APPLICATIONS OF CONFLICT-FREE PETRI NETS TO PARALLEL PROGRAMS AND ASYNCHRONOUS CIRCUITS*

HSU-CHUN YEN

Dept. of Electrical Engineering, National Taiwan University
Taipei, Taiwan 10764, R.O.C.

ABSTRACT We propose a unified approach for dealing with the *race detection problem* for two entirely different models, namely, parallel programs and asynchronous circuits. We first show that the problem of determining whether two transitions in a 1-bounded conflict-free Petri net can become enabled simultaneously is solvable in polynomial time. (This will be referred to as the *pairwise concurrency problem*.) We then show that the race detection problem for parallel programs (asynchronous circuits) and the pairwise concurrency problem for Petri nets are closely related to each other. As a result, race conditions can be detected efficiently (i.e., in polynomial time) for those parallel programs and asynchronous circuits that can be modeled by 1-bounded conflict-free Petri nets. Since most problems concerning Petri nets are very difficult to solve, our polynomial time result is of interest and significance in its own right.

1 Introduction

Petri nets provide an elegant and useful tool for modeling concurrent systems. In many applications, however, modeling by itself is of little practical use if one cannot analyze the modeled system. Unfortunately, Petri nets remain one of the least understood models, despite the broad attention they have been receiving over the past two decades. In fact, most Petri net problems are extremely difficult to solve, if not impossible. For example, for general Petri nets, the boundedness problem is EXPSPACE (exponential space)-complete [13,18], the reachability problem is decidable [11,15] (although the only known algorithm is not even primitive recursive) and is EXPSPACE-hard [13], and the containment and equivalence problems are undecidable [1].

*This work was supported in part by the National Science Council of the Republic of China under Grant No. NSC 80-0408-E-002-16.

A natural approach to ease the high degree of difficulty associated with Petri net analysis is to restrict our attention to 'restricted' classes of Petri nets, hoping for such restricted ones to have easier solutions. Of all the restricted classes of Petri nets studied in the literature, most notable is that of conflict-free Petri nets [5,6,7,8,10,12]. In particular, the boundedness problem has been shown to be PTIME (polynomial time)-complete [8], the reachability problem has been proved to be NP (nondeterministic polynomial time)-complete [6], whereas the containment and equivalence problems have been shown to be Π_2^p (the second level of the polynomial time hierarchy) complete [6]. If we further restrict ourselves to bounded conflict-free Petri nets, the upper bound of reachability can be improved to PTIME [7].

In a recent article [9], it was indicated that the expressive power of conflict-free Petri nets is somewhat limited. Specifically, it was show that conflict-free Petri nets cannot be used to model the well-known mutual exclusion problem. In view of that, a natural question arises: **How useful are conflict-free Petri nets in modeling real-world problems?** In this paper, we will provide evidence to demonstrate that, despite their limited expressive power, conflict-free Petri nets can still be very useful in modeling and analyzing real-world problems. To that end, we first define the so-called *pairwise concurrency problem* for Petri nets. The pairwise concurrency problem is that of, given a Petri net P and two transitions t and t', determining where there exists a reachable marking in P in which t and t' are enabled simultaneously. We then prove that the pairwise concurrency problem for 1-bounded conflict-free Petri nets is solvable in polynomial time. (In contract, the problem is EXPSPACE-complete for general Petri nets.)

CH3129-4/92/0000-0766 $3.00 © 1992 IEEE

Finally, we show that the pairwise concurrency problem, coupled with its polynomial time algorithm, provides an efficient tool for detecting race conditions in parallel problems and asynchronous circuits.

A *race condition* in a parallel program refers to the situation in which two statements try to access the same memory location simultaneously, and at least one of them attempts to alter the content of the location. The existence of race conditions will result in the outcome of the execution of a parallel program to be speed-dependent. As a result, different runs of the same program on the same data may result in different outcomes– which is clearly undesirable in program design. In the study of asynchronous circuits, a fundamental issue is to identify the situation in which the raising of a signal by one circuit element is in conflict with the lowering of the same signal by another element. (Such a conflict constitutes a race.) In this paper, we will show that our derived polynomial time algorithm for the pairwise concurrency problem for 1-bounded conflict-free Petri nets can be applied to answering the race detection problem for parallel programs as well as for asynchronous circuits. In fact, it was posed as an open question in [19] as to whether an efficient algorithm exists in the case of asynchronous circuits. We provide an answer to the question in the affirmative.

The contributions of this paper include the following:

1. From the analytical aspect of Petri nets, we show that the pairwise concurrency problem for 1-bounded conflict-free Petri nets is solvable in polynomial time. Since few Petri net problems have polynomial time solutions, our result is important in its own right. Furthermore, in solving the problem, we adopt a novel technique which may have other applications to the analysis of Petri nets.

2. By demonstrating the relations between the pairwise concurrency problem for Petri nets and the race detection problem for parallel programs and asynchronous circuits, we illustrate that conflict-free Petri nets are not just theoretically nice, they have real-world applications as well.

2 Definitions

Let Z (N) denote the set of (nonnegative) integers, and Z^k (N^k) the set of vectors of k (nonnegative) integers. For a k-dimensional vector v, let v(i), $1 \leq i \leq k$, denote the ith component of v. For a given value of k, let $\mathbf{0}$ denote the vector of k zeros (i.e., $\mathbf{0}(i) = 0$ for i = 1,...,k).

A *Petri Net* (PN, for short) is a 4-tuple (P,T,φ,μ_0), where P is a finite set of *places*, T is a finite set of *transitions*, φ is a *flow function* $\varphi : (P \times T) \cup (T \times P) \rightarrow \{0,1\}$, and μ_0 is the *initial marking* $\mu_0 : P \rightarrow N$. A *marking* is a mapping $\mu : P \rightarrow N$. A transition $t \in T$ is *enabled* at a marking μ iff for every $p \in P$, $\varphi(p,t) \leq \mu(p)$. A transition t may *fire* at a marking μ if t is enabled at μ. We then write $\mu \xrightarrow{t} \mu'$, where $\mu'(p) = \mu(p) - \varphi(p,t) + \varphi(t,p)$ for all $p \in P$. A sequence of transitions $\sigma = t_1 \ldots t_n$ is a *firing sequence* from μ_0 (or a firing sequence of (P,T,φ,μ_0)) iff $\mu_0 \xrightarrow{t_1} \mu_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} \mu_n$ for some sequence of markings $\mu_1,\ldots,\mu_n$. (We also write '$\mu_0 \xrightarrow{\sigma} \mu_n$'.) We write '$\mu_0 \xrightarrow{\sigma}$' to denote that σ is enabled and can be fired from μ_0, i.e., $\mu_0 \xrightarrow{\sigma}$ iff there exists a marking μ such that $\mu_0 \xrightarrow{\sigma} \mu$.

Given a sequence of transitions σ, we define $\#_\sigma$ to be a mapping $\#_\sigma : T \rightarrow N$ such that $\#_\sigma(t)$ = the number of occurrences of t in σ. Let $\mu \xrightarrow{\sigma} \mu'$. The *value* of σ, denoted by $\Delta(\sigma)$, is defined to be $\mu' - \mu$ ($\in Z^k$, where k is the number of places in the Petri net). We let $S(\sigma)$ denote the set of transitions occurring in σ, i.e., $S(\sigma) = \{t | t \in T, \#_\sigma(t) > 0\}$.

Let $\mathcal{P}$ = (P,T,φ,μ_0) be a PN. The *reachability set* of $\mathcal{P}$ is the set $R(\mathcal{P}) = \{\mu \mid \mu_0 \xrightarrow{\sigma} \mu \text{ for some } \sigma\}$. $\mathcal{P}$ is said to be $k-$*bounded* (for some $k \in N$) iff $\forall \mu \in R(\mathcal{P}), \forall p \in P$, $\mu(p) \leq k$. Given a place s, we let $s^\bullet = \{t | \varphi(s,t) = 1, t \in T\}$ and $^\bullet s = \{t | \varphi(t,s) = 1, t \in T\}$. A place s and a transition t are on a *self-loop* iff $t \in s^\bullet$ and $t \in {}^\bullet s$, i.e., s is both an input and output place of t. $\mathcal{P}$ is said to be *conflict-free* iff for every place s, either

1. $|s^\bullet| \leq 1$, or

2. $\forall t \in s^\bullet$, t and s are on a self-loop.

In words, a Petri net is conflict-free if every place which is an input of more than one transition is on a self-loop with each such transition ([10,12]). In a conflict-free Petri net, once a transition becomes enabled, the only way to disable the transition is to fire the transition itself. (That is, $\forall t, t' \in T$, $t \neq t'$, $\mu \xrightarrow{t} \mu'$ and $\mu \xrightarrow{t'}$ implies $\mu' \xrightarrow{t'}$.) Notice that marked graphs are conflict-free, although the converse need not be true. (See [17] for more about Petri nets.)

The *pairwise concurrency problem* for Petri nets is that of, given a Petri net $\mathcal{P}=(P,T,\varphi,\mu_0)$ and two transitions u and v, determining whether there exists a reachable marking μ (i.e., $\mu \in \mathbf{R}(\mathcal{P})$) such that both u and v are enabled in μ.

In the following section, we will show the problem to be solvable in polynomial time.

3 The complexity bound

We first show that, given two transitions u and v, if a path simultaneously enabling u and v exists, then there must exist a short 'witness' which can be partitioned into 'segments'. We then use a lemma from [8] to construct those segments one by one in polynomial time to answer the pairwise concurrency problem. First, we present some lemmas to set the stage for our polynomial time algorithm.

Lemma 3.1: Let $\mu_0 \xrightarrow{\sigma} \mu$ be a computation enabling transitions u and v simultaneously. Then there exist σ_1 and σ_2 such that (1) $\#_\sigma = \#_{\sigma_1\sigma_2}$, (2) $\mu_0 \xrightarrow{\sigma_1\sigma_2} \mu$, (3) $S(\sigma_2) \subseteq S(\sigma_1)$, and (4) $\forall r$, $\#_{\sigma_1}(r) \leq 1$. (In words, $\sigma_1\sigma_2$ is a rearrangement of σ such that if a transition occurs in σ, it can also be found in σ_1; in addition, no transition in σ_1 appears more than one time in σ_1.)

Proof. First, we claim that if $\mu_0 \xrightarrow{\delta_1\delta_2 t} \mu'$ (where δ_1, δ_2 are sequences of transitions and t is a transition) and $\forall$ transition r, $S(\delta_2) \subseteq S(\delta_1)$, $0 \leq \#_{\delta_1}(r) \leq 1$, and $t \notin S(\delta_1)$, then $\mu_0 \xrightarrow{\delta_1 t \delta_2} \mu'$.

To prove the above claim, assume that t is not enabled in μ'', where $\mu_0 \xrightarrow{\delta_1} \mu''$. This implies one of t's input places, say p, must be empty in μ''. However, t is enabled in μ''', where $\mu_0 \xrightarrow{\delta_1\delta_2} \mu'''$, indicating the existence of a transition, say t' in δ_2, which deposits a token to p. Since $S(\delta_2) \subseteq S(\delta_1)$, t' must be in δ_1. Since the Petri net is conflict-free and $t \notin S(\delta_1)$, $\mu''(p) \neq 0$ – a contradiction. So t must be enabled in μ''. Since the Petri net is conflict-free and $t \notin S(\delta_2)$ (because $S(\delta_2) \subseteq S(\delta_1)$ and $t \notin S(\delta_1)$), the firing of t in μ'' will not affect the enabledness of the subsequent transition sequence δ_2. This completes the proof of the claim.

To prove the statement of the lemma, we repeatedly apply the above procedure to σ by examining one transition at a time, starting from the leftmost one. ($\delta_1 = \delta_2 = \lambda$, initially.) If the examined transition has not yet been encountered, add that transition to δ_1; otherwise, add to δ_2. □

The next lemma indicates that if u and v can become enabled simultaneously, then there exists a short witness satisfying certain properties. More precisely, we have:

Lemma 3.2: If $\mu_0 \xrightarrow{\sigma} \mu$ is the shortest (or one of the shortest) computation enabling transitions u and v simultaneously, then σ can be rearranged into $\sigma_1\sigma_2\cdots\sigma_k$ such that (1) $\mu_0 \xrightarrow{\sigma_1\sigma_2\cdots\sigma_k} \mu$, (2) $\forall 1 \leq i \leq k$, $\forall$ transitions r, $\#_{\sigma_i}(r) \leq 1$, (3) $\forall 1 \leq i \leq k-1$, $S(\sigma_{i+1}) \subseteq S(\sigma_i)$, and (4) $k \leq n$ and $|\sigma| \leq n^2$, where n is the number of transitions of the Petri net.

Proof. Conditions (1)-(3) can be derived easily by repeatedly applying Lemma 3.1. In what follows, we prove (4).

We first show that $k \leq n$. Suppose this is not the case. Then there must exist a j, $1 \leq j \leq k-1$ such that $S(\sigma_j) = S(\sigma_{j+1})$ (since $S(\sigma_1), S(\sigma_2), \cdots, S(\sigma_k)$ forms a 'shrinking' sequence of nonempty sets (i.e., $S(\sigma_1) \supseteq S(\sigma_2) \supseteq \cdots \supseteq S(\sigma_k)$); furthermore, there are at most n transitions in the Petri net). Consider the following three cases:

1. $\Delta(\sigma_j)$ contains a negative component: Since the Petri net is 1-bound and conflict-free, σ_{j+1} could never be fired.

2. $\Delta(\sigma_j) > \mathbf{0}$: In this case, 'pumping' σ_j infinitely

many times will render the Petri net unbounded – a contradiction.

3. $\Delta(\sigma_j) = \mathbf{0}$: In this case, σ_j can be removed without affecting the simultaneous enabledness of u and v. This contradicts the assumption that σ be the shortest.

$|\sigma| \leq n^2$ follows immediately from $k \leq n$ and the fact that $\sigma_i, \forall i$, does not contain any transition more than once. □

Even though Lemma 3.2 allows us to put a polynomial upper bound on the length of the shortest path enabling u and v, a polynomial time algorithm for the pairwise concurrency problem does not follow immediately. However, we can use the 'maximum' sequence of transitions enabled in the starting marking of that segment, rather than the exact σ_i stated in Lemma 3.2. It is then possible to construct a path (even though it may no longer be the shortest; it is still polynomial in length) in polynomial time to enable u and v, if any such path exists.

Lemma 3.3: Let $\mu_0 \stackrel{\sigma_1\sigma_2}{\rightarrow} \mu$ be a computation enabling transitions u and v simultaneously. If σ_1' is a sequence satisfying (1) $\mu_0 \stackrel{\sigma_1'}{\rightarrow}$, (2) $\forall$ transitions r, $\#_{\sigma_1}(r) \leq \#_{\sigma_1'}(r) \leq 1$, (3) $S(\sigma_2) \subseteq S(\sigma_1)$, (4) $u, v \notin S(\sigma_1') - S(\sigma_1)$, then $\mu_0 \stackrel{\sigma_1'\sigma_2}{\rightarrow} \mu'$, for some μ', and u and v are enabled in μ'.

Proof. According to conditions (2) and (3) , $S(\sigma_1') - S(\sigma_1)$ does not contain any transition belonging to σ_2. This, in conjunction with the conflict-freedom property, ensures that no transition in σ_2 will become disabled as a result of firing transitions in $S(\sigma_1') - S(\sigma_1)$. Hence, $\mu_0 \stackrel{\sigma_1'\sigma_2}{\rightarrow} \mu'$, for some μ'. In addition, (4) implies that u and v are enabled in μ'. □

The following lemma indicates the existence of a polynomial time algorithm to construct the maximum sequence of transitions in a given marking. In addition, the sequence consists of each of the enabled transitions exactly once.

Lemma 3.4: (from [8]) Given a conflict-free Petri net $\mathcal{P}=(P, T, \varphi, \mu_0)$, we can construct in polynomial time a path σ enabled in μ_0 in which no transition in σ is used more than once, such that if some transition t is not used in σ, then there is no path (emanating from μ_0) in which t is used.

Lemma 3.5: Given a 1-bounded conflict-free Petri net $\mathcal{P}=(P, T, \varphi, \mu_0)$, let $\mu_0 \stackrel{\sigma_1\sigma_2\cdots\sigma_k}{\rightarrow} \mu$, $1 \leq k \leq n$, be the shortest computation guaranteed by Lemma 3.2. Let T_i be $T - (\{u, v\} - S(\sigma_i))$, and φ_i be the restriction of φ on T_i. Suppose $\sigma_1'\sigma_2'\cdots\sigma_k'$ is a sequence of transitions defined recursively as follows: $\forall 1 \leq i \leq k$, σ_i' is the sequence of transitions guaranteed by Lemma 3.4 with respect to $(P, T_i, \varphi_i, \mu_{i-1})$, where $\mu_0 \stackrel{\sigma_1'\cdots\sigma_{i-1}'}{\rightarrow} \mu_{i-1}$. Then $\mu_0 \stackrel{\sigma_1'\sigma_2'\cdots\sigma_k'}{\rightarrow} \mu'$ is a computation enabling u and v simultaneously (i.e., u and v are enabled in μ').

Proof. First consider σ_1'. It is easy to see the following:

1. $\forall r$, $\#_{\sigma_1}(r) \leq \#_{\sigma_1'}(r) \leq 1$.
   $\#_{\sigma_1}(r) \leq 1$ and $\#_{\sigma_1'}(r) \leq 1$ are due to Lemmas 3.2 and 3.4, respectively. (That is, no transition occurs more than once in σ_1 and σ_1'.) According to Lemma 3.4, if a transition is not in σ_1', then there is no path in which the transition is used. Hence, $\forall r$, $\#_{\sigma_1}(r) \leq \#_{\sigma_1'}(r)$.

2. $S(\sigma_2 ... \sigma_k) \subseteq S(\sigma_1)$ – Lemma 3.2.

3. $u, v \notin S(\sigma_1') - S(\sigma_1)$.
   Recall that $T_1 = T - (\{u, v\} - S(\sigma_1))$ and σ_1' is with respect to $(P, T_1, \varphi_1, \mu_0)$. Hence, if u (v) is not in σ_1, then it is not in σ_1' either.

Using Lemma 3.3, we immediately have that $\mu_0 \stackrel{\sigma_1'}{\rightarrow} \mu_1 \stackrel{\sigma_2\cdots\sigma_k}{\rightarrow} \mu''$, for some μ'', and u and v are enabled in μ''. Starting in μ_1 and repeatedly applying the above argument, we can easily show that $\mu_0 \stackrel{\sigma_1'\sigma_2'\cdots\sigma_k'}{\rightarrow} \mu'$ is a computation enabling u and v simultaneously. □

The above lemma suggests an iterative way of constructing a path to enable u and v simultaneously, if such a path exists. In each stage, we construct a sequence of maximum length which is enabled in the resulting marking of the previous stage. The only problem with this approach is that for each i, the presence or absence of u (or v) in $\sigma_i', 1 \leq i \leq k$, must coincide with that in σ_i. Since we have no knowledge of the transitions

used in σ_i, how do we know whether we should include or exclude u and v in constructing the maximum sequence? At first glance, it seems that nondeterministic steps (for guessing the presence of absence of u and v in all $k, 1 \leq k \leq n$, segments) were inevitable. A careful examination, however, reveals that such nondeterminism can be avoided by taking advantage of the 'shrinking property' of the sequence $\sigma_1, \sigma_2, ..., \sigma_k$ mentioned in Lemma 3.2. More precisely, if u (or v) does not appear in, say σ_j, then u (or v) will never occur in subsequent segments $\sigma_{j+1}, ..., \sigma_k$. This is exactly the key property upon which our polynomial time algorithm relies.

Theorem 3.6: The pairwise concurrency problem for 1-bounded conflict-free Petri nets is solvable in polynomial time.

Proof. We let *Find-Max-Seq*(P, T, φ, μ_0) (where (P, T, φ, μ_0) is a conflict-free Petri net) be a procedure that outputs a sequence of transitions σ in which no transition in σ is used more than once, and if some transition t is not used in σ, then there is no path in (P, T, φ, μ_0) in which t is used. The existence of a such procedure is guaranteed by Lemma 3.4.

We construct the sequence $\sigma'_1\sigma'_2 \cdots \sigma'_k$ stated in Lemma 3.5. In our procedure, we use two variables, namely i and j, to represent the indices of the segments in which u and v, respectively, are not used for the first time. (More precisely, u (v) is assumed to be used in σ'_l, $1 \leq l \leq i-1$ (σ'_l, $1 \leq l \leq j-1$), but not in σ'_r, $i \leq r \leq k$ (σ'_r, $j \leq r \leq k$).) Since we do not know in advance the exact values of i and j in the path enabling u and v, we consider all possibilities of i and j, each of which ranges from 1 to $n+1$. (For example, $i = 1$ indicates that u was never used; $i = n+1$ indicates that u is used in every segment.) The algorithm is as follows.

Procedure Concurrent-Pair$(P, T, \varphi, \mu_0, u, v)$
/* $u, v \in T$; This procedure is to determine whether transitions u and v can become enabled simultaneously in a 1-bounded conflict-free Petri net (P, T, φ, μ_0). */

```
for i := 1 to n+1 do
  for j := 1 to n+1 do
    begin
      T' := T;
      μ := μ0;
      for l := 1 to n do
        begin
          if i = l then T' := T' − {u};
          if j = l then T' := T' − {v};
          σ:=Find − Max − Seq(P, T', φ', μ);
          let μ' be the marking such that μ −σ→ μ';
          if u and v are enabled in μ'
              then EXIT and return 'YES'
              else μ := μ'
        end
    end
return 'NO'
```

The procedure should be quite easy to understand. We thus have a polynomial time algorithm for the pairwise concurrency problem. □

4 Two applications

In this section, we demonstrate how the polynomial time result obtained in the previous section (concerning the pairwise concurrency problem for 1-bounded conflict-free Petri nets) can be applied to the detection of race conditions in parallel programs and asynchronous circuits.

4.1 Parallel Programs

As defined in [2], a parallel program consists of several tasks running simultaneously (i.e., in parallel). Tasks can be thought of as independently executing sequential processes except at points where they communicate. Synchronization between tasks is by means of the so-called *event variables*. Two kinds of operations can be applied to an event variable, namely *post* and *wait*. Upon reaching a *wait var* statement, the normal execution of a task will be suspended if the associated event variable *var* has not been posted; the task will resume its normal execution only when *var* becomes posted by another task. A *race condition* can be defined as the situation in which two statements try to

access the same memory location simultaneously, and at least one of them attempts to alter the content of the location. The existence of race conditions will result in the outcome of the execution of a parallel program to be speed-dependent. As a result, different runs of the same program on the same data may result in different outcomes– which is clearly undesirable in program design.

In what follows, we use a simple example to motivate the idea of using the pairwise concurrency problem for Petri nets to capture the essence of race conditions in parallel programs.

Example 1: Consider a simple parallel program P consisting of three tasks T_1, T_2 and T_3 in Table 1. (The program was given in [2] as an illustrating example.)

Task 1	Task 2	Task 3
x:=1	w:=2	v:=3
post e1	wait e1	wait e2
y:=x+1	z:=x+w	x:=v+z
wait e4	post e2	post e3
print x,y,z	wait e3	z:=z+v
	print x,z	post e4

Table 1: A simple parallel program.

The use of Petri nets to model parallel programs is not new. (See e.g. [14,16] for using Petri nets to model Ada programs.) Using the similar idea (as that in [14, 16]), P can easily be modeled by the Petri net depicted in Figure 1. (Notice that places $e1, e2, e3$ and $e4$ are used for representing the four event variables in P; in each of these places, the presence of a token indicates that its associated event variable has been posted.)

Now consider a possible race between statements $y := x+1$ (of Task 1) and $x := v+z$ (of Task 3) in the parallel program P given in Example 1. (Note that variable x is written by Task 3 and is read by Task 1; by definition, this constitutes a possible race.) By tracing the first few steps of the execution of P, it is not hard to see that a race indeed could happen between these two statements. Hence, the program is not race-free. A careful examination of Petri net N will also reveal the same fact. More precisely, there is a firing sequence which enables the two transitions labeled $y := x+1$ (of Task 1) and $x := v+z$ (of Task 3) simultaneously.

In view of the above, detecting whether a parallel program is race-free is closely related to the pairwise concurrency problem for Petri nets. Given a parallel program P, we define PR(P)=$\{(t,t')$ | statements t, t' belong to two different tasks in P, t and t' have certain variables in common, and at least one of t and t' tries to write to one of their shared variables.$\}$ (In words, PR(P) defines the set of all possible pairs of conflict statements in P.) Given a Petri net $N = (P, T, f, \mu_0)$, let $Con(N)$=$\{(t,t')$ | there exists a reachable marking in N such that transitions t, t' are enabled simultaneously $\}$. We have the following easily shown result:

Theorem 4.1: *Given a parallel program P, P is race-free iff PR(P) $\cap$ Con(N) $= \emptyset$, where N is a Petri net modeling P.*

Suppose we let class C to be the class of parallel programs satisfying the following constraints:

1. No if-then-else statements.
2. Do loops are allowed; but post statements are disallowed inside a do loop.
3. No two tasks can post the same event variable.

It is not hard to see that for each class C parallel program, its associated Petri net is 1-bounded and conflict-free. Using the complexity result obtained in the previous section, we have the following theorem:

Theorem 4.2: *The race detection problem for class C parallel programs can be solved in polynomial time.*

4.2 Asynchronous circuits

In [3,4,19], Petri nets were used to model the behaviors of asynchronous circuits. Consider the so-called Muller C-element (see [3]) whose diagram is depicted in Figure 2. Such element has been modeled by a Petri net displayed in Figure 3 [3]. The labels $a+$ and $a-$ represent the raising and lowering, respectively, of input a's signal. ($b+$, $b-$, $c+$, and $c-$ are defined likewise.) By

raising (lowering) a signal we mean the change of the signal from 0 to 1 (1 to 0). In the study of an asynchronous circuit, a fundamental issue is to identify the situation in which the raising of a signal by one element is in conflict with the lowering of the same signal by another element. (This kind of conflict can be thought of as a *race* in the circuit.) In the modeling Petri net, this sort of conflict corresponds to the situation in which two designated transitions (modeling the raising and the lowering of a signal in the modeled circuit) become enabled simultaneously – which is exactly what the pairwise concurrency problem is about. Furthermore, it was suggested in [3,4,19] that the class of 1-bounded conflict-free Petri nets is exactly the one needed to model asynchronous circuits. As a result, our polynomial time algorithm derived in the previous section provides a tool for analyzing whether a circuit suffers from the kind of race described above. In fact, it was posed as an open problem in [19] as to whether a such polynomial time algorithm exists. In this paper, we answer the question in the affirmative.

5 Conclusion

In this paper, we have used the pairwise concurrency problem for (1-bounded conflict-free) Petri nets to capture the notion of a race condition for a class of parallel programs (and asynchronous circuits). We have also demonstrated the pairwise concurrency problem for 1-bounded conflict-free Petri nets to be solvable in polynomial time. As a result, the race detection problem for parallel programs (and asynchronous circuits) can be solved in polynomial time as well. Several questions remain to be answered. They include:

1. Can we extend our work to more complicated parallel programs?
2. Can other problems concerning parallel programs (or asynchronous circuits) be modeled and analyzed by Petri nets?
3. Does the pairwise concurrency problem defined in this paper have other applications?

References

[1] H. Baker. Rabin's Proof of the Undecidability of the Reachability Set Inclusion Problem of Vector Addition Systems. Memo 79. MIT Project MAC, Computer Structure Group, Cambridge, MA, 1973.

[2] V. Balasundaram and K. Kennedy. Compile-time Detection of Race Conditions in a Parallel Program. *Proc. of the International Conference on Supercomputing*, Crete, Greece, June 1989, pp. 175-185.

[3] J. Brzozowski and C. Seger. Advances in Asynchronous Circuit Theory (Part II). *EATCS Bulletin* 43:199-263, 1991.

[4] T. Chu. Synthesis of Self-Timed VLSI Circuits from Graph-Theoretical Specifications. Ph.D. Thesis, MIT, 1987.

[5] S. Crespi-Reghizzi and D. Mandrioli. A Decidability Theorem for a Class of Vector Addition Systems. *Information Processing Letters*, 3(3):78–80, 1975.

[6] R. Howell and L. Rosier. Completeness Results for Conflict-Free Vector Replacement Systems. *J. of Computer and System Sciences*, 37:349–366, 1988.

[7] R. Howell and L. Rosier. On Questions of Fairness and Temporal Logic for Conflict-Free Petri Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1988*, pages 200–226, Springer-Verlag, Berlin, 1988. LNCS 340.

[8] R. Howell, L. Rosier, and H. Yen. An $O(n^{1.5})$ Algorithm to Decide Boundedness for Conflict-Free Vector Replacement Systems. *Information Processing Letters*, 25:27–33, 1987.

[9] R. Howell, L. Rosier and H. Yen. Normal and Sinkless Petri Nets. Proc. of 7th International Conference on the Fundamentals of Computing Theory, Szeged, Hungary, August 1989, pp. 234-243. To appear in *Journal of Computer and System Sciences.*

[10] N. Jones, L. Landweber, and Y. Lien. Complexity of Some Problems in Petri Nets. *Theoret. Comp. Sci.*, 4:277–299, 1977.

[11] R. Kosaraju. Decidability of Reachability in Vector Addition Systems. *Proc. 14th Ann. ACM Symp. on Theory of Computing*, 267-280, 1982.

[12] L. Landweber and E. Robertson. Properties of Conflict-Free and Persistent Petri Nets. *JACM*, 25(3):352–364, 1978.

[13] R. Lipton. The Reachability Problem Requires Exponential Space. Technical Report 62. Yale University, Dept. of CS., New Haven, Conn., January 1976.

[14] D. Mandrioli, R. Zicari, C. Ghezzi and F. Tisato. Modeling the Ada Task System by Petri Nets. *Comput. Lang.* 10 (1): 43-61, 1985.

[15] E. Mayr. An Algorithm for the General Petri Net Reachability Problem. *SIAM J. Comput.* 13 (3): 441-460, 1984.

[16] T. Murata, B. Shenker and S. Shatz. Detection of Ada Static Deadlocks Using Petri Net Invariants. *IEEE Transactions on Software Engineering* 15(3): 314-325, March 1989.

[17] J. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Englewood Cliffs, N.J., 1981.

[18] C. Rackoff. The Covering and Boundedness Problems for Vector Addition Systems. *Theoret. Comp. Sci.*, 6:223–231, 1978.

[19] M. Tiusanen. Some Unsolved Problems in Modeling Self-Timed Circuits Using Petri Nets. *EATCS Bulletin* 36:152-160, 1988.

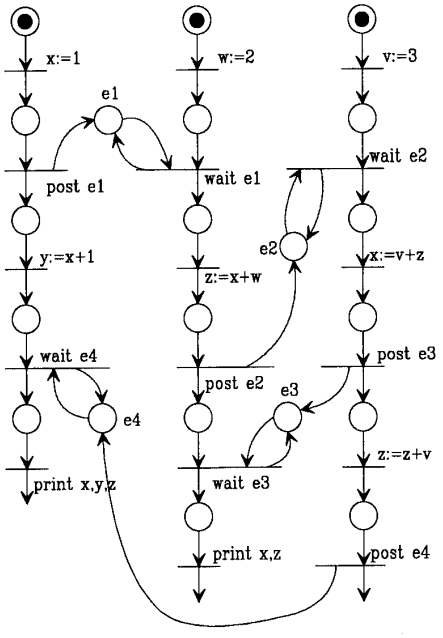

Figure 1. Petri net N modeling program P in Table 1.

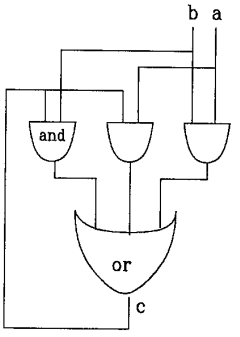

Figure 2: Circuit for Muller's C-element (see [3]).

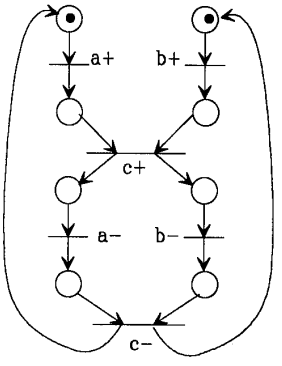

Figure 3: Petri net for C-element (see [3]).