Verification of Dynamic Linear Lists for All Numbers of Processes[⋆]

Farn Wang

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

In real-world design and verification of concurrent systems with many identical processes, the number of processes is never a factor in the system correctness. This paper embodies such an engineering reasoning to propose an almost automatic method to safely verify safety properties of such systems. The central idea is to construct a finite collective quotient structure (CQS) which collapses state-space representations for all system implementations with all numbers of processes. The problem is presented as safety bound problem which ask if the number of processes satisfying a certain property exceeds a given bound. Our method can be applied to systems with dynamic linear lists of unknown number of processes. Processes can be deleted from or inserted at any position of the linear list during transitions. We have used our method to develop CQS constructing algorithms for two classes of concurrent systems : (1) untimed systems with a global waiting queue and (2) dense-time systems with one local timer per process. We show that our method is both sound and complete in verifying the first class of systems. The verification problem for the second class systems is undecidable even with only one global binary variable. However, our method can still automatically generate a CQS of size no more than 1512 nodes to verify that an algorithm in the class: Fischer's timed algorithm indeed preserves mutual exclusion for any number of processes.

[⋆] The work is partially supported by NSC, Taiwan, ROC under grant NSC 88-2213-E-001-002.

Verification of Dynamic Linear Lists for All Numbers of Processes[2]

Farn Wang

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

1 Introduction

Real-world concurrent systems with many identical processes are designed and verified without assumption on the number of participating processes [17, 18]. When we look at the manual verification of such systems, we very often find the *engineering reasoning* that

"at any moment, relations among processes in different operation modes are more important than the acutal numbers of those processes."

Previous researchers did not take advantage of this kind of engineering reasoning in automatic verification research. We believe that without incorporating such design reasoning in verification theory, state-space explosion phenomenon are really too hard to cope with. We propose an almost automatic method based on the above-mentioned engineering reasoning for the verification of dynamic linear networks of unknown numbers of processes. Processes are allowed to be inserted at or deleted from any position in the linear list during transitions. Note that the linear lists are not necessarily static sequences of processes ordered by their identifiers, but may rather dynamically change their element positions in system operations. Such linear lists can happen explicitly as in Peterson's algorithm[18] and MCS lock-spin algorithms[17]. They can also implicitly happen in dense-time systems as the ordering among timers' reading fractional parts[1]. Our method needs very little human guidance and are applied to verify two example algorithms, one of which is Fischer's timed algorithm[6, 22]. The structure constructed for Fischer's algorithm has no more than 1512 nodes.

We are dealing with concurrent systems with unknown numbers of processes running different copies of a same *process program*. We shall write $S_m = \|_{1\leq p\leq m} P_p$ for a *system implementation* of m concurrent processes running the same process program P_p with p as the process identifier parameter variable.

Example 1. : We shall use a mutual exclusion algorithm with a single queue to demonstrate our idea. In Figure 1, we have an automaton representing a local process in such a system. The global queue is named Q and the process is identified by p. The circles are operation modes of processes while the arcs

[2] The work is partially supported by NSC, Taiwan, ROC under grant NSC 88-2213-E-001-002. A more complete version of the paper can be found in technical report TR-IIS-98-019, IIS, Academia Sinica, Taiwan, ROC

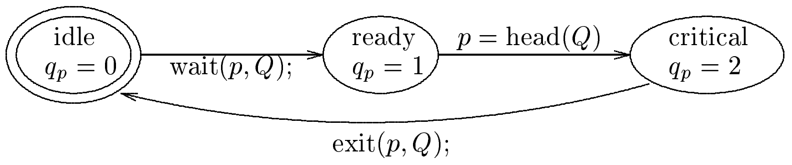

Fig. 1. A concurrent algorithm with a global queue

are transitions. Above each arc, we label a transition triggering condition, if any. Under each arc, we label actions to be taken at the firing of the transition. "wait(p,Q);" is an action that appends process p at the end of queue Q. If p is already in the queue, then the statement has no effect. "p=head(Q)" is a triggering condition on whether p is the queue head. "exit(p,Q);" is an action that deletes process p from queue Q. If p is not in the queue, then the statement has no effect. Interleaving semantics is assumed. Initially the queue is empty and all variables (including q_p's) contains zeros.

Note a process enters critical section only when it is the queue head. Since at any moment, there is only one queue head, mutual exclusion is guaranteed. $\|$

Unlike the previous approaches[2, 7, 8, 11, 12, 15, 16], we model our problem as *safety bound problem* which, given a property η and a safety bound C, asks if there is a computation of a system implemented with some processes such that along the computation, at some moment, more than C processes satisfy η. Such a framework can be used to model mutual-exclusion problems, local state reachability problem (which asks if a process can reach an operation mode in some computation of some implememntation), reader-writer problem, ... etc.

Our verification technique is to construct a finite directed graph $Q = \langle V, I, E\rangle$, called *collective quotient structure (CQS)*, with the following restrictions.

- V is a finite set of global state images for all system implementations with any number of processes. Engineers' wisdom and experiences in design and verification is encoded in the mapping from states to images in V.
- $I \subseteq V$ is the set of initial state images.
- $E \subseteq V \times V$ defines the transitions between state images such that $(v, v') \in E$ iff there are two states ν, ν' of a system implementation S_m, $m \in \mathcal{N}$, the images of ν, ν' are v, v' respectively, and S_m may transit from ν to ν'.

For safety analysis, if we can construct a finite CQS in which the dangerous state images are not reachable from any images in I, then it is good enough to conclude that the system is safe for any number of processes. But if there is a path from some images in I to a dangerous state image, there is no conclusion to make because arcs along the path may be induced by transitions from different system implementations with different numbers of processes. Thus our method can only safely verify systems. However, with well-known worst-case complexities like PSPACE[1], EXPTIME, EXPSPACE, undecidability[3, 4, 5, 19] of many classic verification paradigms, we believe our method is a practical and plausible trade-off between complexity management and verification algorithmicity. By properly encoding human wisdom and experiences into the state-image mappings, our method has good chances to automatically and efficiently verify systems designed with well-accepted engineering rules and verification experiences. In our experience of analyzing our CQS against example 1 and Fischer's timed algorithm, our method reasons very much in the same way as a human engineer will verify algorithms.

The kernel technology in our CQS constructing algorithms is a mapping from linear lists of any lengths to finitely many images (called **PCL**-*images*, acronym for *Process-state-type Counter List*). We shall demonstrate our technique with CQS constructing algorithms for untimed concurrent systems with a single waiting queue. Lemma 2 shows that our method is both sound and complete in verifying systems in this class. We have also applied our technology to dense-time systems [1], with one local timer per process, in which the ordering among fractional parts of all timer readings forms an implicit dynamic linear list. We have shown that reachability problem of such systems, even with restrcition that only one binary global variable may be used, is still undecidable [21]. However, our method can verify Fischer's timed algorithm[6, 22] in this class with a CQS no larger than 1512 nodes.

We have some notations. Given a set or sequence K, $|K|$ is the number of elements in K. For each element e in K, we also write $e \in K$. $\mathcal{N}$ is the set of nonnegative integers. $\mathcal{R}^+$ is the set of nonnegative reals.

2 Related work

Apt and Kozen already showed that in general verification of systems with unknown number of concurrent processes is undecidable[5]. This means that such verification problems are extremely hard and we can only rely on semi-decision procedures or, as in this work, approximation algorithms to answer them. Otherwise, we can also investigate to find out decidable subclasses of the problem. In the following, we briefly describe some of the related work.

Browne, Clarke, and Grumberg [7] use bisimulation equivalence relation between global state graphs of systems of different sizes. The equivalence relation must be strong enough for the method to work. Thus the construction of the equivalence relation is difficult to mechanize.

Clarke, Grumberg, and Jha[11] propose to use regular languages to specify properties in a linear network with unknown number of processes. Then state-equivalence relation is defined based on the regular languages and a mechanical method is defined to synthesize a network invariant $\mathcal{I}$ in the hope that $\mathcal{I}$ can be contained by the specification. But there is no guarrantee that $\mathcal{I}$ is a model of the specification even if the system indeed satisfies the specification. Moreover, it is not known whether using the specification regular languages to derive equivalence class properly perserves the reasonings behind the system design. Lesens, Halbwachs, and Raymond[16] furthered the approach by designing a language for the specification in systems with complex structures and by using fixed-point resolution with different heuristics to calculate many network invariants. Compared to our approach, we argue that the technique of **PCL**-images better captures the design reasoning that the relations between processes in different states are far more important than the actual numbers of processes in different states. We believe in verifying complex systems, without utilizing the reasoning behind the system designs, state-explosion problem cannot be properly dealt with.

Kurshan and McMillan[15] proposes to use network structural induction which is not guarranteed to terminate. Also inductive hypothesis is difficult to construct, although once it is ready, the whole approach is usually very efficient. Compared to our approach, we are using an approximation algorithm which captures the engineers' view of linear list. Users only have to guess the value of bound B, used in CQS construction, which for many real-world concurrent algorithms, small value like 1 will do.

Emerson and Naamjoshi[12] specialized on static token ring networks. They prove that for certain properties, verification on small size networks can be used to guarrantee the verification of large size networks. In contrast, our method is applicable to all different forms of *"dynamic"* linear networks of processes.

Boigelot and Godefroid[8] choose to use state-space exploration to handle the verification problems of systems with unbounded FIFO queues. Their state-space representation is constructed by collapsing FIFO queues. Their approach does not guarrantee termination.

3 PCL-images of linear lists of all lengths

A *list recording (LR)* on symbol set U is a pair (ρ, λ) such that ρ and λ are respectively a finite sequence and a multiset on U. λ represents the set of processes not joining the list ρ at the moment. For example, when $U = \{\mu_1, \mu_2, \mu_3, \mu_4\}$, an LR on U is $(\mu_4\mu_2\mu_2\mu_2\mu_2, (\mu_1 : 3, \mu_2 : 0, \mu_3 : 0, \mu_4 : 0))$. We adopt notation like $(\mu_1 : 3, \mu_2 : 0, \mu_3 : 0, \mu_4 : 0)$ for a multiset in which there are three μ_1's and nothing else.

3.1 Classification of process state type

Since all our processes are of finite-state nature, we can classify process states into finitely many *PSTs (Process State Types)* which defines correspondence among states of different processes. Process states s, s' respectively of processes p, p' are in the same PST if by substituting p for p' in the recording of s, we get s', and vice versa. By representing each process in state recordings as its PST, we can thus omit process identifiers from state recordings. It will be clear from sections 4 and 5, such omission will not affect our ability for safety bound verification.

In our method, we need to construct a *parametric atomic proposition set (PAP set)* to distinguish between processes in different PSTs. The atomic proposition set is parametric because the propositions may contain process identifier parameter variable p. For example, in example 1, the PAP set is $\{q_p = 0, q_p = 1, q_p = 2, p = \text{head}(Q)\}$. To classify a process, say process 1, we then have to substitute all occurrences of p in the set for 1. Thus process state $\{q_1 = 0, 1 = \text{head}(Q)\}$ of process 1 is of the same PST as process state $\{q_2 = 0, 2 = \text{head}(Q)\}$ of process 2. In section 5, we shall demonstrate the mechainical composition of the PAP sets for the untimed systems as in example 1.

Symbolically, we can represent a PST as a subset of the PAP set A. And from now on, we shall do so for convenience. Continuing from example 1, we have four PSTs: $\mu_1 = \{q_p = 0\}, \mu_2 = \{q_p = 1\}, \mu_3 = \{q_p = 1, p = \text{head}(Q)\}, \mu_4 = \{q_p = 2, p = \text{head}(Q)\}$. A reachable state for a system with eight processes can be described by the list recording (LR) of $\Pi = (L, \bar{L}) = (\mu_4\mu_2\mu_2\mu_2\mu_2, (\mu_1 : 3, \mu_2 : 0, \mu_3 : 0, \mu_4 : 0))$. $\mu_4\mu_2\mu_2\mu_2\mu_2$ means that there are five processes participating the linear list and the first one is of PST μ_4 while the rest are of PST μ_2. Let U_A be the set of PSTs that A can distinguish.

3.2 PCL-images

Our kernel technology is a mapping scheme from linear lists of all lengths to finitely many images called *PST counter lists* (**PCL**). The design of **PCL**-images is due to the observation that while verifying safety properties, the actual number of processes in a PST is very often unimportant while the relations among processes in different PSTs are crucial. For example, in verifying the algorithm in example 1, we may say

"While a process p is in critical section, another process p' cannot enter the critical section because p' needs to be the queue head to enter the critical section and p is the queue head at the moment."

Also, to verify Fischer's timed mutual exclusion algorithm[6, 22], we may say

"After a process p enters critical section, the first process, say p', which overwrites $l = p$ to $l = p'$ must detect $l = 0$ true before p enters the critical section."

But what are the important relations ? In linear lists used as queues or stacks, the relative positions of processes in different PSTs toward the head and tail can be important. In dense-time systems, the fractional part relative magnitudes of timer readings of processes in different PSTs can be important. Especially, at any state, we need to know which PST has a process whose timer's reading will advance to an integer value in the next time event. Thus we shall propose the following **two design guidelines of PCL-images**:

(1) For each PST type, we shall only record its first and last segments of consecutive occurrences in the linear lists. Processes in between two such segments will be merged into a single group of processes. Thus, information of ordering among processes inside the same group is omitted.

(2) For each group (or segment) of processes, we shall then record its number of processes of each PST up to a bound B. The actual value of B can be

chosen by users or iteratively tested for by a procedure. By choosing a value for B, the verification engineers should have the intuition that when a PST has more than B processes, the target safety property will not be affected by the actual number of processes in that PST. We suspect that for a lot of well-designed systems, like the two example algorithms in section 5 and 6, $B = 1$ will work.

We need the new number system which respects a given bound B. Thus for every $c \in \mathcal{N}$, $c^{\langle B \rangle} = c$ if $c \leq B$; or $c^{\langle B \rangle} = \infty$ else. In this paper, ∞ is used to represent any number greater than B. Also $c + \infty = \infty + c = \infty$ and $c < \infty$.

We need the following notations for rigorous definition. A *PST counter* is a mapping from U_A to $\mathcal{N}$. Given a *PST counter* γ, $\gamma^{\langle B \rangle}$ is called a *PST counter with bound* B and for each $\mu \in U_A$, $\gamma^{\langle B \rangle}(\mu) = (\gamma(\mu))^{\langle B \rangle}$. Let $X_A{}^{\langle B \rangle}$ be the set of PST counters bounded by B with processes classified by PAP set A.

We execute **guideline (1)** by marking for each PST, its first segment (sublists in the linear list) and last segment of consecutive occurrences in the linear list. Such markings cut the linear list into segments. For example, we may have four PSTs $\mu_1, \mu_2, \mu_3, \mu_4$ such that $\Pi = (L, \bar{L})$ with $L = \mu_1\mu_2\mu_1\mu_2\mu_1\mu_1\mu_2\mu_3\mu_3\mu_3\mu_1$ and $\bar{L} = \{\mu_1 : 0, \mu_2 : 0, \mu_3 : 0, \mu_4 : 1\}$. Our marking scheme will divide L into the following segments.

$$\overbrace{\mu_1}\ \overbrace{\mu_2}\ \mu_1\mu_2\mu_1\mu_1\ \overbrace{\mu_2}\ \overbrace{\mu_3\mu_3\mu_3}\ \overbrace{\mu_1}$$

Here braces are labeled over marked segments for clarity. Subsequence $\mu_1\mu_2\mu_1\mu_1$ has no brace over it because no processes in it represents either first or last occurrences of a PST in the linear list.

We then execute **guideline (2)** by collapsing each segment (including the marked segments) down to a PST counter with bound B. Continuing with the example in last paragraph with $B = 5$, we can collapse each segment down to a PST counter with bound 5 to get $\mathbf{PCL}^{\langle 5 \rangle}(\Pi) = (\mathbf{PCL}^{\langle 5 \rangle}(L), \bar{L}^{\langle 5 \rangle})$ with $\mathbf{PCL}^{\langle 5 \rangle}(L)$ equal to

$$\begin{pmatrix}\mu_1 : 1\\ \mu_2 : 0\\ \mu_3 : 0\\ \mu_4 : 0\end{pmatrix} \begin{pmatrix}\mu_1 : 0\\ \mu_2 : 1\\ \mu_3 : 0\\ \mu_4 : 0\end{pmatrix} \begin{pmatrix}\mu_1 : 3\\ \mu_2 : 1\\ \mu_3 : 0\\ \mu_4 : 0\end{pmatrix} \begin{pmatrix}\mu_1 : 0\\ \mu_2 : 1\\ \mu_3 : 0\\ \mu_4 : 0\end{pmatrix} \begin{pmatrix}\mu_1 : 0\\ \mu_2 : 0\\ \mu_3 : 3\\ \mu_4 : 0\end{pmatrix} \begin{pmatrix}\mu_1 : 1\\ \mu_2 : 0\\ \mu_3 : 0\\ \mu_4 : 0\end{pmatrix}$$

When $B = 2$, $\mathbf{PCL}^{\langle 2 \rangle}(\Pi) = (\mathbf{PCL}^{\langle 2 \rangle}(L), \bar{L}^{\langle 2 \rangle})$ with $\mathbf{PCL}^{\langle 2 \rangle}(L)$ equal to

$$\begin{pmatrix}\mu_1 : 1\\ \mu_2 : 0\\ \mu_3 : 0\\ \mu_4 : 0\end{pmatrix} \begin{pmatrix}\mu_1 : 0\\ \mu_2 : 1\\ \mu_3 : 0\\ \mu_4 : 0\end{pmatrix} \begin{pmatrix}\mu_1 : \infty\\ \mu_2 : 1\\ \mu_3 : 0\\ \mu_4 : 0\end{pmatrix} \begin{pmatrix}\mu_1 : 0\\ \mu_2 : 1\\ \mu_3 : 0\\ \mu_4 : 0\end{pmatrix} \begin{pmatrix}\mu_1 : 0\\ \mu_2 : 0\\ \mu_3 : \infty\\ \mu_4 : 0\end{pmatrix} \begin{pmatrix}\mu_1 : 1\\ \mu_2 : 0\\ \mu_3 : 0\\ \mu_4 : 0\end{pmatrix}$$

Note there can be at most $4|U_A| - 3$ PST counters in any linear list $\mathbf{PCL}^{\langle B \rangle}$-images. Given fixed B and U_A, it can be shown that over the domain of all states of all system implementations with such a linear list, there are only finitely many $\mathbf{PCL}^{\langle B \rangle}$-images of list recordings. Moreover, $\mathbf{PCL}^{\langle B \rangle}$-images indeed capture the relations between processes of different PSTs.

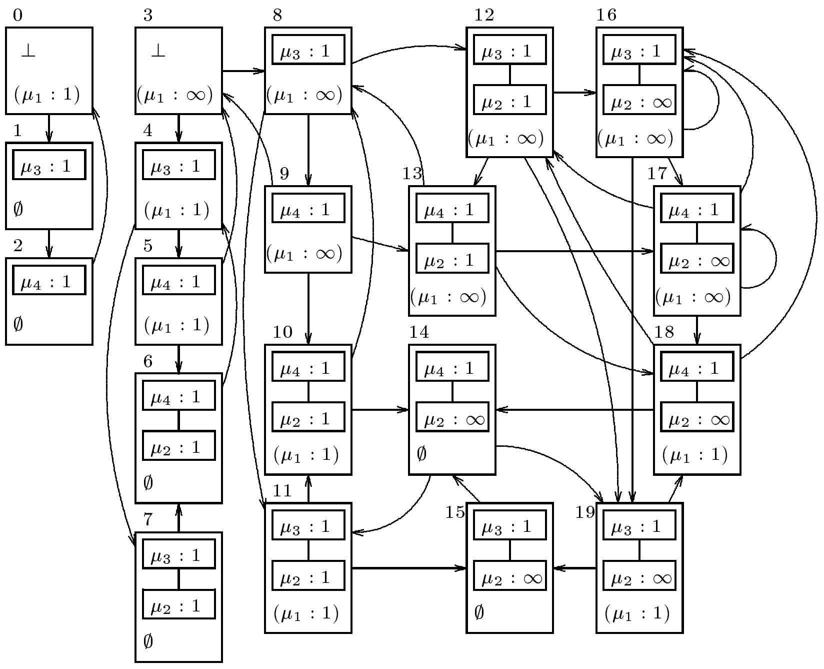

Fig. 2. The CQS for example 1 with $B = 1$

Example 2. : In Figure 2, we have the CQS constructed for example 1 with $B = 1$. The big rectangles represent global state-images, with indices labeled on their upper-left corners, while arrows represent transitions. Inside the big rectangles, we stack the list $\mathbf{PCL}^{\langle 1 \rangle}$-images above the multiset $\mathbf{PCL}^{\langle 1 \rangle}$-images. $\perp$ is the null list. Note ∞ represents any numerical constant bigger than B. For each nonnull list, we draw it as a sequence of boxes with head on top. Inside a box or the parentheses, we only write the nonzero mapping values. For example, $\{\mu_1 : 1\}$ actually means $\{\mu_1 : 1, \mu_2 : 0, \mu_3 : 0, \mu_4 : 0\}$. The two images with null list image are the images for initial global states.

Let us examine image 18. At any algorithm implementation global state represented by image 18, the queue head process is in the critical section and the queue tail has more than one processes which are all in PST μ_2. There is also only one process in the idle state μ_1 and not in the queue.

Let us examine the arc from image 18 to 16. This can happen when the implementation has three or more processes of PST μ_2 in the waiting queue and a μ_4 process leaves the critical section while being removed from the queue.

The CQS is quite small with only 20 global state-images. It is clear from the CQS that the algorithm preserves mutual-exclusion for any number of processes since there is no reachable image violates the mutual-exclusion. ||

The rest of the paper describes the algorithmic construction of CQS for any given B. Before that, a formal definition of **PCL**-images of list recordings follows. A $\chi \in X_A{}^{\langle B\rangle}$ is called a *marker counter* of type μ if $\chi(\mu) > 0$ while $\chi(\mu') = 0$ for all $\mu' \neq \mu$. Given $L = \mu_1 \ldots \mu_m$, $\#_\mu(L) = |\{i \mid 1 \leq i \leq m; \mu_i = \mu\}|$.

Definition 1. : **PCL**-images bounded by constants A *PST counter segmentation* of a linear list $L = \mu_1 \ldots \mu_m$ with respect to bound B is a linear list $\chi_1\chi_2 \cdots \chi_n$ of PST counters in $X_A{}^{\langle B\rangle}$ such that L can be partitioned into n disjoint segments $L_1L_2 \ldots L_n$ and for each $\mu \in U_A$ and $1 \leq i \leq n$, $\chi_i(\mu) = (\#_\mu(L_i))^{\langle B\rangle}$. A PST counter segmentation $\rho = \chi_1 \ldots \chi_n$ is called a **PCL**-*image* of its linear list L with bound B, in symbols $\mathbf{PCL}^{\langle B\rangle}(L)$, iff ρ satisfies the following restrictions.

- For all $\mu \in U_A$ and $1 \leq i \leq n$ with $\chi_i(\mu) > 0$,
  - if there is no $1 \leq j < i$ with $\chi_j(\mu) > 0$, then for all $\mu' \in U_A - \{\mu\}$, $\chi_i(\mu') = 0$; and
  - if there is no $i < j \leq n$ with $\chi_j(\mu) > 0$, then for all $\mu' \in U_A - \{\mu\}$, $\chi_i(\mu') = 0$;

  This restriction forces the first segment and last segment of consecutive occurrences of a PST be represented as PST counters in the $\mathbf{PCL}^{\langle B\rangle}$-images.
- For all $1 \leq i < n$, at least one of χ_i, χ_{i+1} is the first or last marker counter of a PST in L. This restriction collapses infinitely many linear lists of all lengths into finitely many sequences of PST counters with bounded length.
- There is no $1 \leq i \leq n$ such that for all $\mu \in U_A$, $\chi_i(\mu) = 0$. Intuitively, this means empty segments are not presented to help save spaces.
- There is no $1 \leq i < n$ such that χ_i, χ_{i+1} are both marker counters of the same PST. This helps save spaces too.

The $\mathbf{PCL}^{\langle B\rangle}$-image of an LR $\Pi = (L, \bar{L})$, in symbols $\mathbf{PCL}^{\langle B\rangle}(\Pi)$, is then $(\mathbf{PCL}^{\langle B\rangle}(L), \bar{L}^{\langle B\rangle})$. Given a state recording ν which contains an LR Π as a component, the $\mathbf{PCL}^{\langle B\rangle}$-image of ν, in symbols $\mathbf{PCL}^{\langle B\rangle}(\nu)$, is identical to ν except Π is replaced by $\mathbf{PCL}^{\langle B\rangle}(\Pi)$. ||

3.3 Manipulators on $\mathbf{PCL}^{\langle B\rangle}$-images

The following handy manipulators to extract parts of a linear list help us conveniently construct CQS. Suppose we are given a sequence $\rho = \chi_1 \ldots \chi_n$. $\mathbf{front}(\rho)$ and $\mathbf{tail}(\rho)$ are new sequences obtained from ρ by deleting respectively the last and the first elements from ρ. $\mathbf{head}(\rho)$ and $\mathbf{last}(\rho)$ are respectively the first and the last elements in ρ. $\mathbf{element}(\rho, i)$, with $1 \leq i \leq |\rho|$, is the i'th process element in list ρ. For example, $\mathbf{front}(abcddd) = abcdd$, $\mathbf{last}(abacfe) = e$, and $\mathbf{element}(abcdd, 3) = c$.

$\mathbf{addhead}(\rho, \chi)$ and $\mathbf{addtail}(\rho, \chi)$ are the sequences obtained by adding χ as the new first and the new last elements respectively of ρ. For example, $\mathbf{addhead}(abacdd, e) = eabacdd$ and $\mathbf{addtail}(abacdd, e) = abacdde$.

Given a linear list $\mathbf{PCL}^{\langle B\rangle}$-image $\rho = \chi_1\chi_2 \ldots \chi_n$, an integer $i \in [1, n]$, and a PST $\mu \in U_A$, $\mathbf{inc}^{\langle B\rangle}(\rho, i, \mu, \rho')$ is true iff ρ' is identical to ρ except that $(\mathbf{element}(\rho, i)(\mu) + 1)^{\langle B\rangle} = \mathbf{element}(\rho', i)$. $\mathbf{dec}^{\langle B\rangle}(\rho, i, \mu, \rho')$ is true iff ρ' is identical to ρ except that $\mathbf{element}(\rho,i)(\mu) = (\mathbf{element}(\rho',i)+1)^{\langle B\rangle}$. Given PST counters χ,χ' and a PST μ, $\mathbf{add}\mathit{1}^{\langle B\rangle}(\chi,\mu,\chi')$ is true iff χ is identical to χ' except $(\chi(\mu)+1)^{\langle B\rangle} = \chi'(\mu)$. $\mathbf{del}\mathit{1}^{\langle B\rangle}(\chi,\mu,\chi')$ is true iff χ is identical to χ' except $\chi(\mu) = (\chi'(\mu)+1)^{\langle B\rangle}$. Given a PST μ, $\mathbf{make}1(\mu)$ is the PST counter in ${X_A}^{\langle B\rangle}$, with $B > 0$, that maps μ to 1 and everything else to zero.

Remember in definition 1, more requirements are on $\mathbf{PCL}^{\langle B\rangle}$-images than on PST counter segmentations. Given a PST counter segmentation ρ and a linear list $\mathbf{PCL}^{\langle B\rangle}$-image ρ', relation $\mathbf{normal}(\rho,\rho')$ answers with algorithm if ρ' can be the $\mathbf{PCL}^{\langle B\rangle}$-image for a linear list represented by ρ. $\mathbf{normal}()$ will be handy in restoring manipulated PST counter segmentations back to their $\mathbf{PCL}^{\langle B\rangle}$-image format. $\mathbf{normal}(\rho,\rho')$ can be calculated with a nondeterministic algorithm by first picking the first and last occurrences of each PST in ρ to divide ρ into smaller segments. Then adjacent segments are merged to eliminate those segments which

- do not correspond to the first, or last occurrences of some PSTs, or
- are not bounded before and after by the first or the last occurrences of some PSTs.

4 Safety bound problem and our framework

A *parametric propositional formulus* η on PAP set A has the following syntax.

$$\eta ::= \alpha \mid \neg\eta_1 \mid \eta_1 \vee \eta_2$$

α is a PAP in A. Parentheses may be used to disambiguate the syntax. Traditional shorthands are $\eta_1 \wedge \eta_2 \equiv \neg((\neg\eta_1)\vee(\neg\eta_2))$ and $\eta_1 \rightarrow \eta_2 \equiv (\neg\eta_1)\vee\eta_2$. The satisfaction of a parametric propositional formulus η by a PST μ, written $\mu \models \eta$, is defined inductively as follows.

- $\mu \models \alpha$ iff $\alpha \in \mu$
- $\mu \models \neg\eta_1$ iff it is not the case that $\mu \models \eta_1$
- $\mu \models \eta_1 \vee \eta_2$ iff $\mu \models \eta_1$ or $\mu \models \eta_2$

Given an algorithm S, a parametric propositional formulus η, and a bound C, the corresponding *safety bound problem* asks if there is a computation $\nu_0\nu_1\ldots\nu_k\ldots\ldots$ of an implementation of S such that $(\sum_{\mu\in U_A;\mu\models\eta}(\#_\mu(L_k)+\bar{L}_k(\mu)))^{\langle B\rangle} > C$ where ν_k contains a list recording $(L_k,\bar{L}_k)$.

Following is our framework for safety bound verification. Given a $\chi \in {X_A}^{\langle B\rangle}$, $\mathbf{count}_\eta(\chi)$ is the number, respecting bound B, of processes satisfying η recorded by PST counter χ. Formally speaking, $\mathbf{count}_\eta(\chi) = (\sum_{\mu\in U_A;\mu\models\eta}\chi(\mu))^{\langle B\rangle}$. Given an LR $\mathbf{PCL}^{\langle B\rangle}$-image $v = (\rho,\lambda)$ with $\rho = \chi_1\chi_2\ldots\chi_n$, let $\mathbf{count}_\eta(\rho) = (\sum_{1\le i\le n}\mathbf{count}_\eta(\chi_i))^{\langle B\rangle}$ and $\mathbf{count}_\eta(v) = (\mathbf{count}_\eta(\rho)+\mathbf{count}_\eta(\lambda))^{\langle B\rangle}$. Now we have the procedure $\mathbf{SafetyBound}()$ in table 1 to embody our verification method. There are two details to fill in. The first is the construction of PAP set A whose composition depends on target algorithms. The second detail to fill in is the definition of $\mathbf{arc}()$ such that for any two $\mathbf{PCL}^{\langle B\rangle}$-images v,v', $\mathbf{arc}(v,v')$ is true iff there is an algorithm implementaiton in which there are

```
/* $S$ is a system with identical local process description $P_p$.
/* $\eta$ is a proposition formulus describing the "critical section" property.
/* $C$ is the safety bound of processes allowed to satisfy $\eta$ at any state.
/* $B$ is the bound of PST counters. It is assumed $C \leq B$. */
**SafetyBound**$(S, \eta, C, B)$ {
  (1) Let $V :=$ the set of $\mathbf{PCL}^{\langle B \rangle}$-images of all initial states of all imple-
    mentations.
  (2) Copy $V$ to $W$.
  (3) Repeat while there is a $w \in W$, {
    (1) Delete $w$ from $W$.
    (2) Let $V' := \{w' \mid \mathbf{arc}(w, w')\} - V; V := V \cup V'; W := W \cup V'$;
    (3) connect from $w$ to $w'$ with an arrow.
    }
  (4) If there is no $v \in V$ such that $\mathbf{count}_\eta(v) > C$ and $v$ is reachable from
    an image of initial states, then report "YES;" else report "don't know."
}
```

Table 1. Our procedure of safety bound verification

two states ν, ν' such that $\mathbf{PCL}^{\langle B \rangle}(\nu) = v$, $\mathbf{PCL}^{\langle B \rangle}(\nu') = v'$, and a process in the implementation may directly go from ν to ν'. Clearly, **arc**() is dependent on algorithm description. In section 5, we shall demonstrate how to define PAP sets and **arc**() relations for single-queue systems. In section 6, we shall discuss the results of applying CQS technology to an algorithm for dense-time systems with a local timer per process, i.e. Fischer's timed mutual exclusion algorithm.

5 Untimed systems with a global queue

The class of systems in example 1 has an explicit queue. For conciseness of the presentation, we shall make the definition as simple as possible. The process algorithm is represented as a labeled transition systems (directed graph) such that the nodes are indexed by consective natural numbers from zero. On node i, $q_p = i$ is true while $q_p = j$ is false for every $j \neq i$. On the arcs which represents transitions, we may label transition rules like $\eta \rightarrow [\kappa]$ where (1) η represents the triggering condition and is a Boolean combination of $p = \mathbf{head}(Q)$; and (2) κ is either null, or "**wait**(p, Q);", or "**exit**(p, Q);".

The global state recording is exactly an LR with PAP set $\{q_p = 0, q_p = 1, \ldots, q_p = m\} \cup \{p = \text{head}(Q)\}$ where $m+1$ operation modes are in the algorithm description. Interleaving semantics is assumed. Thus a computation of such a system is exactly a sequence of LR's. The $\mathbf{PCL}^{\langle B \rangle}$-images of initial states are like $(\perp, \lambda_0)$ where $\perp$ is the empty list and λ_0 is a PST counter that maps the PST of initial process state into $\{0, 1, \ldots, B, \infty\}$ and everything else to zero.

We shall let $\mathbf{arc}(v, v') \equiv \bigvee_e \mathbf{xtion}_e(v, v')$, for any two $\mathbf{PCL}^{\langle B\rangle}$-images v, v', where $\bigvee_e$ quantifies over all transition rules e in the algorithm and $\mathbf{xtion}_e(v, v')$ is true iff there is an algorithm implementation in which there are two states ν, ν' such that $\mathbf{PCL}^{\langle B\rangle}(\nu) = v$, $\mathbf{PCL}^{\langle B\rangle}(\nu') = v'$, and a process in the implementation may transit ν to ν' with e.

Given a transition $e = \eta \to [\kappa]$, statement $s \in e$ iff $s \in \kappa$, i.e. s is an element statement in κ. Also $e(\mu)$ is the new PST obtained from PST μ by executing the action sequence in κ. Now we can formally define $\mathbf{xtion}_{\eta\to[\kappa]}(\rho, \lambda, \rho', \lambda')$ as in table 2. (Note we omit the left parentheses before ρ, ρ' and right parentheses after λ, λ' for clarity.) Formulus $\mathbf{normal}(\rho, \rho_1)$ on the top is used to make sure

$$\mathbf{xtion}_{\eta\to[\kappa]}(\rho, \lambda, \rho'\lambda') \equiv \exists\mu\exists\rho_1 \begin{pmatrix} \mu \models \eta \wedge \mathbf{normal}(\rho, \rho_1) \\ \wedge \begin{pmatrix} \mathbf{InList}^Q_\kappa(\mu, \rho_1, \lambda, \rho', \lambda') \\ \vee \mathbf{NotInList}^Q_\kappa(\mu, \rho_1, \lambda, \rho', \lambda') \end{pmatrix} \end{pmatrix}$$

$$\mathbf{InList}^Q_\kappa(\mu, \rho_1, \lambda, \rho', \lambda') \equiv \exists 1 \le i \le |\rho_1| \begin{pmatrix} \mathbf{element}(\rho_1, i)(\mu) \neq 0 \\ \wedge \begin{pmatrix} \mathbf{StayInList}_\kappa(\mu, i, \rho_1, \lambda, \rho', \lambda') \\ \vee \mathbf{LeaveList}_\kappa(\mu, i, \rho_1, \lambda, \rho', \lambda') \end{pmatrix} \end{pmatrix}$$

$$\mathbf{StayInList}_\kappa(\mu, i, \rho_1, \lambda, \rho', \lambda') \equiv \begin{pmatrix} \text{exit}(p, Q); \notin \kappa) \wedge \lambda = \lambda' \\ \wedge \exists\rho_2\exists\rho_3 \begin{pmatrix} \mathbf{inc}^{\langle B\rangle}(\rho_2, i, e(\mu), \rho_3) \\ \wedge \mathbf{dec}^{\langle B\rangle}(\rho_1, i, \mu, \rho_2) \\ \wedge \mathbf{normal}(\rho_3, \rho') \end{pmatrix} \end{pmatrix}$$

$$\mathbf{LeaveList}_\kappa(\mu, i, \rho_1, \lambda, \rho', \lambda') \equiv \begin{pmatrix} (\text{exit}(p, Q); \in \kappa) \wedge \exists\rho_4(\mathbf{dec}^{\langle B\rangle}(\rho_1, i, \mu, \rho_4) \\ \wedge \mathbf{normal}(\rho_4, \rho')) \wedge \mathbf{add}\mathit{1}^{\langle B\rangle}{}_{e(\mu)}(\lambda, \lambda') \end{pmatrix}$$

$$\mathbf{NotInList}^Q_\kappa(\mu, \rho_1, \lambda, \rho', \lambda') \equiv \lambda(\mu) \neq 0 \wedge \begin{pmatrix} \mathbf{StayOutOfList}_\kappa(\mu, \rho_1, \lambda, \rho', \lambda') \\ \vee \mathbf{JoinList}_\kappa(\mu, \rho_1, \lambda, \rho', \lambda') \end{pmatrix}$$

$$\mathbf{StayOutOfList}_\kappa(\mu, \rho_1, \lambda, \rho', \lambda') \equiv \begin{pmatrix} (\text{wait}(p, Q); \notin \kappa) \wedge \rho_1 = \rho' \\ \wedge \exists\lambda_1 \left(\mathbf{del}\mathit{1}^{\langle B\rangle}{}_\mu(\lambda, \lambda_1) \wedge \mathbf{add}\mathit{1}^{\langle B\rangle}{}_{e(\mu)}(\lambda_1, \lambda')\right) \end{pmatrix}$$

$$\mathbf{JoinList}_\kappa(\mu, \rho_1, \lambda, \rho', \lambda') \equiv \begin{pmatrix} (\text{wait}(p, Q); \in \kappa) \wedge \mathbf{del}\mathit{1}^{\langle B\rangle}{}_\mu(\lambda, \lambda') \\ \wedge \mathbf{normal}(\mathbf{addtail}(\rho_1, \mathbf{make}\mathit{1}_{e(\mu)}), \rho') \end{pmatrix}$$

Table 2. Formulation of transition relation for single-queue systems

we are dealing with a valid $\mathbf{PCL}^{\langle B\rangle}$-image. $\mathbf{InList}^Q_\kappa()$ handles the case when a process in the linear list makes the transition. $\mathbf{StayInList}_\kappa()$ handles the case when the transiting process stays in the linear list and we only have to change the PST of the transiting process in the same position in the list. $\mathbf{LeaveList}_\kappa()$ handles the case when the transiting process will leave the linear list and we then have to move the PST to λ.

$\mathbf{NotInList}^Q_\kappa()$ handles the case when a process not in the linear list makes the transition. $\mathbf{StayOutOfList}_\kappa()$ handles the case when the transiting process will not join the linear list and we only have to change the PST of the transiting process in λ. $\mathbf{JoinList}_\kappa()$ handles the case when the transiting process will join the linear list and we have to displace the PST from λ to the end of the list.

The following lemma shows that our method with CQS of $B = 1$ is complete in verifying mutual exclusion property of all such waiting queue systems.

Lemma 2. *For all waiting queue systems, as described in this section, with unknown number of processes with algorithm P_p, if there is a path $v_0 v_1 \ldots v_n$ in the CQS with $B = 1$ such that $v_0 \in I$ and v_n violates the mutual exclusion property, then there is indeed an implementation $\|_{1\leq p\leq m} P_p$ violating the mutual exclusion property.*

Proof : Please check appendix A. $\|$

In fact, the CQS of $B = 1$ constructed with our method for the algorithm in example 1 is very small as already shown in figure 2. Another interesting thing about our lemma proof is that the reasoning is exactly the same as a manual proof of the algorithm. This shows that our CQS technology indeed verifies algorithm systems at an abstractness roughly equivalent to that of human engineers.

6 Dense-time systems with one local timer per process

In this section, a CQS constructing algorithm for one-local-timer dense-time systems is presented. The implicit linear list happens, according to [1], as the linear ordering among the fractional parts of all timers' readings.

Here is our presentation plan. We shall first define formally the systems and proves its reachability problem, even with only a binary global variable, is still undecidable. Encoding of state recordings as LRs will then be given. We shall encode $\mathbf{arc}()$ for all $\mathbf{PCL}^{\langle B\rangle}$-images v, v' as $\mathbf{arc}(v, v') = \mathbf{time}(v, v') \vee \bigvee_e \mathbf{xtion}_e(v, v')$. $\mathbf{time}(v, v')$ is true if there are states ν, ν' with $\mathbf{PCL}^{\langle B\rangle}$-images v, v' respectively such that ν can go directly to ν' through time-passage. $\bigvee_e \mathbf{xtion}_e(v, v')$ has similar meaning as in section 5.

6.1 System definition and problem complexity

A process in our dense-time concurrent systems interacts with peer processes through read-write operations to *global variables*. In addition, each process has its own *local variables* and *timers* which no other processes can access. 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 special variable denoting the 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 Z_F^H be the set of all state predicates constructed from H and F.

Definition 3. : **PTMTS**
Process timed mode-transition system (PTMTS) is defined to describe behaviors of 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 $P_p = \langle x_p, Y, Y_p, \Phi_p, T_p \rangle$ with the following restrictions.

- x_p is the local timer. Y is the global variable set. Y_p is the local variable set with $q_p \in Y_p$ as a special variable to record the current *mode* of process p.
- Φ_p is a state predicate in $Z^{\{x_p\}}_{Y \cup Y_p}$ denoting the *invariance condition* of process p.
- T_p is the set of transition rules. A transition rule has the form: $(q_p = i \wedge \eta) \rightarrow [q_p := j; \kappa]$. Here η is a state predicate in $Z^{\{x_p\}}_{Y \cup (Y_p - \{q_p\})}$ 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 is in one of the following three forms: $y := c;$, $y := p;$, and $x_p := 0;$. (Semicolon is the statement terminator.) Again $y \in Y \cup (Y_p - \{q_p\})$, p is a process identifier, and c is a natural number.

Initially all variables contain zeros. We shall also assume that initially all timers have some distinct large readings greater than all the timing constants used in the system description. This is purely for the convenience of later proofs and complexity analysis. Our method can be easily modified to incorporate initial conditions with zero timer readings.

Processes in our systems 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, P_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 P_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 3. : For each process of Fischer's timed mutual exclusion protocol, we have a PTMTS $\langle x_p, Y, Y_p, \Phi_p, T_p \rangle$ with $Y = \{l\}$, $Y_p = \{q_p\}$, $\Phi_p \equiv q_p = 0 \vee (q_p = 1 \wedge 0 \leq x_p \wedge x_p \leq 1) \vee q_p = 2 \vee q_p = 3$, 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 3, we draw the PTMTS as a timed automaton[1] 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 Φ_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 identifier p to variable l, reset x_p to zero, and enter mode $q_p = 2$. ||

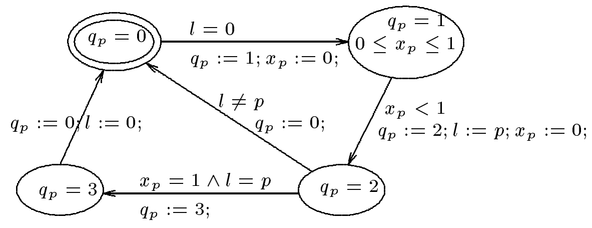

Fig. 3. Fischer's timed mutual exclusion protocol

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

Definition 4. : **States** Suppose we are given a real-time concurrent system implementation $S = \|_{1 \leq p \leq m} P_p$ with $P_p = \langle x_p, Y, Y_p, \Phi_p, T_p \rangle$. A *state* of S is a mapping ν from $\bigcup_{1 \leq p \leq m} \{x_p\} \cup Y \cup \bigcup_{1 \leq p \leq m} Y_p$ such that for each $1 \leq p \leq m$, $\nu(x_p) \in \mathcal{R}^+$; for each $y \in Y$, $\nu(y) \in \bigcup_{1 \leq p \leq m} D_{P_p:y}$; and for $1 \leq p \leq m$ and $y \in Y_p$, $\nu(y) \in D_{P_p:y}$. We let C_S be the largest timing constant (one used to compare with timers in state predicates) in S. A state ν is an *initial state* iff for all $y \in Y \cup \bigcup_{1 \leq p \leq m} Y_p$, $\nu(y) = 0$; and for all $1 \leq p < p' \leq m$, $C_S < \nu(x_p) \neq \nu(x_{p'}) > C_S$. $\|$

The *satisfaction* of a state predicate η by a state ν, in symbols $\nu \models \eta$, is defined in a traditional inductive way.

- $\nu \models y = c$ 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 $1 \leq p \leq m$, $(\nu + \delta)(x_p) = \nu(x_p) + \delta$. 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 5. : **runs** Suppose we have a real-time concurrent system implementation $S = \|_{1 \leq p \leq m} P_p$ with $P_p = \langle x_p, Y, Y_p, \Phi_p, T_p \rangle$ 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 $k \geq 0$,

- $t_k < t_{k+1}$;
- for all $t \in [0, t_{k+1} - t_k]$, $\nu_k + t \models \bigwedge_{1 \leq p \leq m} \Phi_p$; 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

$(\nu_k + (t_{k+1} - t_k))\kappa = \nu_{k+1}$. ||

We shall show that even with severe restriction on dense-time systems with one local timer per process, the local state reachability problem of protocols is still undecidable. A dense-time concurrent system is *basic* iff its processes are identically described by PTMTS P_p and it has only one global variable l, and the only operations allowed on l by a process p are

- test on $l = 0$, $l = p$, and their negations; and
- assignments like $l := 0$; and $l := p$;.

Note Fischer's timed mutual exclusion protocol falls exactly in the class. The following lemma shows that even for protocol in this basic class, the local state reachability problem is still undecidable.

Lemma 6. *: Two-counter machine halting problem is reducible to the local state reachability problem of basic dense-time concurrent systems.*

Proof : Our reduction shall treat the sole global variable l as a bus and interpret the values of l along the time line with the common technology of time-division multiplexing in telecommunication industry. Due to page-limit, we shall leave the definition of two-counter machines and proof details to appendix B. ||

Our reduction actually only uses triggering conditions like $l = 0, l \neq 0$. This means that safety bound problem of dense-time protocols is undecidable even restricted with one timer per processes, one "binary" global variable, and no global timers.

6.2 Encoding for regions

Given a state ν in $S = \|_{1 \leq p \leq m} P_p$ with $P_p = \langle x_p, Y, Y_p, \Phi_p, T_p \rangle$, its *region recording* $\mathbf{region}(\nu)$ is a triple (α, β, z) such that

- α is a sequence $p_1 p_2 \dots p_n$ such that
  - p_j precedes p_k in α iff $\nu(x_{p_j}) - \lfloor \nu(x_{p_j}) \rfloor < \nu(x_{p_k}) - \lfloor \nu(x_{p_k}) \rfloor$.
  - $\{p_1, \dots, p_n\} = \{p \mid \nu(x_p) \leq C_S; 1 \leq p \leq m\}$
- β maps variables to their domain values and timers to their integer values within $[0, C_S] \cup \{\infty\}$. More precisely, for all $1 \leq p \leq m$,
  - $\beta(x_p) = \lfloor \nu(x_p) \rfloor$ if $\nu(x_p) \leq C_S$; $\beta(x_p) = \infty$ otherwise;
  - for all $y \in Y \cup Y_p$, $\beta(y) = \nu(y)$;
- z is a Boolean value which is true iff timer x_{p_1} is at an integer reading.

According to [1], two states make no difference for a model-checking problem instance iff they have the same region recording.

A node in our CQS for dense-time systems shall be a pair (Π, z) where Π is the $\mathbf{PCL}^{\langle B \rangle}$-image of an LR. Now we shall define the mapping from region recordings to LR's. The PST classification of processes needs PAP set A which is

$$\begin{aligned} & \{\lfloor x_p \rfloor = c \mid 0 \leq c \leq C_S\} \\ \cup\ & \{x_p > C_S\} \\ \cup\ & \left\{y = d \mid y \in Y \cup Y_p; d \in D_{P_p:y} \cup \{p\}\right\} \\ \cup\ & \left\{p = d \mid d \in D_{P_p:y}\right\} \end{aligned}$$

The first three sets are kind of direct translation from region recordings. The last set $\{p = d \mid d \in D_{P_p:y}\}$ is needed because we have omitted the process identifiers and we still have to take care when special process identifier constants are assigned to variables. This is important when the system is not symmetric to all processes and some process's identifier constants can be handled by other processes. For example, a centralized system may have a process sets global lock to 1 when it detects a dangerous condition.

Example 4. : For Fischer's mutual exclusion protocol system in example 3, $C_S = 1$ and PAP set $A = \left\{ \begin{array}{l} \lfloor x_p \rfloor = 0, \lfloor x_p \rfloor = 1, \lfloor x_p \rfloor > 1, q_p = 0, \\ q_p = 1, q_p = 2, q_p = 3, l = 0, l = p, p = 0 \end{array} \right\}$. Now suppose we have a state ν in a system with four processes such that $\nu(x_1) = 0.3, \nu(x_2) = 2.5, \nu(x_3) = 0, \nu(x_4) = 0.7, \nu(q_1) = 0, \nu(q_2) = 2, \nu(q_3) = 1, \nu(q_4) = 1$, and $l = 4$. Then $\mathbf{region}(\nu) = (\mu_{(1)}\mu_{(2)}\mu_{(3)}, (\mu_{(1)} : 0, \mu_{(2)} : 0, \mu_{(3)} : 0, \mu_{(4)} : 1), \mathit{true})$ with $\mu_{(1)} = \{\lfloor x_p \rfloor = 0, q_p = 1\}, \mu_{(2)} = \{\lfloor x_p \rfloor = 0, q_p = 0\}, \mu_{(3)} = \{\lfloor x_p \rfloor = 0, q_p = 1, l = p\}, \mu_{(4)} = \{x_p > 1, q_p = 2\}$. Note here $\mu_{(1)}, \mu_{(2)}, \mu_{(3)}, \mu_{(4)}$ represent respectively processes $3, 1, 4, 2$. $\|$

The $\mathbf{PCL}^{\langle B \rangle}$-images of initial states are like $(\perp, \lambda_0, \mathit{false})$ and can be explained the same as at the beginning of section 5 for untimed systems with a waiting queue.

With the PAP set A properly defined, we can then use β to construct U_A (the set of PSTs) and $X_A{}^{\langle B \rangle}$ (the set of PST counters). Then the linear list L can be readily constructed directly from α by replacing process identifiers with their PST's. The component $\bar{L}$ can be viewed as the multiset of PST's of those processes whose local timer readings are greater than C_S. In the next two subsections, With L and $\bar{L}$ ready, we can then enumerate the $\mathbf{PCL}^{\langle B \rangle}$-images of all states and construct arcs among the nodes in the CQS.

6.3 Time passage

We now define relation $\mathbf{time}()$ in table 3 such that $\mathbf{time}(\rho, \lambda, z, \rho', \lambda', z')$ iff (ρ, λ, z) can go to (ρ', λ', z') by a single time-progression. Here is an explanation of the formulae. Formulus $\mathbf{AnIntegerTimer}()$ handles the case that one of the timers has a reading at an integer no greater than C_S. In this case, that timer will advance its reading to a non-integer value. When that timer's reading is C_S, then at the next $\mathbf{PCL}^{\langle B \rangle}$-image ρ', the process of that timer is removed from the linear list and recorded in λ. When that timer's reading is less than C_S, then the $\mathbf{PCL}^{\langle B \rangle}$-image (and λ) of linear list will not change.

Formulus $\mathbf{NoIntegerTimer}()$ handles the case that none of the timers is at an integer reading no greater than C_S. In this case, the timer with the biggest fractional part in its reading will advance to an integer value. The new list of PST counters $\mathbf{addhead}(\mathbf{front}(\rho_1), \mathbf{make}\mathit{1}_{(\mu - \{\lfloor x_p \rfloor = c\}) \cup \{\lfloor x_p \rfloor = c+1\}})$ is the result after making such an advancement.

6.4 Transitions

$\mathbf{xtion}_e(\rho, \lambda, z, \rho', \lambda', z')$ is formally defined in table 4. Formulus $\mathbf{InList}_e^D()$ han-

$$\mathbf{time}(\rho,\lambda,z,\rho',\lambda',z') \equiv \exists\rho_1 \left(\begin{array}{c}\mathbf{normal}(\rho,\rho_1)\\ \wedge\left(\begin{array}{c}\mathbf{AnIntegerTimer}(\rho_1,\lambda,z,\rho',\lambda',z')\\ \vee\mathbf{NoIntegerTimer}(\rho_1,\lambda,z,\rho',\lambda',z')\end{array}\right)\end{array}\right)$$

$$\mathbf{AnIntegerTimer}(\rho_1,\lambda,z,\rho',\lambda',z') \equiv \exists\mu \left(\begin{array}{c} z\wedge\neg z'\wedge\mathbf{head}(\rho_1)(\mu)=1\\ \wedge\left(\begin{array}{c}\mathbf{CrossC}_S(\rho_1,\lambda,\rho',\lambda')\\ \vee\mathbf{StayInC}_S(\rho_1,\lambda,\rho',\lambda')\end{array}\right)\end{array}\right)$$

$$\mathbf{CrossC}_S(\rho_1,\lambda,\rho',\lambda') \equiv \mu\models\lfloor x_p\rfloor=C_S \rightarrow \left(\begin{array}{c}\mathbf{normal}(\mathbf{tail}(\rho_1),\rho')\\ \wedge\mathbf{add}\mathit{1}^{\langle B\rangle}_{(\mu-\{\lfloor x_p\rfloor=C_S\})\cup\{x_p>C_S\}}(\lambda,\lambda')\end{array}\right)$$

$$\mathbf{StayInC}_S(\rho_1,\lambda,\rho',\lambda') \equiv \mu\not\models\lfloor x_p\rfloor=C_S \rightarrow (\rho_1=\rho'\wedge\lambda=\lambda')$$

$$\mathbf{NoIntegerTimer}(\rho_1,\lambda,z,\rho',\lambda',z') \equiv \exists\mu \left(\begin{array}{c}\neg z\wedge z'\wedge\mathbf{last}(\rho_1)(\mu)=1\wedge\lambda=\lambda'\\ \wedge\mathbf{AdvanceToInteger}(\rho_1,\rho')\end{array}\right)$$

$$\begin{array}{l}\mathbf{AdvanceToInteger}(\rho,\rho')\\ \quad\equiv\exists 0\le c<C_S\\ \quad\left(\begin{array}{l}\mu\models\lfloor x_p\rfloor=c\\ \wedge\mathbf{normal}\left(\mathbf{addhead}\left(\mathbf{front}(\rho_1),\mathbf{make}\mathit{1}\left(\begin{array}{c}(\mu-\{\lfloor x_p\rfloor=c\})\\ \cup\{\lfloor x_p\rfloor=c+1\}\end{array}\right)\right),\rho'\right)\end{array}\right)\end{array}$$

Table 3. Formulation of time-passages for dense-time systems in section 6

dles the case when a process in the linear list makes the transition. $\mathbf{NoReset}^I_e()$ handles the case when the transition does not reset the local timer to zero and we only have to change the PST of the transiting process in the same position in the list. $\mathbf{AReset}^I_e()$ handles the case when the transition resets the local timer to zero and we then have to move the process to the beginning of the list.

Formulus $\mathbf{NotInList}^D_e()$ handles the case when a process not in the linear list makes the transition. The nested disjunct starting with $\mathbf{NoReset}^{NI}_e()$ and $\mathbf{AReset}^{NI}_e()$ respectively correspond to $\mathbf{NoReset}^I_e()$ and $\mathbf{AReset}^I_e()$ in last paragraph.

6.5 On Fischer's timed mutual exclusion protocol

According to example 3, Fischer's timed mutual exclusion protocol falls in our class of dense-time concurrent systems with one local timer per process. We have the following lemma to prove that with $B = 1$, our method of CQS can prove that Fischer's protocol maintains mutual exclusion for system implementations with all numbers of processes.

Lemma 7. *: In the CQS constructed using our method with bound $B = 1$, no state $\mathbf{PCL}^{\langle 1\rangle}$-image v with $\mathbf{count}_{q_p=3}(v) > 1$ is reachable from a initial state $\mathbf{PCL}^{\langle 1\rangle}$-image.*
Proof : Due to page-limit, we leave the proof in appendix C. ||

$$\mathbf{xtion}_e(\rho,\lambda,z,\rho',\lambda',z') \equiv \exists\mu\exists\rho_1 \begin{pmatrix} \mu \models \eta \wedge \mathbf{normal}(\rho,\rho_1) \\ \wedge\begin{pmatrix} \mathbf{InList}^D_e(\mu,\rho_1,\lambda,z,\rho',\lambda',z') \\ \vee\mathbf{NotInList}^D_e(\mu,\rho_1,\lambda,z,\rho',\lambda',z') \end{pmatrix} \end{pmatrix}$$

$$\mathbf{InList}^D_e(\mu,\rho_1,\lambda,z,\rho',\lambda',z') \equiv \exists 1 \le i \le |\rho_1| \begin{pmatrix} \mathbf{element}(\rho_1,i)(\mu) \neq 0 \wedge \lambda = \lambda' \\ \wedge\begin{pmatrix} \mathbf{NoReset}^I_e(i,\rho_1,z,\rho',z') \\ \vee\mathbf{AReset}^I_e(i,\rho_1,z,\rho',z') \end{pmatrix} \end{pmatrix}$$

$$\mathbf{NoReset}^I_e(i,\rho_1,z,\rho',z') \equiv \begin{pmatrix} (x_p := 0; \notin e) \wedge (z \to \neg z') \\ \wedge\exists\rho_2\exists\rho_3 \begin{pmatrix} \mathbf{inc}^{\langle B\rangle}(\rho_2,i,e(\mu),\rho_3) \\ \wedge\mathbf{dec}^{\langle B\rangle}(\rho_1,i,\mu,\rho_2) \\ \wedge\mathbf{normal}(\rho_3,\rho') \end{pmatrix} \end{pmatrix}$$

$$\mathbf{AReset}^I_e(i,\rho_1,z,\rho',z') \equiv \begin{pmatrix} (x_p := 0; \in e) \wedge z' \\ \wedge\exists\rho_4 \begin{pmatrix} \mathbf{dec}^{\langle B\rangle}(\rho_1,i,\mu,\rho_4) \\ \wedge\mathbf{normal}(\mathbf{addhead}(\rho_4,\mathbf{make}\mathit{1}_{e(\mu)}),\rho') \end{pmatrix} \end{pmatrix}$$

$$\mathbf{NotInList}^D_e(\mu,\rho_1,\rho') \equiv \begin{pmatrix} \lambda(\mu) \neq 0 \\ \wedge\begin{pmatrix} \mathbf{NoReset}^{NI}_e(\rho_1,\lambda,z,\rho',\lambda',z') \\ \vee\mathbf{AReset}^{NI}_e(\rho_1,\lambda,z,\rho',\lambda',z') \end{pmatrix} \end{pmatrix}$$

$$\mathbf{NoReset}^{NI}_e(\rho_1,\lambda,z,\rho',\lambda',z') \equiv \begin{pmatrix} (x_p := 0; \notin e) \wedge (z \to \neg z') \wedge \rho_1 = \rho' \\ \wedge\exists\lambda_1 \begin{pmatrix} \mathbf{del}\mathit{1}^{\langle B\rangle}{}_{\mu}(\lambda,\lambda_1) \\ \wedge\mathbf{add}\mathit{1}^{\langle B\rangle}{}_{e(\mu)}(\lambda_1,\lambda') \end{pmatrix} \end{pmatrix}$$

$$\mathbf{AReset}^{NI}_e(\rho_1,\lambda,z,\rho',\lambda',z') \equiv \begin{pmatrix} (x_p := 0; \in e) \wedge z' \wedge \mathbf{del}\mathit{1}^{\langle B\rangle}{}_{\mu}(\lambda,\lambda') \\ \wedge\mathbf{normal}(\mathbf{addhead}(\rho_1,\mathbf{make}\mathit{1}_{e(\mu)}),\rho') \end{pmatrix}$$

Table 4. Formulation of transitions for dense-time systems in section 6

Also for the simple protocol used in [13], our method can also be proved to work.

We want to point out in our lemma proof, the reasoning is exactly the same as a manual proof of the protocol. This shows that our CQS technology indeed verifies protocol systems at an abstractness roughly equivalent to that of human engineers.

We proceed to analyze the size of CQS for Fischer's protocol to show that our method is very efficient for systems designed with engineering rules. We have nine PSTs.

$$\begin{array}{ll} \mu_1 = \{q_p = 0; x_p > 1\} & \mu_6 = \{q_p = 2; \lfloor x_p \rfloor = 0; l = p\} \\ \mu_2 = \{q_p = 1; \lfloor x_p \rfloor = 0\} & \mu_7 = \{q_p = 2; \lfloor x_p \rfloor = 1; l = p\} \\ \mu_3 = \{q_p = 2; \lfloor x_p \rfloor = 0\} & \mu_8 = \{q_p = 2; x_p > 1; l = p\} \\ \mu_4 = \{q_p = 2; \lfloor x_p \rfloor = 1\} & \mu_9 = \{q_p = 3; x_p > 1; l = p\} \\ \mu_5 = \{q_p = 2; x_p > 1\} & \end{array}$$

We observed the following interaction patterns among the processes with different PSTs in a single $\mathbf{PCL}^{\langle 1\rangle}$-image of any state. Only PSTs $\mu_2, \mu_3, \mu_4, \mu_6, \mu_7$ join the formation of the linear list. And processes of PSTs μ_3, μ_4, μ_6, or μ_7 always precede processes of PST μ_2. With $C_S = 1$ and our interleaving semantics, there will only be one process in the linear list of either PST μ_4 or PST μ_7. There is at most one process of PSTs satisfying $l = p$. Also processes of PSTs μ_4, μ_6, μ_7 are always at the beginning of the linear list.

With all such restrictions, we find the linear list must have the following pattern:

$$\underbrace{\dot{\mu}}_{\rho_{(1)}}\underbrace{\mu_3 \ldots \mu_3}_{\rho_{(2)}}\underbrace{\mu_2 \ldots \mu_2}_{\rho_{(3)}}$$

where $\dot{\mu}$ is one of μ_4, μ_6, μ_7. We underlabel the three segments of the linear list with $\rho_{(1)}, \rho_{(2)}, \rho_{(3)}$ for convenience of discussion. The number of different possibilities for the $\mathbf{PCL}^{\langle 1 \rangle}$-images of segment $\rho_{(2)}$ (and hence $\rho_{(3)}$) is 3. Thus the number of different **PCL**-images for the linear list is no more than $1 + (3 \times (3 \times 3)) = 28$ where the beginning 1 denotes the null list case.

As for $\bar{L}$, only processes of PST $\mu_1, \mu_5, \mu_8, \mu_9$ can join it. At any $\mathbf{PCL}^{\langle 1 \rangle}$-image, there can be at most one process in PST μ_8 or μ_9. Processes in PST μ_8, μ_9 cannot coexist. Thus the number of different possibilities of $\mathbf{PCL}^{\langle 1 \rangle}$-images of $\bar{L}$ is at most $3^2 \times 3 = 27$. Finally, z has two values. Putting all numbers together, we find that the total number of $\mathbf{PCL}^{\langle 1 \rangle}$-images for all reachable states is at most $28 \times 27 \times 2 = 1512$.

7 Conclusion

With the known huge complexities of most verification problems in theory[3, 4, 5, 19], it is apparent that the current technology of model-checking is incapable of verifying nontrivial systems. We believe such a dilemma results from the fact that current verification technology[1, 9, 10, 13, 19, 20] does not distinguish "good" designs from "bad" designs. Our **PCL**-image technology is a successful example to verify well-designed concurrent systems in which relations among processes in different PSTs are more important than the actual numbers of processes in each PST. We believe our technology can be extended to verify concurrent systems with multiple linear lists and multiple local timers per process.

Acknowledgment

The author thanks Dr. Tyng-Ruey Chuang, Dr. Pao-Ann Hsiung, Dr. Churn-Jung Liau, Dr. Da-Wei Wang, and Dr. Hsu-Chun Yen for their comments and suggestions on the first draft of the manuscript.

References

1. R. Alur, C. Courcoubetis, D.L. Dill. Model Checking for Real-Time Systems, IEEE LICS, 1990.
2. P.A. Abdulla, K. Cerans, B. Jonsson, Y.-K. Tsay. General Decidability Theorems for Infinite-State Systems. In proceedings of the IEEE LICS, 1996.

3. R. Alur, T.A. Henzinger. A Really Temporal Logic. In proceedings of the 30th IEEE FOCS, 1989.
4. R. Alur, T.A. Henzinger. Real-Time Logics : Complexity and Expressiveness. Information and Computation **104**, 35-77 (1993).
5. K.R. Apt, D.C. Kozen. Limits for Automatic Verification on finite-state concurrent systems. Information Processing Letters, 22:307-309, 1986.
6. F. Balarin. Approximate Reachability Analysis of Timed Automata. IEEE RTSS, 1996.
7. M.C. Browne, E.M. Clarke, O. Grumberg. Reasoning about Networks with Many Identical Finite State Processes. Information and Computation **81**, 13-31, 1989.
8. B. Boigelot, P. Godefroid. Symbolic Verification of Communication Protocols with Infinite State Spaces using QDDs. CAV 1996, LNCS, Springer-Verlag.
9. 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.
10. E.M. 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.
11. E.M. Clarke, O. Grumberg, S. Jha. Verifying Parameterized Networks using Abstraction and Regular Languages. CONCUR'95, LNCS 962, Springer-Verlag.
12. E.A. Emerson, K.S. Namjoshi. Reasoning about Rings. ACM POPL, 1995.
13. 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.
14. J.E. Hopcroft, J.D. Ullman. Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 1979.
15. R.P. Kurshan, K.L. McMillan. A Structural Induction Theorem for Processes. Information and Computation **117**, 1-11(1995).
16. D. Lesens, N. Halbwachs, P. Raymond. Automatic Verification of Parameterized Linear Networks of Processes. ACM POPL, 1997.
17. J.M. Mellor-Crummey, M.L. Scott. Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors. ACM Transactions on Computer Systems, Vol. 9, No. 1, Feb., 1991, pp. 21-65.
18. G.L. Peterson. A New Solution to Lamport's concurrent programming problem using small shared variables. ACM TOPLAS **5**, 1 (Jan. 1983), 56-65.
19. 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.
20. F. Wang. Timing Behavior Analysis for Real-Time Systems. IEEE LICS 1995.
21. F. Wang. Automatic Verification of Dynamic Linear Lists for All Numbers of Processes. Technical Report TR-IIS-98-019, IIS, Academia Sinica.
22. F. Wang, P.-A. Hsiung. Iterative Refinement and Condensation for State-graph Construction. Technical Report TR-IIS-98-009, IIS, Academia Sinica.

APPENDICES

A Completeness for basic queue systems

lemma 2 : *For all basic queue systems with many identical processes with program P_p, if there is a path $v_0 v_1 \dots v_n$ in the CQS with $B = 1$ such that $v_0 \in I$ and v_n violates the mutual exclusion property, then there is indeed a $\|_{1 \le p \le m} P_p$ with $m \in \mathcal{N}$ such that there is a computation from initial state to a state violating the mutual exclusion property.*

Proof : We shall construct from $v_0 v_1 \dots v_n$ a computation for some $\|_{1 \le p \le m} P_p$. There are two types of information we need to fill in. The first is the ordering among processes recorded by a PST counter while the second is the PST counts which exceed B and are recorded as ∞. This can done in several steps. We first transform $v_0 v_1 \dots v_n$ to $v_0^{(1)} v_1^{(1)} \dots v_n^{(1)}$ such that $v_0 = v_0^{(1)}$ and for all $1 \le i \le n$ with $v_i^{(1)} = (\rho_i^{(1)}, \lambda_i^{(1)})$, there is no PST counter χ in $\rho_i^{(1)}$ and two distinct μ, μ' with $\chi(\mu) \neq 0 \wedge \chi(\mu') \neq 0$. This can be done by working on v_i iteratively from $i = 1$ to n. Any PST counters in $\rho_i^{(1)}$ which were merged from a sublist of PST counters, generated from $\rho_{i-1}^{(1)}$ with transitions which changes $\rho_{i-1}^{(1)}$ to $\rho_i^{(1)}$, shall be replaced by that sublist. After such replacements have been done sequentially from v_1 to v_n, we are sure that for all $0 \le i \le n$, $\rho_i^{(1)}$ is a sequence of marker counters with bound B.

We can now visualize $v_0^{(1)} v_1^{(1)} \dots v_n^{(1)}$ as a sequence for the recording of PST count changes. A PST counter χ' in $v_{i+1}^{(1)}$ may be now identified as representing the result of another PST counter χ in $v_i^{(1)}$ after the transition from $v_i^{(1)}$ to $v_{i+1}^{(1)}$. Such χ and χ' can be identified as the snapshot of the same PST counting device at different states. Now we can check if the numbers of processes decremented and incremented in a PST counting device are consistent along $v_0^{(1)} v_1^{(1)} \dots v_n^{(1)}$. Suppose we find that from $v_i^{(1)}$ to $v_{i+1}^{(1)}$, a decrement needs to happen while there is not enough number of corresponding increments from $v_0^{(1)}$ to $v_i^{(1)}$. Since $B = 1$, we know that the counting device must have once recorded a count > 1 from $v_0^{(1)}$ to $v_i^{(1)}$. We can then trace the transition sequence back to a process p, in initial PST μ_0, transiting immediately from $v_j^{(1)}$, $0 \le j \le i$, to a PST other than μ_0. Then we shall insert a new process p' which follows the same transiting sequence as that of p till $v_i^{(1)}$. The transitions from p' shall interleave with those in $v_0^{(1)} \dots v_i^{(1)}$ and take places in an earliest way but no earlier than the corresponding ones of p's. This is possible because the only condition that can externally withhold a process from making a transition is $p = \text{head}(Q)$. By inserting transitions for p' in an earliest way but no earlier than the corresponding ones of p's, we can make sure p' will always immediately follow p in the waiting queue.

Suppose we find that from $v_i^{(1)}$ to $v_{i+1}^{(1)}$, a decrement shall happen but the PST counting devices with bound B indicates that not enough number of increments have been consumed by corresponding decrements. Then what we shall do is to insert another decrement for the troubling PST counting devices immediately before $v_i^{(1)}$. For transitions without triggering condition $p = \mathrm{head}(Q)$, this is OK because we can assume the transiting process exists and can autonomously make the transition. When the triggering condition is $p = \mathrm{head}(Q)$, there will be only one process recorded by the corresponding PST counting device and the trouble should not occur at all.

Assume that after the insertion of all the transitions, we get a new path $\Delta = v_0^{(2)} v_1^{(2)} \ldots v_k^{(2)}$. Since the numbers of increments and decrements of each PST counting device are consistent now and each PST counter in $v_0^{(2)}, v_1^{(2)}, \ldots, v_k^{(2)}$ is a marking counter, Δ can be straightforwardly translated into a computation with number of local processes equal to the maximum number of increments applied to a PST counting device along Δ. $\|$

B Undecidability of basic dense-time system mutual exclusion verification problem

We shall first briefly define 2-counter machines and then present our reduction. A counter can hold a nonnegative integer value. A 2-counter machine has a finite-state control, can increment, decrement a counter value by one, and can test if a counter value is zero. The halting problem of 2-counter machine is to answer whether the finite-state control can reach its final state. It is known that 2-counter halting problem is undecidable. [14]

lemma 6 : *Two-counter machine halting problem is reducible to the mutual exclusion verification of basic dense-time systems.*

Proof : The automaton representation of a process has the structure in figure 4 in which the automaton is partition into three parts. Only one process can enter

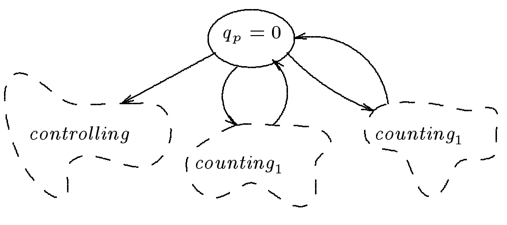

Fig. 4. A process for 2-counter machine reduction

part *controlling* while the numbers of processes in parts *counting*$_1$ and *counting*$_2$ respectively represents the contents of the two counters. The unique process which enters part *controlling* is identified by *control* and controls all other processes. The first transiting process in computation will test for a period of $l = 0$ for 12 time units to enter part *controlling* and becomes process *control*. Once process control is established, it will make sure $l \neq 0$ for one time units with the period of 12 time units. (Process *control* makes $l \neq 0$ true by writing *control* into l.) And all processes in parts *counting*$_1$ and *counting*$_2$ or to enter those two parts will listen to this periodicity. By our interleavling semantics with strict time progressing for each transition, the mechanism can be enforced by placing the path in figure 5 to guard entrance to part *control* and guarantees that only

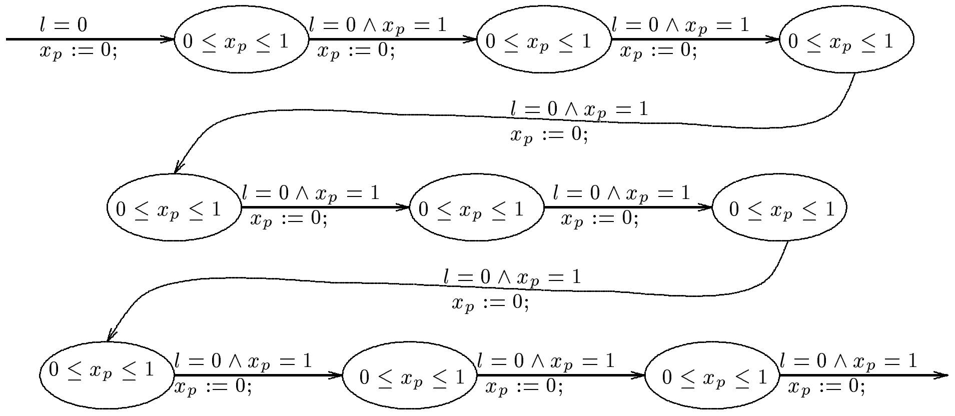

Fig. 5. Guarding path for entrance to part *control*

one process will be in part *controlling*.

Part *controlling* encodes the finite-state control of the target two-counter machine and communicates with all other processes by changing values of l along time line. It divides the time line into *cycles* of 12 time units. We call each time units a *slot*. A slot through which $l \neq 0$ ($l = 0$) is maintained is called a one-slot (zero-slot). Each cycle starts with a one-slot followed by four zero-slots. All processes except process *control* will observe this slot patterns for the start of a cycle. After the starting six slots of each cycle, process *control* will send one of the following six *instruction* slot patterns: 11000 (for testing if counter 1 equals zero), 11001 (for testing if counter 2 equals zero), 11100 (for incrementing counter 1 by one), 11101 (for incrementing counter 2 by one), 11110 (for decrementing counter 1 by one), and 11111 (for decrementing counter 2 by one). After the instruction slot-patterns, process *control* will set l to zero and wait for responses from other processes.

Processes in the initial mode, in part 1, and in part 2 will all observe the five time slots as instructions. When the instruction is 11000, all process in part *controlling*, if any, will set l to their identifiers interleavingly in the next time slot. Process *control* can then test if $l = 0$ to check if counter 1 contains zero.

When the instruction is 11100, a process in the initial mode will set l to its identifer in the next time units with triggering condition $l = 0$. Thus only one process will enter part counting$_1$ and corresponds to the "increment by one." However, if process *control* did not detect $l \neq 0$ in the twelveth time slot of the cycle, then it means there are not enough number of processes to emulate counter-increments and the emulation is unsuccessful.

When instruction is 11001, 11101, 11110, or 11111, the reduction can be done in similar ways. With more details filled in, we can then prove that the reduction is good and the mutual exclusion verification problem of basic dense-time protocol is indeed undecidable. $\|$

C Verification of Fischer's protocol

lemma 7 : *In the CQS constructed using our method with bound $B = 1$, no state $\mathbf{PCL}^{\langle 1 \rangle}$-image v with $\mathbf{count}_{q_p=3}(v) > 1$ is reachable from an initial state $\mathbf{PCL}^{\langle 1 \rangle}$-image.*

Proof : We assume there is a shortest path from an initial state $\mathbf{PCL}^{\langle 1 \rangle}$-image to another state $\mathbf{PCL}$-image v with $\mathbf{count}_{q_p=3}(v) = 2$. Now in $\mathbf{PCL}^{\langle 1 \rangle}$-image v, there are two processes satisfying $q_p = 3$. Conveniently let P_a and P_b be respectively the first and the second process getting into mode $q_p = 3$ along the path. Since our $B = 1$, representing processes with corresponding PSTs will evolve for P_a and P_b respectively along the path with corresponding transiting arcs. This can be checked because our $\mathbf{time}()$ and $\mathbf{xtion}_e()$ relations never reduce the process count for a PST from one to zero in the $\mathbf{PCL}^{\langle 1 \rangle}$-image v' unless a corresponding process is transformed to another PST. Notationally, we use $P_\alpha^{c \rightarrow d}$ to denote the transitting arc for process P_α transiting from mode $q_\alpha = c$ to mode $q_\alpha = d$ along the path.

Right after transitting arc $P_a^{2 \rightarrow 3}$, $l = a$ is maintained along the path until some other process overwrites it. Since P_a is the first process entering the critical sectioin along the path, we know that there must be a first overwriting along the path from some process P_c with transitting arc $P_c^{1 \rightarrow 2}$ and statement $l := c$; along the path. Since P_c makes the first overwriting, we know transitting arc $P_c^{0 \rightarrow 1}$ happens, on condition $l = 0$, before transitting arc $P_a^{2 \rightarrow 3}$; otherwise a third process will have to overwrite $l = a$ with $l := 0$ before P_c's overwriting.

Due to the linear sequence maintained in our $\mathbf{PCL}^{\langle 1 \rangle}$-image for the timer ordering, we further infer that the transitting arc $P_c^{0 \rightarrow 1}$ must occur between transitting arcs $P_a^{1 \rightarrow 2}$ and $P_a^{2 \rightarrow 3}$ in that order. For $P_a^{2 \rightarrow 3}$ to happen, in between the two transitting arcs $P_a^{1 \rightarrow 2}$ and $P_a^{2 \rightarrow 3}$, $l = a$ must be maintained. But this is impossible because $P_c^{0 \rightarrow 1}$ necessarily observes $l = 0$ and no process will change l's value back to a because P_c does the first overwriting by assumption. This contradicts our assumption.

By checking the relation between the above-mentioned transitting arcs and their recordings in $\mathbf{PCL}^{\langle 1 \rangle}$-images of states, we can then prove that such an transitting arc sequence cannot happen along any path in our CQS. $\|$