人　工　知　能　71－3
（1990．7．13）

知識ベース表現システムへの時間要素の付加原理

アィサム．A．ハミド．　　大須賀節雄

東京大学　先端科学技術研究センター

より高水準な言語プログラムの時間に関する仕様の定め方とその正当性の証明に関して述べる．我々は，実時間と計算時間との相違について述べた．さらに，我々は Hoareロジックをこれに実時間経過の効果の記述要素を加えて拡張し，プログラム要素の実行時間の上限と下限を与える．また，システムのタイミングに関する制約の与え方とその正当性の証明の仕方は満足のいくものであることを示した．これにより，あるシステムの仕様に時間仕様を記述する理論を構築することができ，全システムの無難(safety)なタイミングを実現する時間記述の正当性を証明することができた．

Principles of adding time to knowledge base representation system.

Issam A. Hamid　and　Setsuo Ohsuga

Research Center for Advanced Science and Technology,
University of Tokyo

A methodology for specifying and proving about time in higher level language program is described. We have given distinction between real-time and computer times. Also we give an upper bound and lower bounds on the execution times of program elements, with simple extension of Hoare logic to add the effects of real time passage. Also, we have given a mechanism to prove and assist system is satisfiable with its timing constraints. Therefore, we could reach to have theory of specifying the time for a certain system represented as specification and there after we prove the correctness, of the time assertion on these specification to meet the safety timing for the total system.

1-Introduction

Real time systems and many other computer applications must meet specifications and perform tasks that satisfy timing as well as logical criteria for correctness. In real time computing the correctness of the system depends not only on the logical result of the computation but also, on the time at which the results are produced. We use the term real-time to describe systems that must supply information within specified real-time limits. If information is supplied too early or too late, it is not useful. for systems that are not real-time information that is delivered earlier that required is acceptable, and information that comes a little later than required is still usable.

It is hard to think of a research area within artificial intelligence that does not involve reasoning about time. Medical diagnosis systems try to determine the time at which the virus infected the blood system, circuit-debugging programs must reason about the period over which the charge in the capacitor increased, robot programmers must make sure that the robot meets various deadlines when carrying out a set of tasks. Two major forces are pushing real-time systems into the next generation: their need for artificial intelligence capabilities and the rapid advance in hardware.

We present a scheme for reasoning with and about time and for specifying timing properties in concurrent programs. Specifically, we need for the computation to continually satisfy stringent timing constraints, and the need to guard against an imperfect execution environment which may violate design assumptions.

The objectives are to predict the timing behavior of high level language programs and to prove that they meet their timing constraints through the direct analysis of program statements.

What is mostly missing in the higher level language is the ability to predict the timing behavior of these programs and methods to reason about time within programs.

Two major ideas are developed.

(1) Upper and lower bounds on execution times for statements can be derived, based on given bounds for primitive statements and elements in the language and underlying system.

Schemes for obtaining bounds are presented for conventional sequential statements including loops, for timing related statements that refer to imperfect computer clocks and for synchronization and communication operations with timeouts.

(2) Extend Hoare logic to include the effects of updating real-time after each statement execution, such that to formalize the safety analysis of timing properties in real-time systems.

Haase [Ha81] assumes a concurrent programming language based on guarded commands running on a non-von Neumann machine, a deterministic execution time is given for each simple statement but execution times for conditional elements and iterations need not be defined. We unlike Haase we distinguish between computer time and real-time and analyze the timing behavior of timing related statements and communication operations. In [Ja86] there is a good formal approach for the specification and analysis of timing properties in real-time systems. However, this approach[Ja86] is in general semi-decision because Presburger arithmetic which were used in the analysis with even a single uninterpreted function is undecidable. However, their work is limited for least small problem instances, therefore, their approach is impractical for large problems because a decision procedure for Presburger arithmetic(which is used to proof the time specification) is at least double exponential.

2-Machine Architecture for Real-time systems

The architecture in real-time systems is so important to completely specify the wide need timing requirements for the system specification related with time. Our proposed machine is a large number of Processors (PE's) with private memory for each one, and are communicating through a rearrangeable type of Interconnection network called Benes IN [Be65], and with reconfigurable controlling algorithm of real-time setting steps necessary to realize arbitrary mapping [Ha89]. We have give in a previous papers[Ha87][Ha88][Ha89] a parallel algorithms for setting Benes rearrangeable IN to be reconfigurable for arbitrary mapping functions, such that its reconfigurability is realizable in a real-time. We want here, to use such IN for the realization of real-time applications. Therefore, we have several problems related to what type of specification necessarily needed to specify the timing parameters in the language, which we use to program our system sensitively effected with time. How to prove the total system is safe in the sense it can satisfy all the timing constraints given by a certain type of representation.

Taking our proposed architecture presented in previous work[Ha87][Ha88][Ha89][Ha90], we want to represent the time related applications on our architecture. Therefore we should at first analyze the and represent the time in the language and prove such representation is safe to satisfy the total system timing requirements. Here, there are many processors with private memories communicating through a shared memory through our Benes IN.

We have first introduced a theory which represents the notation of time using higher level language and prove this notation can be true in the

sense, it satisfies the total system specification. We are unlike [Ja86] give a constraint graph technique in integer programming and work substantially faster to modularly design system (when the violation of a safety assertion can be limited to a small subset of the specification. Also, in this work we have coherently, add dynamic IN for the implementation of this analysis procedure.

3-On-line Scheduling for real-time tasks

Unluckily, the scheduling problem often becomes mathematically intractable(NP-hard and therefore likely requires exponential time to solve), whenever more than two processors are involved. If all the start times are known a priori[De89] scheduling can be done at compile time. In particular, if we do not have a priori knowledge of any one of the following parameters; (1) Deadlines, (2) Computation time, or (3) the start times, then for any algorithm one might propose, one can find always a set of tasks which cannot be scheduled by the proposed algorithm but which can be scheduled by another algorithm[De89].

Theorem[De89]

If a schedule exists which needs the deadlines of a set of tasks whose start-times are the same, then the same set of tasks can be scheduled at run time even if this start times are different and not known a priori. Knowledge o f the preassigned deadlines and computation times done is enough for optimal scheduling.

Due to the above published results we have given here, a mechanism that analyzes real-time tasks such that to schedule them at the run time. The analysis depends on deriving an execution formula such that to find out their computation time and deadlines. Given the timing specification of a system and a safety assertion to be analyzed, the goal is to relate the safety assertion to the systems specification. Therefore, (1) The safety assertion is a theorem derivable from the system specification. The system is safe with respect to the behavior denoted by the safety assertion as long as an implementation satisfies the requirements specification. (2) The system assertion is unsatisfiable with respect to system specification. The system in inherently, unsafe because the requirement specification will cause the safety assertion to be violated. (3) The negation of the safety assertion is satisfiable under certain conditions. Additional constraints must be imposed on the system to ensure it is safety.

4-Logic representation for time

We can classify tasks into classes of events:
(1) External events represented as μ, (2) Start event represented as σ, (3) End event represented as ϕ, (4) Transition events represented as τ.

Therefore, for the transition events it can be represented by the following notation: $\tau(e,i)$, which represents the time of ith occurance of event e, where e is any class(i.e., $e \in \{\mu,\sigma,\phi,\tau\}$ of event i, i is integer constant/variable.

The realization of time refers to an idealization of real-time a realized by a perfect global clock, which can be denoted as (rt), like the Greenwich Mean time. Computer time is the discrete approximation to real time implemented on machines by a variety of hardware and software methods.

At the hardware level there may be fixed interval or programmable interval times that produce, tick interrupts or absolute timers that periodically update a software-accessible counter. A software clock would typically use the tick interrupts or the value of an absolute time counter to generate a computer time. Computer time (CT), $CT=RT+\delta$, $|\delta| \leq \varepsilon$ is determined by the accuracy of the hardware clock, tick interval, synchronization interval, and synchronization method.

It has been assumed implicitly that our abstract real time (RT) is represented by a real-number and that each CT is a computer approximation to a real number. CT is a more complex data structure with separate components.

Although the analysis of real time systems necessarily involves reasoning about system behavior with respect to time, we do not think that temporal logic is appropriate for this task.

Temporal logic is more concerned with the relative order in which actions are executed rather that the absolute timing of events. For instance, A;(B|| C), in temporal logic represents two execution sequences, ABC, ACB, but nothing is said about the completion of B in the second sequence. Even for the same sequence ABC, we might want to distinguish between the execution that schedules A at time=0, B at time=1, and C at time=2, from the execution that schedules A at time=0, B at time=2, and C at time =3. For temporal logic this distinction is unimportant. For example if all three actions A,B,C each takes 1 time units to execute and the composite action above must be completed by time=2, then B and C must be executed in parallel on two PEs. In this neither the sequence ABC nor ACB captures the desired behavior,.

It has been analyzed by [Be81], that real time can be modeled in temporal logic simply as another global variable, the clock and then assertions involving real time will simply be temporal logic formulas involving the clock variable. The main problem with this approach is when to increment the clock variable in relation to the other activities in the system. One possible way to achieve this is to insert

the assignment statement;
Clock:= Clock+C, at the end of every action where C is the time required to execute that action.

If all action are executed in some sequential order, such as on a single PE, then the clock variable will indeed keeps track of real time.
((We should treat each action and the clock update statement at its end to be one inseparable atomic action.))

But this is not true when two or more actions can be executed in parallel. The rate of scheduling the clock process when there is one PE will differ when two PEs are available to execute all the PEs. Proof rules which are sound under one execution environment may not apply under a different execution environment.

The validity of the assertions that we want to prove about real time systems often can not be established without knowing more details about the run-time scheduler. This is unlike the usual safety and liveness properties of no-time critical systems, which we want to hold true in spite of the scheduler as long as we are assured that some fairness criterion in scheduling is not. Real time logic reasons about occurrences of events, execution of actions are represented as occurrences of start and stop events. Time is captured by a function which assigns time values to event occurrences.

5-Timing analysis of Higher level Language

A particular execution of a statement; S can be represented by four events; as it was mentioned in Sec.4.
t(S) is the real time between these two events,
For every statement S we want to know the execution time t(S) in a given program.

However, the value t(S) depends on the context of S, the data of the program, the compiler , the run time system, the target machine and other parameters. But it is possible to obtain bounds for t(S) .
if $T(S)=[t_{min}(S), t_{max}(S)]$, then $t_{min}(S) \leq t(S) \leq t_{max}(S)$ for all execution of S in a given program.

5-1- Timed Hoare Logic

Hoare logic uses assertion P and Q, respectively before and after a statement S. {P} S {Q}, means if P is true before the execution of S and S is executed, then Q is true after S, (assuming S terminates).
With the perfect knowledge of timing we would augment the above form to;
{P} < S; RT:=RT+t(S) > {Q}, where P and Q may have relations involving real-time (RT) before and after the execution of S; the brackets (<,>) indicate that the execution of S and incrementing RT occur at the same time.
The axiom of assignment can be used in P and Q for assertions about RT.
For example, if P=P(RT,...) then {P(RT,...)} S {(RT-t(S),...)} or {P(RT+t(S),...}S{P(rt,...)}
Example:

The extended logic can express conveniently basic timing properties and constraints;

(1) **Performance Specification;**

If S starts executing in the time interval $RT_{start'}$ it will finish sometime in the interval $RT_{start}+T(S)$, therefore, $\{RT=RT_{start}\}$ S $\{RT=RT_{start'}+T(S)\}$, a simple variation is; $\{rt=t_{start}\}$ S $\{rt=t_{start}+t$, t in $T(S)\}$. $RT=t_{start}+T(S)$, where $t_{start'}$ is shortening of $[t_{start'}, t_{start}]$.

(2) **Deadlines,**

Let $RT_{dl}=[t_{dlmin}, t_{dlmax}]$ be deadlines such that a program S must be completed no earlier than t_{dlmin} and no later than t_{dlmax}.
$\{R(T)+T(S) = RT_{dl}$ S $\{RT=RT_{dl}\}$
i.e., at the start event of S, realtime must be bounded. $t_{dlmin}-t_{min}(S) \leq rt \leq t_{dlmax}-t_{max}(S)$.

(3) **Control of a Periodic Process,**

Consider a program S which is to be executed periodically starting at time rt_{start}. An approximate invariant involving the time rt before and after each execution is;
Next=rstart+n× period=rt+delta, |delta| ≤eps.
where next is the start time for each cycle (next is a program variable). n is the number of execution of S. period is the time interval allocated to a cycle, delta is the error bounded by eps of the next relative to real time.
We have given an analysis procedure to analyze the sequential program statement in term of times.

5-2 Reasoning about time

Using the event classification we can construct the timing specification for the time statements.
Example, Executing statement, S1, then program P is executed within 3 time units, During each execution of program P the result is sampled, and subsequently transmitted to the display as an output=Out. The computation time of P is 2 time units.
To represent the above example in time specification, we can have, $\forall x\ \tau(\mu S1,x) \leq \tau(\sigma P,x) \wedge \tau(\phi P,x) \leq \tau(\mu S1,x)+3$, Also, $\forall y\ \tau(\sigma P,y)+2 \leq \tau(\phi P,y)$.
We have also the following safety timing needs,
if the transmitted information is displayed(output+0ut) within 1 time unit after the completion of program P, then we can assure that within 4 time units of invoking statement S1, the requested information is displayed. Then to represent this in the timing specification we have;

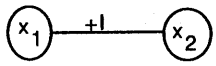
$\forall u\ \forall w\ \tau(\phi P,u) \leq \tau(\mu Out,w) \wedge \tau(\mu Out,w) \leq \tau(\phi P,u)+1 \rightarrow \tau(\mu S1,u) < \tau(\mu S1,w) \wedge \tau(\mu Out,w) \leq \tau(\mu S1,w)+4.$

Such representation for analysis purpose can be transform by replacing each ocurrance function $\tau(e,i)$ by a function $fe(i)$. However, this approach has also been studied by [Ja86], but is impractical for use in verifying the full specification when the system is large or have some sort of complexity. The time specification formulas specifically are restricted to arithmetic inequalities of the form; occurance function ±integer ≤occurance function. variable in the occurance function can be arbitrarily quantified by $\forall$, $\exists$. Moreover, inequalities can be connected using the logical connectives ($\rightarrow$,$\vee$,$\wedge$). Such subclass allows inequalities involving two occurance functions and an integer constant. In principle, this is consistent with the view that the timing constraints of a real-time system is represented as imposing an integer constant lower/upper bounds on occurance of pairs of events.

The solution to this problem can be approached by taking only subset of the specification which have a bearing on the validity of a particular system specification. Therefore, we can extract the or choose the suitable assertions from the system specification before discussing the primary decision procedure. Therefore, we can modify the inference mechanism by considering the subclasses of the total logical specification. The operation of many process control systems is often structured according to different modes. Intuitively, a mode corresponds to some assertions about the values of a set of state attribute. The assertion in a system specification are often qualified by a collection of modes, i.e., they are required to be true only under those modes. Under other modes, these assertions need not be true. It is interesting to point out that techniques to improve the efficiency of the inference mechanism enable us to prove the validity of a safety assertion independent of whether a feasible schedule(one that meets all the performance requirements of the system) exists.

We can see that the specification terms consists mainly, of arithmetic inequalities which may be quantified involving two terms and an integer constant, (term is is either a variable or a function).

For instance, $x_i - x_j \leq \pm a_{ij}$, where x_i, x_j are variables, and $\pm a$ is integer constant. This expression can be represented by directed graph, as x_i and x_j are nodes, and the edge which is connecting these nodes, represents the literals, a is the weight which represents the corresponding integer constant in literals. Therefore, we can have also, $x_i \pm a_{ij} \leq x_j$ which has the following meaning in graph representation;

Literal: $x_1 \pm I \leq x_2$ 

The nodes representing terms involving the same function symbol will be grouped together to keep track of the consistency among the different instances of the same function. Therefore, nodes can be partitioned into disjoint groups, each one has one or more nodes. All nodes of the same function symbol belong to a single cluster.(One cluster share the same function symbol)

Then we will the Algorithm-1 which can construct a graph G from a given specification formula. Please note that, a term is a constant, a variable or an n-ary functional symbol followed by n arguments all of which are terms. A literal is an atom or the negation of an atom. A clause ˆs a finite disjunction of zero or more distinct literals. All variables in a clause are (implicitly) universally quantified and their scope is just the clause in which they occur. A clause set is the implicit conjunction of a set of clauses. Statements can be represented in the clause language by first representing them (where possible) with a formula in the first order predicates calculus. The procedure first transforms formula to prenex normal form and then to conjunctive normal form and finally removes each existentially quantified variable by instead employing an appropriate function or constant, which are called Skolem function[Da60]. The remaining quantifiers are then dropped to yield the set of clauses. The typical proof found by such a program is a proof by contradiction to seek such a proof a formula in the first order predicate calculus is written, and that corresponds to assuming the purported theorem false.

A set of clauses is satisfiable if there exists Herbrand interpretation of S for which AS evaluates to true. A set S of clauses is unsatisfiable if no Herbrand interpretation exists that establishes the set to be satisfiable in other words, S evaluates to false for every Herbrand interpretation.

Herbrandıs theorem,

A set of clauses is unsatisfiable iff there exists a finite set of (herbrand) ground instances of S that is truth functionally unsatisfiable [Wo84].

Algorithm-1

For each clauses C_i, for each literal; $x_1 \pm I \leq x_2$

(1) Find the group of nodes corresponding to the term x_1.

(The group is associated with the function symbol of x_1)

If the group does not exist create an empty one.
(2) Search the cluster from step 1 for a node labeled x_1
(3) Repeat steps 1 and 2 for the term x_2.
(4) Add a directed edge (x_1,x_2), with weight $\pm I$ from node x_1 to node x_2.

The above algorithm iteratively adds all the literals in a class to the output graph one clause at a time. Therefore for each literal $x_1 \pm I \leq x_2$, the nodes x_1,x_2 are added to the graph and a directed edge(x_1,x_2) with weight $\pm I$ from node x_1 to x_2

The graph has interested property such that for a given pair of nodes x_0 and x_n, in a graph G, there is a path from x_0 to x_n if there is a sequence of edges, (x_0,x_1), $(x`_1,x_2)$, $(x`_2,x_3)$,...$(x`_{n-2},x_{n-1})$, $(x`_{n-1},x_n)$, and a substitution Δ, such that pairwise unification of x_i and $x`_i$ for all $1\leq i\leq n-1$ can be constructed , i.e., $x_i\Delta=x`_i\Delta$, where $x_i\Delta$ and $x`_i\Delta$ denote the terms after applying $\Delta$ to $x_i$ and $x`_i$, respectively.

We can notice that; each pair of x_i and $x`_i$, $1\leq i\leq n-1$ either to be the same or belongs to the same cluster. Therefore we may have a cycle in a graph G if there is a sequence of $(x_0,x_1)$, $(x`_1,x_2)$, $(x`_2,x_3)$,...$(x`_{n-2},x_{n-1})$, $(x`_{n-1},x_n)$ and a substitution Δ such that there is a path from $x_0$to $x_1$and x_0 and $x_n$can be unified with the substitution Δ. Here, x_0 and $x_n$is the same node or belong to the same cluster.

In a graph G we have the weight of a path as the sum of the weights of the corresponding edges forming the path.

For the analysis procedure we use the same suggested idea in [Bl80] to show that a safety assertion A is consistent with a system specification B. It must be proved that the formula $C=B\rightarrow A$ is valid. This is equivalent[Bl80] to show that $\neg C$ is unsatisfiable. [Bl80] proved this depending on a modified resolution procedure to show that $\neg C$ is unsatisfiable. We have used this result in our analysis of real-time specification and the invoked time assertion, by design a technique and a mechanism to prove the unsatisfiability of $\neg C$.

Theorem 1

Let G be a graph constructed from a given formula F`` using Algorithm.1. If a cycle with positive weight exists in G, the formula P consisting of the conjunction of literals (inequalities) corresponding to the edges involved in the cycle is unsatisfiable.

Proof:

The following sequence of edges represent the positive cycle (x_0,x_1), $(x`_1,x_2)$, $(x`_2,x_3)$, ... $(x`_{n-2},x_{n-1})$, $(x`_{n-1},x_n)$.

From the definition of a cycle there is a substitution Ψ, such that;

$x_i\Psi=x`_i\Psi$ for all $1\leq i\leq n-1$ and $x_0\Psi=x_n\Psi$,

since each edge correspond to an inequality L_i, one can construct an instance of the conjunction of L_i`s by applying Ψ to each L_i.

$$x_0\Psi+I \leq x_i\Psi \wedge x`_1\Psi+I_1 \leq x_2\Psi \wedge x`_2\Psi+I_2 \leq x_3\Psi \wedge \ldots\ldots \wedge x`_{n-1}\Psi+I_{n-1} \leq x_n\Psi$$

where the I`s denote the integer constants in the inequalities. If the set of inequalities are summed, $x_0,x_1,x`_1,x_2\ldots\ldots x_n$ cancel. The resulting inequality asserts that; $I_0+I_1+I_2+\ldots\ldots+I_{n-1}\leq 0$.

This is clearly unsatisfiable because the cycle has a positive weight, i.e., sum of the integer constants in the inequalities must be a positive integer. We proved that an instance of the conjunction of the inequalities corresponding to the positive cycle is unsatisfiable, thus we conclude the original set of inequalities is unsatisfiable. (end of proof)

Since the literals corresponding to a positive cycle may belong to nonunit disjunctive clauses, Th-1 does not guarantee that the detection of a positive cycle in G means the unsatisfiability of the analysis. The algorithm to be described for detecting positive cycles depends on an operation for removing nodes within a cluster such that the positive cycles in the original graph are preserved.Repeating this operation will eventually remove all nodes and subsequently all the clusters, so that positive cycles can be detected.Each iteration removes a node and its incident edges, and adds the appropriate edges (and some time nodes to other clusters) to preserve the cycles that may have existed in the graph. A self-loop with positive weight denotes a positive cycle in the original graph.

Algorithm-2

(Detecting Positive Cycles);
For each cluster in the graph,
For each node in the cluster.
(1) $S=\{x_1,x_2,\ldots\ldots x_n\}$, be the cluster to which x belongs; (all the x_i`s have the same function symbol), S also, includes x itself for each x_i in S, if x and x_i can be unified ,
(a) Let Ψ be the most general unifiers of x and x_i.
(b) For each pair of nonself-loop edges (x`,x) and $(x_i,x``)$ with weights I_1, I_2 add the nodes x` Ψ and

x``Ψ to the respective clusters if not already there, and add the edge (x`Ψ,x``Ψ0 with weight (I_1+ I_2) where (x` Ψ) denote the label for a node after applying Ψ to x`.

x``Ψ denote the label for a node after applying Ψ to x``.

(c) Repeat step b, for each pair of nonself-loop edges (x`,x_i), and (x,x``).

(2) Remove node x and all edges incident on it from the graph.

(3) While generating edges in step (1) and (2), a self-loop with a positive weight signifies a positive cycle in the graph.

(An edge from a node to another node in the same cluster is not a self-loop).

((A self-loop with a non-positive weight signifies a non-positive cycle in the graph))

In the implementation of this algorithm after an edge is added to the graph, a note is made of the two edges combined to generate the new edge.

This information is important in identifying the edges from the initial graph involved in a cycle after detection.

Certain negative cycle detected in step-3 can be useful to detect the unsatisfiabilty.

Theorem 2;

The node removal operation in Algorithm-2 preserves the cycle of the original graph.

(Appendix-B)

Theorem-3

The node removal operation described in Algorithm 2 does not introduce any new cycle which does not exist in the original graph.

How Algorithm-2 terminates;

Two types of functions appear in the clausal form of our formula;

Occurance functions corresponding to particular events, and Skolem function appearing as arguments to the occurance functions.

Algorithm-2 may not terminate if a function is allowed to take itself as an argument.

We add the condition to ensure that a function symbol does not appear more than once in a term corresponding to a node in our graph.

Therefore step 1 of Algorithm 2 can be written to include this condition.

Let S={x_1,x_2,.....x_n} be the cluster to which x belong;

For each x_i in S if x and x_i can be unified ;

(a) Let Ψ be the most general unifier of x and x_i.

(b) For each pair of non-self-loop edges (x`,x) and (x_i,x``) with weights I_1, and I_2.

If a function symbol does not appear more than once in x`Ψ and x``Ψ add the nodes x`Ψ and x``Ψ to the respective clusters if not already there, and add the edge (x`Ψ, x``Ψ) with weight (I_1+I_2).

(c) Repeat step b for each pair of nonself-loop edge (x`,x_i) and (x,x``).

The added condition in step (1b) that a function symbol may not appear more than once in x`Ψ and x``Ψ ensures the termination of Algorthim-2.

(The same condition is also imposed on step 1c).

Since an occurance function does not take an instance of itself as an argument, the condition in step(1b) does not impose any restriction on the occurance functions.

However, the condition prevents a Skolem function from taking an instance of itself as an argument. For example a node labeled (f(h(h(x))) is not allowed in our constraint graph where f denotes an occurance function and h is a Skolem function. Skoelm function replaces an existentially quantified variable. The clausal form of the initial set of formulas under investigation does not have any term in which a Skolem function appears more than once. Hence the condition in step (1b) does not restrict the subclass of formula described in this paper.

However, the node removal operation may in some cases produce nodes in which a Skolem function symbol appears more than once.

The condition in step (1b) prevents the generation of such terms. Although the completeness can not be claimed for Algorithm-2 under this condition we impose this condition in the implementation of this algorithm since it seems to be a good compromise in practice. It is straightforward to show that Algorithm-2 terminates.

In each iteration nodes in a cluster are removed one by one unit the cluster is empty.

However, as nodes in a cluster are removed new nodes may be added either to the same cluster or to the remaining clusters (not yet processed). The number of nodes which are added to the same cluster cannot grow without a bound because a Skolem function symbol is not allowed to appear more than once in any node, which is added to remaining cluster is eventually removed when the respective cluster is being processed.

Furthermore, after all node in a cluster are removed the corresponding occurance function symbol does not appear in any mode.

Hence, all clusters are eventually removed thus ensuring termination.

6- Verification Procedure

Here we want to presents an algorithm for determining the unsatisfiability of F`` as positive cycles are detected in the corresponding graph G. It is

straightforward to show that if all edges involved in a positive cycle in G correspond to literals (inequalities) which belong to unit clauses, F`` must be unsatisfiable. The case where one edge corresponds to a literal (inequality) which belongs to a nonunit disjunctive clause C_i is also involved in a difficult positive cycle. The problem becomes increasingly computationally intensive when more edges in a positive cycle belong to nonunit disjunctive clauses.

However, this problem is hard as the CNF satisfiability problem of propositional logic where each disjunctive clause contains either all negated literals or all unnegated (positive) literals.

The set of all clauses in a formula F`` can be viewed as a collection (conjunction) of m clauses, $C_1, C_2, C_3, \ldots C_m$, where each C_k, is a disjunctive clause. Note that this is merely a notational convention and we are not requiring literals in different clauses be distinct. Let the following notation denote the list of inequalities corresponding to the edges in the ith positive cycle detected, $C_{i,1}, C_{i,2}, C_{i,3}, \ldots\ldots C_{i,mi}$, where $C_{i,j}$, represents the jth literal (edge) in the ith positive cycle detected and each $X_{i,j}$,is a literal in at least one of the C_k`s.

The following theorem is a variation of Herbrand`s .

Theorem-4

A set of clauses is unsatisifiable if and only if there is a finite unsatisfiable set of ground instances of S and $\neg$ Pi for $1 \leqq i \leqq n$ where each Pi is the conjunction of inequalities corresponding to the edges in a positive cycle detected in the constraint graph for S.

The above formulation permits one to use any method in propositional logic to check for unsatisfiability as positive cycles are detected and the appropriate clauses are added to the existing set of clauses.

How to test the unsatisfiability for formulas in which each clauses contains either only negated or only unnegated literals. This algorithm is more suited for safety analysis of RT systems because;

(1) It is desirable to have a dynamic algorithm in the sense that as each new cycle is detected , the corresponding clause is added to the set of existing ones, and is then checked for unsatisfiability. If it is shown to be unsatisfiable we can stop at once. Otherwise we need to continue the node removal operation until another positive cycle is found. The well-known procedures require redoing the computation each time a new clause is added, i.e., a positive cycle is detected. In contrast, the algorithm to be described builds on top of the computation already done to check for unsatisfiability as each new positive cycle is detected.

(2) Even though the algorithm to be described has an exponential time complexity in the worst case as one may expect, it is desirable to have an algorithm whose running time in the worst case is exponential with respect to the number of positive cycles detected rather than being exponential with respect to the number of literals or the total number of clauses, the rationale being that we expect to have only a few positive cycle detected in most cases.

(3) While checking for satisfiability after detecting a positive cycle if the formula is still satisfiable, the proposed algorithm may terminate very quickly after generating only a small part of the search tree. In constr act the Davis method[Da60] must still complete removing almost all of the literals before concluding satisfiability.

The running time for the algorithm to check the unsatisfiability is proportional to the number of conjunctions in the disjunctive normal form. A new level may be added to the search tree each time a positive cycle is detected. Therefore, in the worst case the height of the search tree is the number of positive cycles detected. This algorithm has exponential time complexity.

We are also interesting to find a parallel version of these algorithms to run them in parallel on interconnection network like that in[Ha89]. We can take use of such parallelism to revise the notation of verficatication procedure of real time notation schemes for distributed systems.

6- Conclusion

The methodology for specifying predicating and proving assertions about time is new and promising but still incomplete and untested in practice. In order to compute bounds on structured statements in general and to compute times for statements in particular programs, it would be desirable to build some automatic tools to help in the analysis.

The main results and contributions of this work are the techniques that in principle permit the prediction of the timing as well as the logical behavior of programs. Therefore, an approach is presented for safety analysis of timing properties of real-time system expressible in a subclass of logic. To show the safety assertion is consistent with a given specification we need to prove that the corresponding formula consisting of the specification in conjunction with the negation of the safety assertion is unsatisfiable. The analysis procedure is based on three algorithmic steps:

(1) Construct a graph representing the specification and the negation of the safety assertion.

(2) Detects positive cycles in the graph using a node removal operation.

(3) Decides unsatisfiability based on the positive cycles detected.

References

[Be81] Bernstein, A., et.al., 'Proving real-time properties of programs with temporal-logic,' in Proc. 8th Symp., Opea. Syst. Priciples, ACM, SIGOPS, 1981, pp.1-11.

[Bl80] Bledsoe, W., et.al., 'Variable elemination and chaining in a resolution based prover for inequalities,' in 15th Conf. Automated Deduction, Lecture Notes in Computer Science, Springer Verlag, pp.70-87, 1980.

[Da6] Davis, M. et.al., 'A computing procedure for quantification theory,' JACM, 7, 201-215, 1960.

[Ha90] Hamid, I.A., et.al., 'A general solution for the optimal mapping problem on bounded degree graph,' Workshop on Computer Architecture, IPSJ, 81-1, March, 23, 1990, pp.1-8.

[Ha89] Hamid, I.A., et.al., 'A new fast parallel computation model for setting Benes interconnection Network,' Trans. IEICE, vol.72, April, 1989.

[Ha86] Halpern, J.Y., et.al., 'A propositional modal logic of time intrvals,' Logic in computer Science, Springer-Verlag, Newyork, June 1986.

[Le87] Lee, I., et.al., 'Adding time to synchronous process communications,' IEEE, Trans. Comput., c-36, No.8, August, 1987, pp.941-948.

[Li90] Liu, L., et.al., 'Static analysis of real-time distribute systems,' IEEE ,Trans. Soft.Eng., vol.16., No.4., April 1990, pp.373-388.

[Ja86] Jahanian, F., et.al., 'Saftey analysis of timing properties in real-time systems,' IEEE Trans. Soft.Eng., vol.SE-12, September, 1986, pp.890-904.

[Sh89] Shaw,A., 'Reasoning about time in higher level language software,' IEEE, Trans. Soft.Eng., vol.15, July, 1989, pp.875-889.

[Ts90] Tsai, J.,et.al., 'A noninvasive architecture to moniter real-time distributed systems,' IEEE, Comput, Magazine, March, 1990, pp.11-23.

[Wo84] Wos, L. et.al., 'Automate Reasoning: Introduction and applications's, Prentice-Hall, Englewod, Cliffs, NJ. 1984.

Acknowledgement

The first author would like to give his thanks to Prof. Gregor Bochmann of the University of Montreal, for opening the opportunity to discover such wonderful research problems discussed in this paper. We would like also, to give our appreciations to the visiting professors of RCAST, (endowed chairs of CSK) and Profs. Hori, and Yamaguchi for the encouraging helpful discussion.