Iterative Refinement and Condensation for State-graph Construction*

Farn Wang and Pao-Ann Hsiung
Institute of Information Science, Academia Sinica, Taipei, Taiwan 115, Republic of China
+886-2-27883799 ext. 1717; FAX +886-2-27824814; farn@iis.sinica.edu.tw

Abstract

The traditional technique to generate a global state-graph representation for a concurrent system is to calculate the product of state-graph representations of the local processes. We develop new techniques, and prove their correctness, to help condensing intermediate state-graphs in the iterative product-calculation. Experiments with Fischer's timed mutual exclusion protocol have been carried out to justify the approach.

1 Introduction

A concurrent system is usually described as a set of local processes [4, 8, 10, 15, 16]. To generate a representation for the global state space, either explicitly or implicitly, for model-checking is difficult because global state-graph construction can easily grow out of memory space, even when the local process representations are simple[1, 4, 14]. We propose a framework to construct global state-graphs by iteratively composing local process representations. Two new techniques are then presented and proven for the condensation of the intermediate state-graph representations in the iterations.

Here we first give a brief description of our idea. A *state-graph* is a representation for system behavior with *node*s and *arc*s. Each node represents, and is called, a *state subspace* which is a subset of the global state-space. Each arc is an arrow going from a source node to a destination node and represents *transition*s from a state in the source state subspace to a state in the destination state subspace. Suppose we have a concurrent system S represented by m local processes with state-graphs $G_1, \ldots, G_m$ respectively. The traditional technique of generating a global state-graph for S is Cartesian product calculation, i.e., calculating $G_1 \times G_2 \times \ldots \times G_m$. Such a product calculation can be done in an iterative way described in table 1(a). Our idea is to realize the framework shown in table 1(b), which allows us to condense G in each iteration with a set of provenly correct condensation operators. **merge**() is a binary state-graph product calculator and produces a state graph which is usually bigger than its two arguments and thus represents refinement of state-graphs. $\mathbf{condense}_1(), \ldots, \mathbf{condense}_n()$ are state-graph condensation operators which try to make their argument state-graphs smaller. In the following, we describe our major techniques used to condense state graphs. The first two are our originals.

- **Stability of variable values.** In a concurrent system, we can usually deduce the ranges of variable contents in a state subspace by analysis on read-write event ordering. For example, suppose we have a state subspace named q in which $y = 0$ (y is a register-like variable) is invariably true. Suppose from q, we have two outgoing transitions e_1, e_2 with triggering conditions $y = 1$ and $y \neq 1$ respectively. Now if we know in all the state-graphs of the local processes, nobody else will write to y with value 1, then we can eliminate e_1 from q because we know its triggering condition will never be true as long as the system keeps staying in q. We shall give an example in subsection 4.1 to illustrate how the idea works in more details.

*The work is partially supported by NSC, Taiwan, ROC under grant NSC 87-2213-E-001-007.

(a)	(b)

product$(G_1,\ldots,G_m)$ {
(1) $G := G_1$;
(2) for $i := 2$ to m, do {
(1) let $G := G \times G_i$;
}
(3) return G;
}
	product$(G_1,\ldots,G_m)$ {
	(1) $G := G_1$;
	(2) for $i := 2$ to m, do {
	(1) let $G :=$ **merge**(G, G_i);
	(2) let $G :=$ **condense**$_1(\ldots($**condense**$_n(G))\ldots)$;
	}
	(3) return G;
	}

Table 1: Global state-graph calculation through Cartesian product calculation

- **Shielded local timers.** We shall prove lemma 2 which says from a state, if a local (used only by one process) timer has no effect in triggering conditions of triggerible transitions, invariance conditions, and specification before any reset statement, then the timer's current reading is of no effect to the system behaviors. This observation can be used to broaden the state subspace equivalence relation and thus condense state-graphs. We shall give example in subsection 4.2 to illustrate how the idea works in more details.
- **Zones.** Timer readings in a state of dense-time system are symbolically recorded in a Difference-Bound Matrix[2, 9].
- **Symmetry under index permutation.** The technique is adapted from [10] to deal with timers[2, 9]. For the verification of a set of identical processes, such a technique is indispensable.

After we have defined our process description language and specification language formally in sections 2 and 3 respectively, we shall explain the techniques with examples and prove their correctness in section 4.

We implement the ideas and report in section 6 our experiment on Fischer's timed mutual exclusion protocol. Compared to the technology of pure symbolic manipulation [2, 3, 4, 5, 11, 12], one major advantage of our approach is the possibility for incorporation of condensation techniques which utilize information derivable from those intermediate state-graphs generated in the iterations. To utilize such information after the final global state-graph is generated will be too late to avoid state-graph size explosion.

Section 2 describes the syntax and semantics of our interested real-time concurrent systems. Section 3 describes our specification language TCTL[1]. Section 4 contains lemmas 1 and 2 and explains with examples our idea of value stability and timer shieldedness. Section 5 discusses implementation details. Section 6 describes our experiments. Section 7 is the conclusion.

We shall adopt the following notations. Given a set or sequence F, $|F|$ is the number of elements in F. For each element e in F, we also write $e \in F$. $\mathcal{N}$ is the set of nonnegative integers, $\mathcal{Z}$ is the set of integers, and $\mathcal{R}^+$ is the set of nonnegative reals.

2 Timed mode transition systems

A real-time concurrent system is composed of many *process*es. Each process runs autonomously and interacts with others through read-write operations to *global variables* and *timers*. In addition, each process has its own *local variables* and *timers* which no other processes can access. For a system with m processes, we shall use integer $1, \ldots, m$ to identify the m processes.

Given a timer set H and a variable set F, a *state predicate* η of H and F is a formulus constructed according to the following syntax.

$$\eta \ ::= \ y = c \mid y = p \mid x + c \sim x' + d \mid x \sim c \mid \neg\eta \mid \eta \vee \eta'$$

y is a variable in F. c, d are natural numbers. p is a process identifier. x, x' are timers in H. $\sim$ is an inequality operator in $\{\leq, <, =, >, \geq\}$. Common shorthands like truth, falsehood, conjunction, and implication can be defined. Notationally, we let B_F^H be the set of all state predicates of H and F.

Definition 1 *:* **PTMTS**

Process timed mode-transition system (PTMTS) is defined to describe behaviors in an atomic process in a *real-time concurrent system*. Notationally, we use subscript p to denote those transitions, local variables, and timers of process p. A PTMTS for process p is a tuple $A_p = \langle X, X_p, Y, Y_p, I_p, T_p \rangle$ with the following restrictions.

- X is the global timer set, X_p is the local timer set, Y is the global variable set, and Y_p is the local variable set.
- $q_p \in Y_p$ is a special local variable which records the current *mode* of process p.
- I_p is a state predicate in $B_{Y \cup Y_p}^{X \cup X_p}$ denoting the *invariance condition* of process p. (Thus the global invariance condition is $\bigwedge_{1 \leq p \leq m} I_p$.)
- T_p is the set of transition rules with the following form.

$$(q_p = c \wedge \eta) \rightarrow [q_p := d; \kappa]$$

Here c and d are natural numbers. η is a state predicate in $B_{Y \cup (Y_p - \{q_p\})}^{X \cup X_p}$ denoting the *transition triggering condition*. κ is a finite sequence of *assignment statements* which is executed on the happening of the transition. Each assignment statement in κ has the following syntax.

$$y := c; \mid y := p; \mid x := 0;$$

Again $y \in Y \cup (Y_p - \{q_p\})$, $x \in X \cup X_p$, p is a process identifier, and c is a natural number.

Initially all timers and variables contain zeros. The processes act by performing transitions in an interleaving fashion, i.e. at any moment, at most one transition can happen. Right before a transition $e = (q_p = i \wedge \eta) \rightarrow [q_p := j; \kappa] \in T_p$ happens, A_p is in mode i and η is satisfied. On the happening of e, which is instantaneous, variables are assigned new values and timers are reset to zeros according to κ, and then A_p enters mode j. In between the happenings of transitions, all variable contents stay unchanged and all timer readings increment at a uniform rate. ||

Example 1 *:* For each process of Fischer's timed mutual exclusion protocol, we have a PTMTS $\langle X, X_p, Y, Y_p, I_p, T_p \rangle$ with $X = \emptyset$, $X_p = \{x_p\}$, $Y = \{l\}$, $Y_p = \{q_p\}$, $I_p = q_p = 0 \vee (q_p = 1 \wedge 0 \leq x_p \wedge x_p \leq 1) \vee q_p = 2 \vee q_p = 3$, and

$$T_p = \left\{ \begin{array}{l} (q_p = 0 \wedge l = 0) \rightarrow [q_p := 1; x_p := 0;], \\ (q_p = 1 \wedge x_p < 1) \rightarrow [q_p := 2; l := p; x_p := 0;], \\ (q_p = 2 \wedge l \neq p) \rightarrow [q_p := 0;], \\ (q_p = 2 \wedge x_p = 1 \wedge l = p) \rightarrow [q_p := 3;], \\ (q_p = 3) \rightarrow [q_p := 0; l := 0;] \end{array} \right\}$$

In Figure 1, we draw the PTMTS as a timed automaton which is more visually readable. The circles are modes and the starting mode is doubly circled. Inside the circles, we put down the mode names and invariance conditions enforced by I_p in the modes. On each transition, we put down the triggering condition (η), if any, above the assignment statements (κ), if any. For example, in mode $q_p = 1$, $0 \leq x_p \leq 1$ must be true for process p. In mode $q_p = 1$, when $x_p < 1$, process p may assign p to variable l, reset x_p to zero, and enter mode $q_p = 2$. We also label each transition with boldface number near their sources for later use. ||

Given a PTMTS $A_p = \langle X, X_p, Y, Y_p, I_p, T_p \rangle$ and a variable y in $Y \cup Y_p$, we let $D_{A_p:y}$ be the union of $\{0\}$ and the set of values assigned to y in T_p. That is, $D_{A_p:y}$ is the domain of y.

Definition 2 *:* **States** A real-time concurrent system, in our definition, is represented as a set of PTMTS's. Suppose we are given a real-time concurrent system S with m PTMTS's $A_1, \ldots, A_m$ such that for all $1 \leq i \leq m$, $A_p = \langle X, X_p, Y, Y_p, I_p, T_p \rangle$. A *state* of S is a mapping ν from $X \cup \bigcup_{1 \leq p \leq m} X_p \cup Y \cup \bigcup_{1 \leq p \leq m} Y_p$ such that for each $1 \leq p \leq m$ and $x \in X \cup \bigcup_{1 \leq p \leq m} X_p$, $\nu(x) \in \mathcal{R}^+$; for each $y \in Y$, $\nu(y) \in \bigcup_{1 \leq p \leq m} D_{A_p:y}$; and for $1 \leq p \leq m$ and $y \in Y_p$, $\nu(y) \in D_{A_p:y}$. A state ν is an *initial state* iff for all $z \in X \cup \bigcup_{1 \leq p \leq m} X_p \cup Y \cup \bigcup_{1 \leq p \leq m} Y_p$, $\nu(z) = 0$. ||

Given a state and a state predicate η, we can define $\nu \models \eta$ (ν *satisfies* η) in a traditional inductive way.

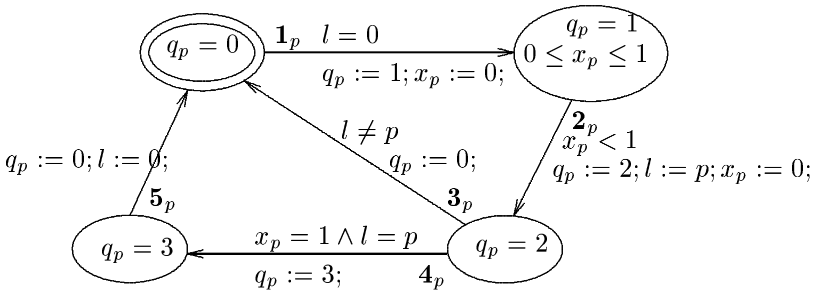

Figure 1: Fischer's timed mutual exclusion protocol

- $\nu \models y = c$ iff $\nu(y) = c$
- $\nu \models y = p$ iff $\nu(y) = p$
- $\nu \models x + c \sim x' + d$ iff $\nu(x) + c \sim \nu(x') + d$
- $\nu \models x \sim c$ iff $\nu(x) \sim c$
- $\nu \models \neg\eta$ iff it is not the case that $\nu \models \eta$
- $\nu \models \eta \vee \eta'$ iff $\nu \models \eta$ or $\nu \models \eta'$

Given a state ν and $\delta \in \mathcal{R}^+$, we let $\nu + \delta$ be a mapping identical to ν except that for each $x \in X \cup \bigcup_{1 \leq p \leq m} X_p$, $(\nu + \delta)(x) = \nu(x) + \delta$. Given a sequence of assignment statements κ, we let $\nu\kappa$ be a new mapping identical to ν except that variables are assigned new values and timers are reset to zero according to κ.

Definition 3 *:* **runs** Suppose we are given a real-time concurrent system $A_1, \ldots, A_m$ such that $A_p = \langle X, X_p, Y, Y_p, I_p, T_p \rangle$ for all $1 \leq p \leq m$. A *ν-run* is an infinite sequence of state-time pair $(\nu_0, t_0)(\nu_1, t_1) \ldots (\nu_k, t_k) \ldots\ldots$ such that $\nu = \nu_0$, $t_0 t_1 \ldots t_k \ldots\ldots$ is a monotonically increasing real-number (time) divergent sequence, and for all $k \geq 0$,

- for all $t \in [0, t_{k+1} - t_k]$, $\nu_k + t \models \bigwedge_{1 \leq p \leq m} I_p$; and
- either $\nu_k + (t_{k+1} - t_k) = \nu_{k+1}$; or there are $p \in \{1, \ldots, m\}$ and $(q_p = \nu_k(q_p) \wedge \eta) \rightarrow [q_p := \nu_{k+1}(q_p); \kappa] \in T_p$ such that $\nu_k + (t_{k+1} - t_k) \models \eta$ and $(\nu_k + (t_{k+1} - t_k))\kappa = \nu_{k+1}$.

If a ν-run describes the behavior of the system from the beginning of computation, then we require ν to be an initial state. ||

3 TCTL and model-checking

Our verification framework is model-checking. That is, the system description is given in PTMTS's and the specification is given in TCTL formulae[1, 11], and a verification problem instance asks if a given concurrent system with PTMTS processes satisfies a given TCTL formulus. A TCTL formulus has the following syntax.

$$\phi ::= \eta \mid \exists\Box\phi' \mid \exists\phi'\mathcal{U}\phi'' \mid \neg\phi' \mid \phi' \vee \phi''$$

Here η is a state predicate in $B^{X \cup \bigcup_{1 \leq p \leq m} X_p}_{Y \cup \bigcup_{1 \leq p \leq m} Y_p}$. $\exists\Box\phi'$ means there exists a computation, from the current state, along which ϕ' is always true. $\exists\phi'\mathcal{U}\phi''$ means there exists a computation, from the current state, along which ϕ' is true until ϕ'' becomes true. Traditional shorthands like $\exists\Diamond$, $\forall\Box$, $\forall\Diamond$, $\forall\mathcal{U}$, $\wedge$, and $\rightarrow$, can all be defined.

The satisfaction of a TCTL formula ϕ by a state ν in a real-time concurrent system S, written $S, \nu \models \phi$, can be defined in a standard way.

- $S, \nu \models \eta$ iff ν satisfies η as a state predicate.
- $S, \nu \models \exists\Box\phi$ iff there is a ν-run $(\nu_0, t_0)(\nu_1, t_1) \ldots (\nu_k, t_k) \ldots\ldots$ such that for all $k \geq 0$ and $t \in [0, t_{k+1} - t_k]$, $S, \nu_k + t \models \phi$.

- $S, \nu \models \exists \phi \mathcal{U} \phi'$ iff there is a ν-run $(\nu_0, t_0)(\nu_1, t_1) \ldots (\nu_k, t_k) \ldots\ldots$, a $k \geq 0$, and a $t \in [0, t_{k+1} - t_k]$ such that
  - $S, \nu_k + t \models \phi'$; and
  - for all $0 \leq h \leq k$ and $t' \in [0, t_{h+1} - t_h]$, if $t_h + t' < t_k + t$, then $S, \nu_h + t' \models \phi$.
- $S, \nu \models \neg \phi$ iff it is not the case that $S, \nu \models \phi$
- $S, \nu \models \phi \vee \phi'$ iff $S, \nu \models \phi$ or $S, \nu \models \phi'$

Also we write $S \models \phi$ (S satisfies ϕ) iff for the initial state ν of S, $S, \nu \models \phi$.

Example 2 *:* The mutual exclusion specification of Fischer's protocol in figure 1 is $\forall \Box \neg \bigwedge_{1 \leq p < p' \leq m} (q_p = 3 \wedge q_{p'} = 3)$. ||

4 How do the condensations work ?

In this section, we discuss the techniques we used to condense state-graphs. Subsections 4.1 and 4.2 discuss our newly developed techniques in details with lemmas and examples. Specifically, lemma 1 in subsection 4.1 derives value-stability properties of state subspaces of those intermediate state-graphs and can be used to eliminate state subspaces and transitions which can never happen. Lemma 2 in subsection 4.2 derives a path-based property for timer-elimination and can be used to deduce the behavior-equivalence among states.

Subsection 4.3 discusses briefly how we incorporate the techniques of symbolic manipulation. Subsection 4.4 discusses how we adapt the technique of verification by symmetry to dense-time systems.

4.1 Variable value stability under concurrent read/write

Suppose we have an intermediate state-graph composed from state-graphs for processes with identifiers in set H. For a given global variable y, we let $D_{H:y}$ be the set of values written to y by processes with identifiers in H but NOT by processes without identifiers in H. We present the following lemma, with proof sketch, which governs the stability of global variable values, derivable from intermediate state-graphs, in concurrent read/write systems.

LEMMA 1 *Suppose we are given a variable y and a finite run segment $(\nu_h, t_h)(\nu_{h+1}, t_{h+1}) \ldots (\nu_k, t_k)$ such that for all $h \leq i < k$, ν_i goes to ν_{i+1} without making assignment to y on a transition from a process with identifier in H.*

- *If we enter state ν_h with an assignment $y := a$;, then for all $h \leq i < k, t_i \leq t \leq t_{i+1}$, $\nu_i + t \models \bigwedge_{b \in (D_{H:y} - \{a\})} y \neq b$*
- *If we enter state ν_h without an assignment to y but with a triggering condition $y = a$, then for all $h \leq i < k, t_i \leq t \leq t_{i+1}$, $\nu_i + t \models \bigwedge_{b \in (D_{H:y} - \{a\})} y \neq b$*
- *If we enter state ν_h without an assignment to y but with a triggering condition $y \neq a$ with $a \in D_{H:y}$, then for all $h \leq i < k, t_i \leq t \leq t_{i+1}$, $\nu_i + t \models y \neq a$*

Proof : If the values in $D_{H:y}$ are not going to be written to y by other processes with identifiers not in H, since we have the full knowledge that processes with identifiers in H will neither write values in $D_{H:y}$ to y along the segment, thus the lemma must hold. ||

Example 3 *:* In figure 2, the square boxes represent state subspaces described by the state predicate inside while the arcs represent transitions with transition indices and process identifiers labeled by their sides. The crossed-out arcs represent transitions detected as untriggerible by lemma 1. For example, in state subspace 2, we can deduce $l \neq 1$ and thus conclude transition $\mathbf{4}_1$ will never be triggered from state subspace 2. Thus lemma 1 can be used to early delete arcs and nodes which eventually is unreachable in the final state-graphs. ||

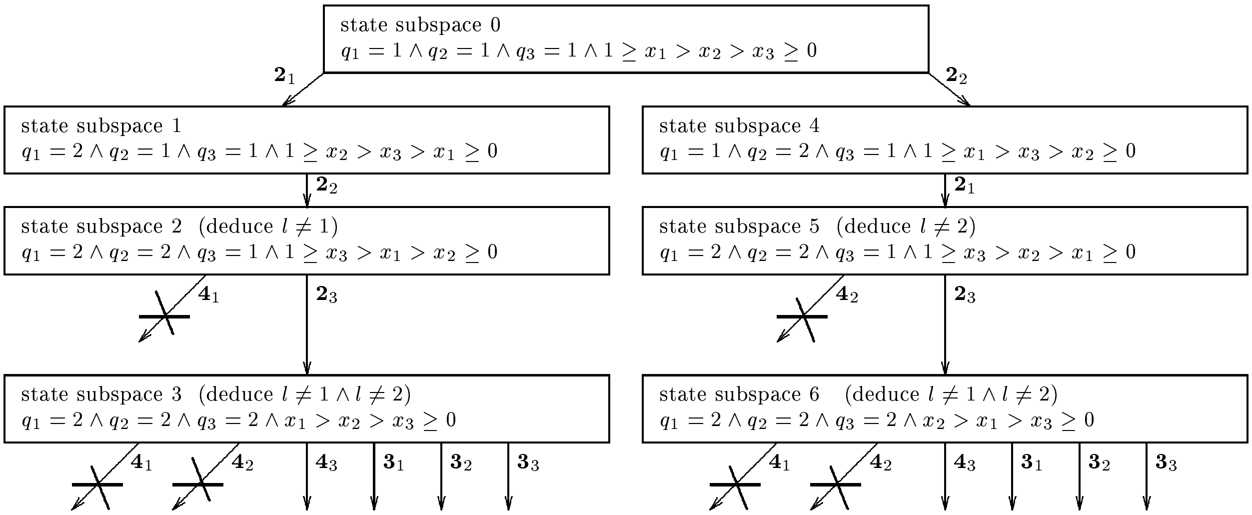

Figure 2: Illustration of lemma 1

4.2 Irrelevance of shielded timers

Here we present the following concept to condense timer recordings based on path conditions in state-graphs. An *atom* is a state predicate without negation ($\neg$) and disjunction ($\wedge$). A *timing atom* is an atom in which one or two timers are used. A timing atom in one of the following forms: $x \sim c$, $x + c \sim x' + d$, and $x' + c \sim x + d$ is called an x's timing atom. A *literal* is either an atom, called *positive literal*, or an atom preceded by a logic negation ($\neg$), called *negative literal*.

A set Γ of timing atoms is shielded in a state ν with respect to a state predicate η iff the truth values of the timing atoms in Γ together do not affect the truth value of η. Formally speaking, we define $\Omega(\Gamma, \nu, \eta)$ inductively as follows.

- $\Omega(\Gamma, \nu, \chi) = \{\mathit{true}, \mathit{false}\}$ if $\chi \in \Gamma$.
- $\Omega(\Gamma, \nu, \chi) = \{\nu \models \chi\}$ if χ is atomic and $\chi \notin \Gamma$.
- $\Omega(\Gamma, \nu, \neg\eta) = \{\neg b \mid b \in \Omega(\Gamma, \nu, \eta)\}$.
- $\Omega(\Gamma, \nu, \eta \vee \eta') = \{b \vee b' \mid b \in \Omega(\Gamma, \nu, \eta); b' \in \Omega(\Gamma, \nu, \eta')\}$.

Here we assume that $\neg\mathit{true} \equiv \mathit{false}$, $\neg\mathit{false} \equiv \mathit{true}$, $\mathit{true} \vee \mathit{true} \equiv \mathit{true}$, $\mathit{true} \vee \mathit{false} \equiv \mathit{true}$, $\mathit{false} \vee \mathit{false} \equiv \mathit{false}$, and $\mathit{false} \vee \mathit{true} \equiv \mathit{true}$. Then Γ is shielded in ν w.r.t. η iff $|\Omega(\Gamma, \nu, \eta)| = 1$. For instance, in example 1, timing atom set $\{0 \leq x_p, x_p \leq 1\}$ is shielded in any state satisfies $q_p = 0$ w.r.t. $I_p = q_p = 0 \vee (q_p = 1 \wedge 0 \leq x_p \wedge x_p \leq 1) \vee q_p = 2 \vee q_p = 3$.

Definition 4 *:* **Timer shieldedness** Suppose we are given a model-checking problem instance with system S and TCTL formulus ϕ. Let U_x be the set of all timing atoms of x. A timer x is *shielded* in a state ν for the problem instance iff

(1) x is not used in the specification ϕ; and

(2) for any $k \in \mathcal{N}$ and ν-run $(\nu_0, t_0)(\nu_1, t_1) \ldots (\nu_k, t_k) \ldots\ldots$, if either

  (a) $k > 0$ and ν_{k-1} goes to ν_k on a transition $\eta \rightarrow [\kappa]$ such that $|\Omega(U_x, \nu_{k-1} + t_{k-1}, \eta)| > 1$; or

  (b) $k \geq 0$ and for some $t \in [0, t_{k+1} - t_k]$, $|\Omega(U_x, \nu_k + t, \bigwedge_{1 \leq p \leq m} I_p)| > 1$,

then there is a $0 < h < k$ such that ν_{h-1} goes to ν_h on a transition $\eta' \rightarrow [\ldots; x := 0; \ldots]$. ||

Condition (2) is a path property which intuitively says that if the current reading of timer x is not tested (with invariance conditions or triggered transitions) before it is reset, then the current reading of timer x is of no influence to system behavior in the future.

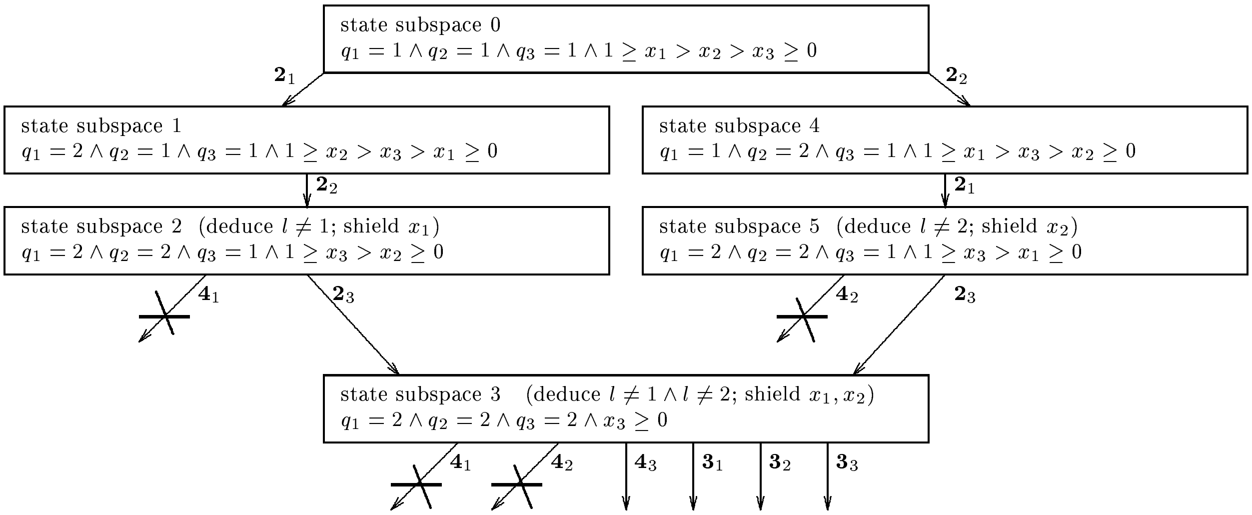

Figure 3: Illustration of lemma 2

LEMMA 2 *For every subformula ψ of the specification formulus and every two states ν, ν' such that ν and ν' are identical except for the readings of x which is shielded in both ν and ν', $S, \nu \models \psi$ iff $S, \nu' \models \psi$.*

Proof : Suppose we have a ν-run $(\nu_0, t_0)(\nu_1, t_1) \dots (\nu_k, t_k) \dots \dots$. We want to establish that we can construct another ν'-run $(\nu'_0, t_0)(\nu'_1, t_1) \dots (\nu'_k, t_k) \dots \dots$ such that for all $k > 0$, ν_{k-1} goes to ν_k and ν'_{k-1} goes to ν'_k on the same transition $e_k = \eta \rightarrow [\kappa]$ of the same process. Note that ν, ν' only differ at readings of timers in Λ.

Assume the construction is impossible because there is a least $k > 0$ such that e_k can be triggered in the ν-run but cannot be triggered in the ν'-run due to either unsatisfied triggering condition or unsatisfied invariance condition. Since the only difference of the two runs is caused by the different readings of x in ν and ν'. This implies either $\Omega(U_x, \nu_{k-1}+t_k-t_{k-1}, \eta) > 1$ or $\Omega(U_x, \nu_k, \bigwedge_{1 \leq p \leq m} I_p) > 1$ which again implies by our shieldedness condition, that before e_k, there is another transtion which has reset x to zero in both runs. This means $(\nu_{k-1}+t_k-t_{k-1})(x) = (\nu'_{k-1} + t_k - t_{k-1})(x)$ which is a contradiction to our assumption. Thus we know the construction can be made and by induction on the subformulus structure, we can show the "iff" relation. $\|$

Example 4 *:* In figure 3, we illustrate how lemma 2 works. Figure 3 can be thought as a condensation from figure 2. Let us examine how this condensation is possible. By analyzing the intermediate state-graphs G after merging the state-graphs for process 1, 2, and 3, we can deduce that in figure 2, x_1 is shielded in state subspace 2; x_2 is shielded in state subspace 5; and x_1, x_2 are shielded in both state subspaces 3 and 6. To convince ourselves of these, let us look at state subspace 2 and timer x_1. Condition (1) is satisfied obviously (check example 2). Condition (2) can be established as follows. In state subspace 2, process 1 is in mode $q_1 = 2$ from which, x_1 is read only by process 1 with transition $\mathbf{4}_1$ before any reset. But from example 3, we already knew that from mode $q_1 = 2$, transition $\mathbf{4}_1$ is not triggerible. Thus we deduce conditions (1) and (2) are both satisfied.

After shielding the clocks in different state subspaces, we find that according to lemma 2, state subspaces 3 and 6 are equivalent. Thus we can just keep one of them. However, we note the readers that the shieldedness of the timers in this example are detected only after we have eliminated arcs using lemma 1. $\|$

Note that the second condition in the definition of shieldedness is a path condition. In section 5, we shall give an algorithm which detects a set of shielded timer in a sequence.

(x_i, x_j)	0	x_1	x_2	x_3
0	$(0, \leq)$	$(0, <)$	$(-2, <)$	$(-3, <)$
x_1	$(1, <)$	$(0, \leq)$	$(-1, <)$	$--$
x_2	$--$	$(5, <)$	$(0, \leq)$	$(1, <)$
x_3	$(4, <)$	$--$	$(1, <)$	$(0, \leq)$

(a)

(x_i, x_j)	0	x_1	x_2	x_3
0	$(0, \leq)$	$(0, <)$	$(-2, <)$	$(-1, <)$
x_1	$(1, <)$	$(0, \leq)$	$(-1, <)$	$(0, <)$
x_2	$(3, \leq)$	$(3, <)$	$(0, \leq)$	$(1, <)$
x_3	$(4, <)$	$(4, <)$	$(1, <)$	$(0, \leq)$

(b)

"$--$" means unspecified.

Table 2: Two zones in Difference Bounds Matrices

4.3 Zones

The prevailing technology of symbolic manipulation partitions real-time system state spaces using the concept of zones[2, 9]. A *zone* is a matrix which defines the integer difference between timer readings. For example, in a system state with timers $x_1 = \pi, x_2 = 4, x_3 = \frac{25}{6}$, a zone representation which the state is in is shown in table 2(a). A special timer 0 is added as x_0 which always reads zero. Entry $(x_i, x_j) = (c, \sim)$ means that $x_i - x_j \sim c$. By applying the all-pair shortest path algorithm, we can transform the zone representation to a canonical representation, as shown in table 2(b).

In our implementation, we shall record the zones instead of the actual readings of timers for each state. This symbolic technique is also crucial in condensation of state-space representations.

4.4 Region and transition condensation by symmetry

In a real-time concurrent system with isomorphic processes, utilizing symmetry is a must in efficient verification. The idea is very similar to the one presented in [10] except we extend the symmetry to cover timing inequalities of timers in zones. For example in figure 3, after permuting process identifiers 1 and 2, we find that state subspaces 1 and 4 are equivalent after permutation; and state subspaces 2 and 5 are also equivalent after permutation. Thus we can delete state subspaces 4 and 5 by connecting an arcs from subspaces 0 to 1 with transition $\mathbf{2}_2$ and permutation (2 1 3). Note this condensation leaves the state-graph as a multi-graph.

5 Implementation

This section describes some implementation details. Compared to table 1, we have implemented the **merge**() operator and two condense operators. $\mathbf{condense}_1()$ is **shield**() and utilizes lemma 2 to detect the equivalence among state subspaces. $\mathbf{condense}_2()$ is **symmetry**() and utilizes a process identifier sorting procedure to calculate process identifier permutations and detect symmetric equivalence under process identifier permutation.

- **On-the-fly merging of two state-graphs**
  Here we define operator **merge**() which given two state-graphs G_1, G_2 in a real-time concurrent system, returns a new state-graph $\mathbf{merge}(G_1, G_2)$ representing the behavior of all processes described by G_1 and G_2. The operator utilizes on-the-fly construction technique. In addition, it uses lemma 1 to further detect behavior-equivalence among state subspaces.
- **Shielded timer detection**
  Lemma 2 tells us that we can conceal literals of those shielded timers in a state subspace recording. To utilize the timer-shieldedness technique in subsection 4.2, we have to record the set of shielded timers in each state-graph node. One tricky point is that two timers can both be individually shielded in a state but they are not together shielded. The reason follows. Given a state ν and a conjunction like $L \wedge x < 1 \wedge x' < 2$, it may happen that $\nu \models L$, $\nu \not\models x < 1$, and $\nu \not\models x' < 2$. Thus it is clear that when we conceal both $x < 1$

shield(G) /* $G = \langle V, v_0, \mathit{Shielded}, E\rangle$ with state subspace labelling function *Shielded*() and timers $x_1, \ldots, x_n$ in sequence */ {

(1) For each $v \in V$, do {

  (1) let $\mathit{Shielded}(v) := \emptyset$;

  (2) For $i := 1$ to n, if x_i is shielded in v, then {

    (1) $\mathit{Shielded}(v) := \mathit{Shielded}(v) \cup \{x_i\}$;

    (2) conjoin out (see table 4) x_i from all state subspace recordings of v and outgoing transition recordings from v.

    }

  }

(2) return G;

}

Table 3: the **shield**() operator

conjoin$_Y(\phi)$ /* ϕ is a conjunction of literals. */ {

(1) Let $\eta := \phi$;

(2) Iteratively for each $y \in Y$, do {

  (1) Rewrite every literal of ϕ such that all literals take one of the following three forms :

  - **Type 1** : $\omega \sim x$ where ω is x-free and $\sim \in \{\leq, <\}$;
  - **Type 2** : $x \sim \omega$ where ω is x-free and $\sim \in \{\leq, <\}$;
  - **Type 3** : in the literal, there is no x-appearances.

  Let G_1, G_2, G_3 be the sets of **Type 1, 2, 3** literals respectively.

  (2) Let $\eta := \left(\bigwedge_{\omega\sim x\in G_1; x\sim'\omega'\in G_2} \omega \dot{\sim}(\sim,\sim')\, \omega'\right) \wedge \left(\bigwedge_{\tau\in G_3} \tau\right)$ where $\dot{\sim}(\sim,\sim')$ is computed according to the table on the right.

$\dot{\sim}$	$\leq$	$<$
$\leq$	$\leq$	$<$
$<$	$<$	$<$

  }

(3) return η;

}

Table 4: conjoin$_x(\Phi)$ to remove variable x from Φ

and $x' < 2$, the truth value of the conjunction changes. A remedy for this problem is to one by one test the shieldedness of timers and condense state subspaces as shown in table 3.

- **Symmetry among transitions**

  symmetry(G) condenses state-graphs by adapting the process symmetry technique[10] to real-time timing constraints. Basically, we build the operator on a sorting procedure for the process identifiers in a state subspace. The output of the sorting procedure is a permutation on process identifiers which is then recorded on transitions.

  One implementation detail is about how to condense on the symmetry of transitions. The condensation by symmetry will introduce multiplicity of arcs between a pair of state subspaces. Thus it is important to detect the equivalence under symmetry of those arcs with labeled permutations.

  One intrigueing scheme is to choose to keep only one transition among those equivalents under symmetry. Thus more computation will be needed to reconstruct back those deleted transitions while we need them in the merger and model-checking procedure. However, we can also show that for symmetric TCTL formulas, the answer for model-checking is not affected even if we do not reconstruct those deleted transitions. Due to page-limit, we have to abandon the elaboration.

	Balarin					Wang &
#proc	strong	weak	KRONOS	Wong-Toi	UPPALL	Hsiung
6	115s	18s	1174s	74s	$\approx 20s$	20s
7	398s	48s	M/O	164s	$\approx 140s$	62s
8	986s	116s	M/O	375s	N/A	182s
9	2220s	247s	M/O	891s	N/A	483s
10	M/O	576s	N/A	N/A	N/A	1212s
11	N/A	N/A	N/A	N/A	N/A	3057s

times in seconds; M/O: memory overflow; N/A: not available;

Table 5: Verification time comparison

	#proc	2	3	4	5	6	7	8	9	10	11
merge()	#region	70	225	555	1123	2017	3347	5231	7801	11204	1545
	#transitions	160	729	2119	4696	9125	16364	26910	42808	65514	7608
shield()	#region	31	100	231	447	774	1241	1880	2726	3817	1194
	#transitions	70	307	842	1794	3379	5844	9301	14340	21336	6366
symmetry()	#region	16	39	76	130	204	301	424	576	760	220
	#transitions	35	109	247	472	810	1290	1944	2807	3917	1064

Table 6: Intermedate state-graph sizes for 10 processes

6 Experiments on Fisher's mutual exclusion protocol

We have experimented our approach and algorithms on Fischer's timed mutual exclusion protocol (check example 1). Table 6 compares our performance with those performance numbers presented in [4]. Our implementation is on Sun UltraSparc 200MHz. Note that Balarin and Wong-Toi's work are all based on approximation while ours is based on exact verification. Besides the hardware performance upgrade, we note the readers that the time complexity of our implementation increases slower, with respect to concurrency sizes, than previous research on exact verification.

Specifically, we list the sizes of intermediate state-graphs for the verification of Fischer's protocol with 11 processes in table 6. The sudden jump-down of state-graph sizes from 10 to 11 is due to causality analysis. A lot of the variable-value changes in the intermediate state-graphs are not explained by causality while in the final state-graph, every one of them must be explained. Note that in our approach, the space requirement after the index-normalization and timer-shielding is reduced to a factor of one order of magnitude in each iteration.

In fact, when we analyze the state subspaces generated in the iterations in our algorithm, we observe the following pattern. The set of processes is partitioned into five groups. The first two groups contain processes in mode $q_p = 0$ and $q_p = 1$ respectively. The second group contains all processes in mode $q_p = 2$ except the one that enters mode $q_p = 2$ last. The fourth group contains all processes in mode $q_p = 3$. The fifth group contains a single process which last enters mode $q_p = 2$. We found that except for timers used by processes in group 2 and 5, all other timers are shielded. This analysis can be carried out manually to show that our time complexity against Fischer's mutual exclusion protocol is polynomial.

7 Conclusion

We propose to iteratively refine and condense state-graphs for efficient verification. There is abundant knowledge in those intermediate state-graphs which can be deduced to condense the state-space representation. In the long run, we anticipate to have more proven state-graph operators coming out which extend to another dimension of verification efficiency.

References

[1] R. Alur, C. Courcoubetis, D.L. Dill. Model Checking for Real-Time Systems, IEEE LICS, 1990.

[2] R. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, H. Wong-Toi. An Implementation of Three Algorithms for Timing Verification Based on Automata Emptiness. IEEE RTSS, 1992.

[3] R. Alur, T.A. Henzinger, P.-H. Ho. Automatic Symbolic Verification of Embedded Systems. in Proceedings of 1993 IEEE Real-Time System Symposium.

[4] F. Balarin. Approximate Reachability Analysis of Timed Automata. IEEE RTSS, 1996.

[5] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L.Dill, L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond, IEEE LICS, 1990.

[6] R.E. Bryant. Graph-based Algorithms for Boolean Function Manipulation, IEEE Trans. Comput., C-35(8), 1986.

[7] E. Clarke and E.A. Emerson. Design and Synthesis of Synchronization Skeletons using Branching-Time Temporal Logic, Proceedings of Workshop on Logic of Programs, Lecture Notes in Computer Science 131, Springer-Verlag, 1981.

[8] E. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications, ACM Transactions on Programming Languages and Systems 8(2), 1986, pp. 244-263.

[9] D.L. Dill. Timing Assumptions and Verification of Finite-state Concurrent Systems. CAV'89, LNCS 407, Springer-Verlag.

[10] E.A. Emerson, A.P. Sistla. Utilizing Symmetry when Model-Checking under Fairness Assumptions: An Automata-Theoretic Approach. ACM TOPLAS, Vol. **19**, Nr. 4, July 1997, pp. 617-638.

[11] T.A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine. Symbolic Model Checking for Real-Time Systems, IEEE LICS 1992.

[12] K.L. McMillan, "Symbolic Model Checking", Kluwer Academic Publishers, Boston, MA, 1993.

[13] F. Wang, A.K. Mok, E.A. Emerson. Real-Time Distributed System Specification and Verification in APTL. ACM TOSEM, Vol. 2, No. 4, Octobor 1993, pp. 346-378.

[14] F. Wang. Timing Behavior Analysis for Real-Time Systems. IEEE LICS 1995.

[15] F. Wang. Reachability Analysis at Procedure Level through Timing Coincidence. in Proceedings of the 6th CONCUR, Philadelphia, USA, August 1995, LNCS 962, Springer-Verlag.

[16] F. Wang, C.T. Lo. Procedure-Level Verification of Real-Time Concurrent Systems. to appear in Proceedings of the 3rd FME, Oxford, Britain, March 1996; in LNCS, Springer-Verlag.