handled only by the usual attached procedure. In case (a), the expression is directly evaluated numerically and the truth value of the corresponding literal is obtained. The expression in case (b) is processed symbolically. Since all the constraints are supposed to be given, the truth value of the expression can be determined by referring to these constraints. As for case (c), the truth value of a given inequality cannot always be determined and the value "unknown" should be returned from the inequality handling program. These symbolic manipulations of equalities and inequalities may be performed by an ordinary symbolic manipulation system. However, it should be portable enough for the use as attached procedure in a theorem-proving process. For this purpose, a program has been developed which only processes equalities and inequalities symbolically. In order to manipulate equalities and inequalities in the theorem-proving process, two functions are needed. The first receives an equality or inequality expression and keep the relationships among the variables consistent. The second receives an inequality and checks whether the specified inequality is valid, invalid, or unknown by referring to the relations among the variables. When storing an inequality, it is necessary to check the validity of the inequality. Therefore, the first function includes the second one. The flowchart of the program which stores an equality or inequality is shown in **Fig. 6**. As mentioned before, the program is a special purpose one and it only accepts the expressions (R1 Var1 Num), (R1 Var1 (R2 Var2 Num)), and (R1 Var1 (R2 Var2 Var3)), where R1={=, >, ≧, <, ≦}, R2={+, −}, Var1-Var3 are variables, and Num is a numeric value. These expressions are referred to as Nun, One-var, and Two-var in their "Etype" classification. The underlying idea is that when storing an equality or an inequality the program generates all the possible relations concerned with the specified variable based on the already given relations, making an inequality consulta-

tion easy. In Fig. 6, procedures ② and ③ generate inequalities by referring to the inequalities of the variable which appears in the specified equality. On the other hand, inequalities concerned with the variable in the specified inequality are generated by referring to the corresponding equalities in step ⑥. The inequality checking function corresponds to the step ④. It collects only the inequalities corresponding to the variables specified in the expression and decides the return values, i.e. No, Yes, and Unknown.

These programs are invoked in steps ④ and ⑧ in Fig. 5 as an attached procedure and they return truth values for inequality literals.

(3) Theorem-proving under multiple hypotheses

In case (c) above, it is necessary to hypothesize the given inequality in order to continue theorem-proving. Then it is easily anticipated that theorem-proving should be able to be performed under multiple hypotheses. This is the very mechanism which allows extraction of behaviours from the proposed design. That is to say, under each group of hypotheses, the event propagates from the primary inputs to the primary output generating the possible behaviour of the proposed design.

The resolution procedure based on hyper-resolution and the connection graph method are improved to perform this theorem-proving. The equality and inequality handling program shown in Fig. 6 is also modified to handle hypothesis processing. The two main modifications are that the resolution process is executed under some hypotheses restricting the available clauses and new hypotheses are generated in the course of resolution. The improvements are shown in **Fig. 7** for the resolution procedure and in **Fig. 8**, for the link inheritance procedure.

In steps (a) and (b) in Fig. 7, if there is no inequality literal or no ground inequality literal in the resolvent, it is of no use to generate a hypothesis in step (b-1-1). On the other hand, when the attached procedure returns "Unknown" in (b-1-2) meaning that there are not enough conditions for checking the inequality, a new hypothesis about the inequality, i.e. timing relation is generated and the resulting theorem-proving procedures are executed under this

```
(1) Get hypothesis CP attached to the link.
(2) Generate a resolvent.
(3) If it generates contradiction, terminate with failure
    else
    (a) if there exists some partial ground equality literal L1,
        store the equality and delete L1 from the resolvent,
    (b) if there exists some ground inequality literal L2,
        (b-1) if L2 is false under CP, terminate with failure,
              else
              (b-1-1) if L2 is true under CP, delete L2 from the resolvent,
              (b-1-2) if L2 is unknown under CP, generate a new hypothesis
                      L2 and set it to CP,
              (b-1-3) check the inheritance of R-links,
        else
    (c) check the inheritance of R-links.
```

Fig. 7 Resolution procedure.

```
(1) For each literal L in the resolvent,
    for each link x in the parent literal of L,
        set IP the hypothesis attached to x
        if IP is not included in the ancestor hypotheses of CP,
          (a) if IP is false under CP, terminate,
              else
              (a-1) if IP is unknown under CP, generate a new hypothesis
                    IP under CP and set it to CP,
              (a-2) inherit the link x,
        else inherit the link x.
```

Fig. 8 Link inheritance procedure.

hypothesis.

As for the link inheritance, the link is inherited when the hypothesis attached to the link is found to be included in the ancestor of the current hypothesis. Otherwise, the hypothesis, i.e. inequality is checked under the current hypothesis. The result "unknown" in this case implies that the hypothesis attached to the link is independent of the current hypothesis, therefore, these hypotheses are merged into a single hypothesis in step (a-1).

4.2 Verification process

As mentioned earlier, design specifications for sequential control circuits are given as relations between the primary input and output signals which are described as events. The verification processes based on the theorem-proving method are regarded as event propagation from the primary inputs to the primary output. Therefore, each verification process is terminated irrespective of the correctness of proposed design. If a proposed design is correct, then the verification terminates successfully, i.e. the negation of the specification generates contradiction. On the other hand, when the proposed design has some flaws, the verification process terminates with some primary output event which can not refute the given specification.

Since the theorem-proving procedure of the verification system is capable of generating

hypotheses concerning the timing relation if needed, the following specifications are accepted for verification.

(1) All the signals are represented with rigid numerals.

(2) Some of the signals are represented with symbolic time constants and some constraints among them are missing.

In case (1), the system responds "true" if the proposed design is correct and "false" otherwise. As for case (2), if all the constraints concerning time constants are given, the output of the system is the same as that in case (1). If no constraints are given or the given constraints are insufficient, the system automatically generates the possible timing relations which satisfy the given specification. The result of the verification, then is an extracted behaviour, that is pairs of primary inputs and output with timing conditions.

5. Verification Example

The current verification system was implemented by VOS3LISP (Common lisp). The system was applied to verification of the auto-reclosing circuit shown in Fig. 1 and it verified all of the design specifications given to this circuit.

As a typical example of behaviour extraction from the proposed design, the verification result of a specification which relates the six primary input signals, i.e. "Reclosing in service," "Main protection in service," "CB 3 poles closed," "Normal air pressure," "Reclosing start," "Loop condition," and the output signal "Reclose command" is shown in **Fig. 9**. The design specification in this case was given as follows.

(If(and(Step I1 0); If the signal I1 becomes on at time 0,
(Step I2 0); the signal I2 becomes on at time 0,
(Pulse I3 T1 T2); the signal I3 forms a pulse between T1 and T2
(Step I4 0); the signal I4 becomes on at time 0,
(pulse I5 T3 T4); the signal I5 forms a pulse between T3 and T4,
(Step I6 T5)); the signal I6 becomes on at time T5,
(Pulse O1 Tx Ty)); then, the signal O1 forms a pulse between Tx and Ty.

The given constraints were the least restrictive ones; all the constants were positive (the initial conditions in Fig. 9). In this case, the system generated 14 cases. Figure 9 shows one of them. This result means that if time constants in the input signals satisfy the six relations (generated hypotheses in Fig. 9) then the given specification is approved. The initial connection graph of this problem contained 235 clauses, 774 literals, and 992 links. The system performed this verification in 1 minute on a mainframe computer.

From the viewpoint of theorem-proving, the verification method explored in this paper performs efficient pruning as to selection of axioms. But, the performance of the hypothesis generat-

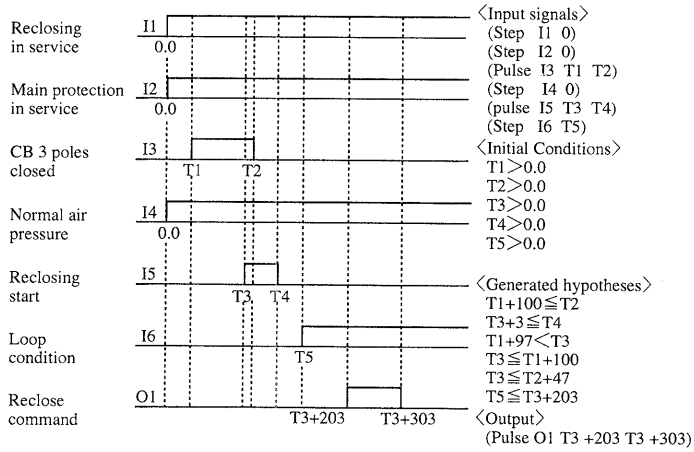


Fig. 9 Example verification result.

ing mechanism depends on the number of time variables and constraints. In the worst case in which no constraint is imposed on time variables, the hypothesis generation falls into combinatorial explosion. However, in the real design of sequential control circuits in electric power circuits, some of the input signals are only used for preconditions of circuit function and their signal values are easily specified by the designers. Furthermore, the number of components used is on the order of one hundred, and much less than those used in VLSIs. Therefore, the developed verification system is sufficiently effective for practical usage.

6. Conclusions

A design verification method for sequential control circuits was developed. The method is based on a theorem-proving technique in classical predicate logic. The control strategy for theorem-proving was a combination of hyper-resolution and the connection graph method which had a mechanism of attached procedures for handling equalities and inequalities symbolically. These approaches were improved to handle hypotheses generation and to perform theorem-proving under multiple hypotheses.

The verification method has the following advantages.

(1) It can handle time varying signals explicitly.
(2) It can verify the design specifications with timing conditions expressed symbolically.
(3) It can extract the behaviours with timing conditions from the proposed design when the specified timing conditions are insufficient.

The developed verification method was applied to the verification of an auto-reclosing circuit in an electric power substation. The circuit contained about 40 components. The verification for each specification was performed correctly in about 1 minute on a mainframe computer. This method is satisfactory and efficient for practical verification of sequential control circuits.

References

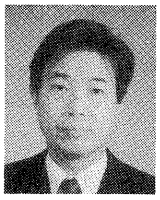
1) Chang, C. L. and Lee, R. C.: *Symbolic Logic and Mechanical Theorem Proving*, Academic Press (1973).
2) Fujita, M. et al.: Aid to Hierarchical and Structured Logic Design Using Temporal Logic and Prolog, *IEE Proc.*, Vol. 133, Pt. E, No. 5 (1986).
3) Galton, A. ed.: *Temporal Logics and Their Applications*, Academic Press (1987).
4) Auffray, Y. et al.: Strategies for Modal Resolution: Results and Problems, *Journal of Automated Reasoning*, Vol. 6, pp. 1-38 (1990).
5) McMillan, K. L.: Symbolic Model Checking—An Approach to the Explosion Problem, CMU-CS-92-131 (1992).
6) Yamada, N. et al.: A Theorem Proving System for Logic Design Verification, *J. Inf. Process.*, Vol. 11, No. 2, pp. 92-104 (1988).
7) Yamada, N. et al.: Design Verification of Sequential Control Circuits Based on Theorem-Proving Method, *J. Inf. Process.*, Vol. 14, No. 2, pp. 126-133 (1991).

(Received March 26, 1993)
(Accepted September 6, 1994)

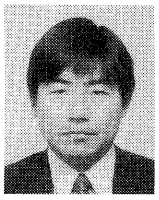
Naoyuki Yamada was born in Yamaguchi in 1953. He received the B. S., M. S. and Ph. D. degrees in nuclear engineering from Osaka University in 1976, 1978 and 1990 respectively. In 1978, he joined Hitachi, Ltd., where he is currently a senior researcher of Energy Research Laboratory. His interests includes the application of artificial intelligence to engineering design problems. He is a member of IPSJ, AAAI and IEEE/CS.

Yoshikatsu Ueda was born in Nara in 1952. He received the B. S. and M. S. degrees in control engineering from Osaka University in 1974 and 1977 respectively. In 1977, he joined Hitachi, Ltd., where he is currently a senior engineer of Systems Engineering Division. His interests includes the planning and development of Computer integrated manufacturing systems in process industry. He is a member of IPSJ.

Junko Ito was born in Tokyo in 1962. She received the B. S. degree in Physics from Japan Womans University in 1985. In 1985, she joined Hitachi, Ltd., where she is currently an engineer of Systems Engineering Division. Her interests includes the planning and development of computer integrated manufacturing systems in process industry.



Tomoharu Nakamura was born in Fukuoka in 1953. He received the B. S. and M. S. degrees in control engineering from Kyushu Institute of Technology in 1976 and 1978 respectively. In 1978, he joined Hitachi, Ltd., where he is currently a senior engineer of Kokubu Works. His interests includes the development of control and protection systems of high voltage direct current transmission. He is a member of the Institute of Electrical Engineers of Japan and IEEE.



Junichi Yoshizawa was born in Tochigi in 1960. He received the B. S. degree in electrical engineering from Niigata University in 1982. In 1982, he joined the Tokyo Electric Power Company, where he was engaged in the design and maintenance of substations and since 1985, he has worked on the development of building expert systems for power facilities at the AI Technology Department. He is a member of the Institute of Electrical Engineers of Japan.

Satoshi Matsuda was born in Tokyo in 1949. He received B. S., M. S. and the Ph. D. degrees in computer science from Waseda University in 1971, 1973, and 1976 respectively. He joined Fujitsu Limited in 1976, where he researched and developed system programs, especially on-line systems and operating systems, and artifical intelligence. Since 1987 he has been working for Computer and Communication Research Center of Tokyo Electric Power Company, as a senior research scientist. His interests are computer science, especially artificial intelligence and neural computation. He is a member of the Institute of Electronics, Information and Communication Engineers, Japanese Society for Artificial Intelligence, Japan Society for Software Science and Technology, Association for Computing Machinery, and IEEE/CS.