Accepted for publication in *Acta Informatica*.

Reduction Rules for Time Petri Nets

Robert H. Sloan* and Ugo Buy†
Dept. of Electrical Engineering and Computer Science
851 S. Morgan St. Room 1120
University of Illinois at Chicago
Chicago, IL 60607-7053 USA

Abstract

The goal of net reduction is to increase the effectiveness of Petri-net-based real-time program analysis. Petri-net-based analysis, like all reachability-based methods, suffers from the state explosion problem. Petri net reduction is one key method for combating this problem.

In this paper, we extend several rules for the reduction of ordinary Petri nets to work with time Petri nets. We introduce a notion of equivalence among time Petri nets, and prove that our reduction rules yield equivalent nets. This notion of equivalence guarantees that crucial timing and concurrency properties are preserved.

*Partially supported by NSF grants CCR-9108753 and CCR-9314258. Email: `sloan@eecs.uic.edu`.

†Partially supported by NSF grants CCR-9109231 and CCR-9314258. Email: `buy@eecs.uic.edu`.

1 Introduction

Petri nets have proven to be a very useful tool for the analysis of concurrent systems. To date several approaches have been defined that use Petri nets to model a system being analyzed (e.g., [1, 8, 9, 11, 20]). Other researchers use suitable timed extensions of Petri nets for the analysis of real-time systems (e.g., [4, 5, 15, 22]). Most of the existing approaches are aimed at generating the set of reachable states of a Petri net model. A problem shared by these approaches is that the state space of a Petri net can be quite large, even for a relatively small system. This problem is generally referred to as the *state explosion problem*. In order for Petri-net-based reachability analysis to be practical, suitable techniques for controlling the state explosion problem must be developed.

To date, various techniques for reducing state space size in (untimed) concurrent systems have been defined (e.g., [13, 19–21]). Among these techniques, Petri net reduction has been shown to be especially effective, for instance, for automated deadlock detection in Ada programs [9]. This technique is aimed at reducing the size of a Petri net modeling a concurrent program, while retaining crucial concurrency properties, such as deadlock, in the reduced model. In general, any reduction in Petri net size results in even greater reductions in the size of the corresponding state space.

Here we show how the reduction techniques established for untimed nets can be extended to a kind of Petri net that includes the notion of time. This work is part of an ongoing project for the real-time analysis of programs written in high-level programming languages, such as Ada. Our work is distinct from other work on net reduction techniques in that we are interested in preserving timing properties, in addition to safety properties (e.g., deadlock), in our reduced nets. For this reason, we first introduce a language-theoretic notion of equivalence between a reduced and an unreduced net. Next, we define a set of reduction rules for time Petri nets [15]. Finally, we prove that each reduction rule satisfies our definition of equivalence for reduced and unreduced nets.

Researchers have developed several extensions of Petri nets to the timed domain. For our net reduction technique, we have chosen Merlin and Faber's *time Petri nets* [15]. These nets provide a reasonable compromise between expressive power and analyzability. They have also been shown to have more expressive power than several other timed extensions of Petri nets [4, 12]. At least two other approaches to real-time program analysis use time Petri nets [4, 22].

Several other authors have investigated reduction techniques for Petri nets. In particular, Berthelot originally developed a large set of reduction rules for general Petri nets [2, 3]. Berthelot's approach was subsequently extended by Haddad to

Colored Petri nets [14]. Reduction rules for a Petri net subset called *free-choice nets* were developed by Desel [8], and by Esparza and Silva [10].

Tu et al. applied Berthelot's approach to the analysis of Ada programs [20]. In particular, Tu et al. exploited Ada semantics in order to speed up the application of Berthelot's rules, some of which have high computational complexity. Duri et al. report on experiments with various optimization techniques for Petri-net-based deadlock detection in Ada programs [9]. Their approach was been successfully applied to a variety of examples of concurrent programs, including some real-world systems developed at industrial organizations. Their experimental results showed that the net reduction technique is especially effective at reducing state space size [9].

Our work is similar in spirit to the work reported by Berthelot and Haddad [3, 14], in that we lay the theoretical foundation for practical applications of the net reduction technique to real-time program analysis. We developed these rules for use in our system for analyzing real-time programs. However, we believe that net reduction will be necessary for *any* automated method that uses time Petri nets for the analysis of truly large systems.

This paper is organized as follows. In Section 2, we introduce time Petri nets and formally define our notion of net equivalence. In Section 3, we define four rules that reduce time Petri nets by merging transitions. In Section 4, we define rules that reduce time Petri nets by removing places. Finally, in Section 5, we present some conclusions and future research directions.

2 Notation and definitions

2.1 Brief review of untimed Petri nets

In this subsection we will quickly review key definitions. A more general discussion on Petri nets can be found in Peterson's book [17], and in the excellent survey article by Murata [16].

A *Petri net* is a four-tuple $N = (P, T, F, M_0)$ where P and T are the node sets and F the edges of a directed bipartite graph, and $M_0 : P \to \mathbb{N}$ is called the *initial marking* (or initial state) of N.[1] (We use $\mathbb{N}$ to denote the set $\{0, 1, 2, \ldots\}$.) We call P the set of *places* of N and T the set of *transitions* of N. In diagrams, we will show places as circles and transitions as bars. Formally, $F \subseteq (P \times T) \cup (T \times P)$, and F is called the *flow relation* (or edges) of N. In general a marking of N associates a nonnegative integer number of markers or *tokens* with each place.

[1] In much of the literature these would be called ordinary Petri nets.

For net $N = (P, T, F, M_0)$, we use the following symbols and notations for the sets of predecessors and successors of a place $p \in P$ and transition $t \in T$.

$$\begin{aligned}
{}^\bullet t = \{p | (p,t) \in F\} \quad &= \quad \text{the set of input places of } t \\
t^\bullet = \{p | (t,p) \in F\} \quad &= \quad \text{the set of output places of } t \\
{}^\bullet p = \{t | (t,p) \in F\} \quad &= \quad \text{the set of input transitions of } p \\
p^\bullet = \{t | (p,t) \in F\} \quad &= \quad \text{the set of output transitions of } p.
\end{aligned}$$

We extend this notation from a single node to a set of nodes; for instance, if S is a set of places, then ${}^\bullet S = \bigcup_{p \in S} {}^\bullet p$.

A transition is *enabled* when all its input places have at least one token. When an enabled transition t is fired, a token is removed from each input place of t and a token is added to each output place; this gives a new marking. For net $N = (P, T, F, M_0)$, the *language of* N, denoted $L(N)$, is the set of all legal sequences $\omega \in T^*$ of transition firings starting from marking M_0.

Petri net $N = (P, T, F, M_0)$ is *safe* if $M_0 : P \to \{0, 1\}$, and if all markings reachable by legal sequences of transition firings from the initial marking have either zero or one tokens in every place.

2.2 Time Petri nets

With *time Petri nets* [4, 15] a global time is considered, and the state of a net is defined not only by a marking, but also by a value of the global time. As we will discuss, transitions can fire only at particular values of time.

Formally, a *time Petri net* is a five-tuple (P, T, F, M_0, SI) where (P, T, F, M_0) is an ordinary Petri net, and SI associates a *static (firing) interval* $\mathcal{I}(t) = [a, b]$ with each transition *t*, where a and b are rationals in the range $0 \le a \le b \le +\infty$, with $a \ne +\infty$.

Static intervals change the behavior of a time Petri net with respect to an ordinary Petri net in the following way. If transition t with $\mathcal{I}(t) = [a, b]$ becomes enabled at time θ_0, then transition t *must* fire in the time interval $[\theta_0 + a, \theta_0 + b]$, unless it becomes disabled by the removal of tokens from some input place in the meantime. The *static earliest firing time* of transition t is a; the *static latest firing time* of t is b; the *dynamic earliest firing time* of t is $\theta_0 + a$; the *dynamic latest firing time* of t is $\theta_0 + b$; the *dynamic firing interval* of t is $[\theta_0 + a, \theta_0 + b]$. (We will omit the word "dynamic" when it is clear from context.) Note that ordinary, untimed Petri nets are the special case of time Petri nets with all static intervals equal to $[0, +\infty]$.

Note that for time Petri nets, as opposed to ordinary, untimed Petri nets, once a transition is enabled it must fire by the end of its dynamic firing interval (unless it becomes disabled before then). This feature is called *strong time semantics*; other models of Petri nets with time that do not force transitions to fire have *weak time semantics*.

The state of a time Petri net is a triple (M, Θ, I), where M is the marking of the underlying untimed Petri net, Θ is the global time, and I is a vector containing the dynamic firing interval of each transition enabled by M. The initial state of a time Petri net consists of its initial marking, time 0, and a vector containing the static firing interval of each transition enabled by this marking.

We say that a transition t is *fireable* in state (M, Θ, I) if it is enabled (M associates at least one token with all of ${}^\bullet t$), and if it is legal to fire t next. This will be true if and only if the dynamic earliest firing time of t is less than or equal to the dynamic latest firing time of all other enabled transitions. Of course, even with strong time semantics, t's being fireable in state (M, Θ, I) does not necessarily mean that t *will* fire at time Θ.

A *(firing) schedule* for time Petri net $N = (P, T, F, M_0, SI)$ is a finite sequence of ordered pairs (t_i, θ_i) such that transition t_1 is fireable in state $(M_0, \Theta_1, I_{\Theta_1})$ (where I_{Θ_1} is the vector of dynamic firing intervals of enabled transitions of N at time Θ_1), and transition t_i is fireable in the state reached at time Θ_i by starting in the initial state of N and firing the transitions t_j for $1 \leq j < i$ in the schedule at the given times.[2] For time Petri net N with schedule ω, we denote the state reached by starting in N's initial state and firing each transition in ω at its associated time by $\mathcal{S}(N, \omega)$. The time of $\mathcal{S}(N, \omega)$ is the global firing time of the last transition in ω. Since the set of all firing schedules of time Petri net N is an extension of the language of an untimed net, we will also denote it by $L(N)$.

Henceforth, we will assume that the underlying Petri net of all our time Petri nets is safe. This assumption is reasonable because nets generated automatically from high-level programming languages are generally safe [7, 9, 20]. Other authors have typically made some somewhat weaker restriction in their definitions (for instance, Yoneda et al. require that the markings in the states reachable in the time Petri net, as opposed to the underlying untimed Petri net, contain only one token per place [22]), but all the examples we have seen presented in the literature conform to our assumption. Without this assumption, some of the rules we introduce here are not valid, resulting in a weaker technique. This is further discussed in Section 3.1 below.

[2] The definition of states for time Petri nets implies that any one transition firing takes us from one state to another, although more than one transition can fire at the same global time.

Let X be a set of nodes of time Petri net N. We use $N|X$ for the time Petri net obtained by restricting the underlying graph to X, and $N - X$ to denote $N|\bar{X}$. This notation is also used for net states and firing schedules. For example, $N - \{t_1, t_2\}$ refers to what is left of net N once one removes the transitions t_1 and t_2 together with all arcs incident on either t_1 or t_2. As another example, if (M, Θ, I) is a state of net N, then $(M, \Theta, I) - \{t, p\}$ is the same state with place p deleted from the marking and transition t's firing interval, if any, deleted from vector I.

Finally, for time Petri net $N = (P, T, F, M_0, SI)$ and $X \subset P \cup T$, we define $L(N)|X$ to be the set of all schedules of N with all pairs containing a transition not from X deleted. Also, $L(N) - X = L(N)|(T \setminus X)$.

2.3 Interval arithmetic

The following notation will be helpful: Let $I_1 = [a_1, b_1]$ and $I_2 = [a_2, b_2]$, with $0 \leq a_i \leq b_i \leq +\infty$. Then we define $I_1 + I_2$ to be the interval $[a_1 + a_2, b_1 + b_2]$. In the special case where I_2 is a point interval—that is, $a_2 = b_2$—and, additionally, $a_2 \leq a_1$ we define $I_1 - I_2$ to be the interval $[a_1 - a_2, b_1 - b_2]$. The sum of an interval with a constant is defined in the obvious way.

2.4 Equivalence of nets

The goal of this work is to take a time Petri net N and to reduce it to a smaller net N' that is more amenable to analysis. Of course, we want the net N' to be in some sense the "same" as the original net N. We will concentrate on language preservation (among unmodified transitions). This means that timing properties are preserved, as is liveness. Consequently, answers to questions concerning the real-time behavior of concurrent programs are also preserved in the reduced nets.

DEFINITION. Fix some rule for transforming a time Petri net N to a time Petri net N'. Let U be the set of transitions of N which are left completely unmodified by this transformation. We say that the (reduced) net N' is *equivalent to net N* after the transformation if

1. $L(N)|U = L(N')|U$, and
2. If the underlying untimed Petri net of N is safe, then so is the underlying untimed net of N'.

Note that the second condition is almost, but not quite, implied by the first condition. We need it to make sure that none of the transitions in U destroy the safeness of the net. This is important because typically a real-time program will be correct only if the time Petri net that models it is safe. Note that the key property we wish to preserve is the safeness of the time Petri net; safeness of the underlying Petri net is a technically convenient way to achieve this goal.

2.5 Time Petri nets and Ada code

Our reduction method is part of a long-term project for the real-time analysis of Ada tasking programs. An important aspect of our project is the translation of Ada code into a suitable Petri net model that incorporates a notion of time. Our translation method extends the method defined by Shatz et al. for (untimed) concurrency analysis Ada tasking programs [18]. As with that earlier method, our nets exhibit a highly uniform structure because similar subnets model different occurrences of a given construct in an Ada program. For instance, the subnet modeling communication between two processes (i.e., a rendezvous between two tasks in Ada terminology) consists of six places and four transitions. In contrast to the earlier method, our method also takes into account timing constructs appearing in Ada code.

Time Petri nets can capture timing aspects of Ada code in a relatively straightforward manner. For instance, Ada's *delay* statement, which delays execution by a given amount of time, can be modeled by a simple subnet consisting of two places serially connected by a transition. In this subnet, the transition is the only output of the first place and the only input to the second place. Thus, a delay of d time units is captured by associating the interval $[d, d]$ with this transition.

Our approach to real-time analysis uses Simple Time Petri (STP) nets to model Ada code. STP nets are a special case of time Petri nets in which interval delays associated with transitions are replaced by point delays.[3] In addition to delay statements, our translation method can capture most other Ada constructs involving time, such as timed entry calls and delay alternatives in selective wait statements. Additional details on our translation method can be found elsewhere [6].

3 Reduction rules that fuse transitions

In this section we present several reduction rules that fuse together two or more transitions. We will give the proof that the first rule preserves equivalence in complete detail, and then be a little briefer for the remaining rules.

[3]This simplification was specifically introduced in STP nets to speed up state space generation and to facilitate real-time analysis [5].

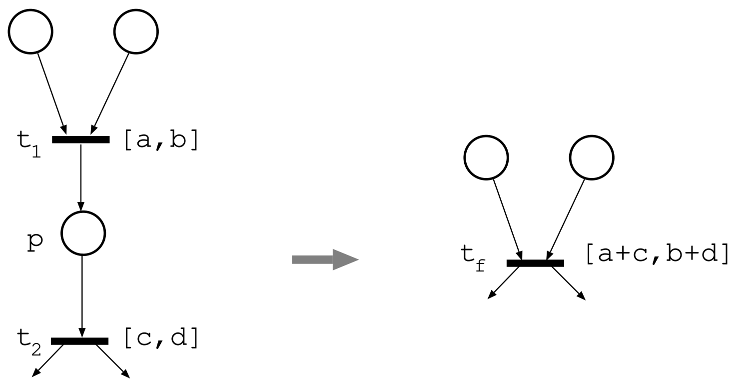

Figure 1: Serial fusion reduction rule.

3.1 Serial Fusion

This is a variation of Berthelot's post-fusion rule [3, 20] which we have found occurs frequently in practice.

Reduction Rule 1 (Serial fusion) *Preconditions: There exist two transitions t_1 and t_2 and a place p satisfying:*

1. *Place p is unmarked in the initial marking.*
2. *$t_1{}^\bullet = \{p\} = {}^\bullet t_2$. (Place p is the unique output place of t_1 and the unique input place of t_2.) Also, $p \notin {}^\bullet t_1$ and $p \notin t_2{}^\bullet$.*
3. *Place p is disconnected from all other transitions.*
4. *Either $({}^\bullet t_1)^\bullet = \{t_1\}$ (transition t_1 does not share any of its input places with other transitions), or $\mathcal{I}(t_2) = [0, 0]$.*

Rule: Transitions t_1 and t_2 may be replaced by a single transition t_f with static interval $\mathcal{I}(t_f) = \mathcal{I}(t_1) + \mathcal{I}(t_2)$, with input set ${}^\bullet t_f = {}^\bullet t_1$, and with output set $t_f{}^\bullet = t_2{}^\bullet$.

(See Figure 1.)

The first three preconditions are necessary for untimed Petri nets as well [20]. We illustrate the necessity of the fourth condition in Figure 2. On the left is a net fragment that meets the first three conditions of the serial fusion rule, but violates the fourth condition. In this net, when a token arrives in place p_1, either transition t_1 or transition t_3 may fire one time unit later. If one applied the serial fusion rule

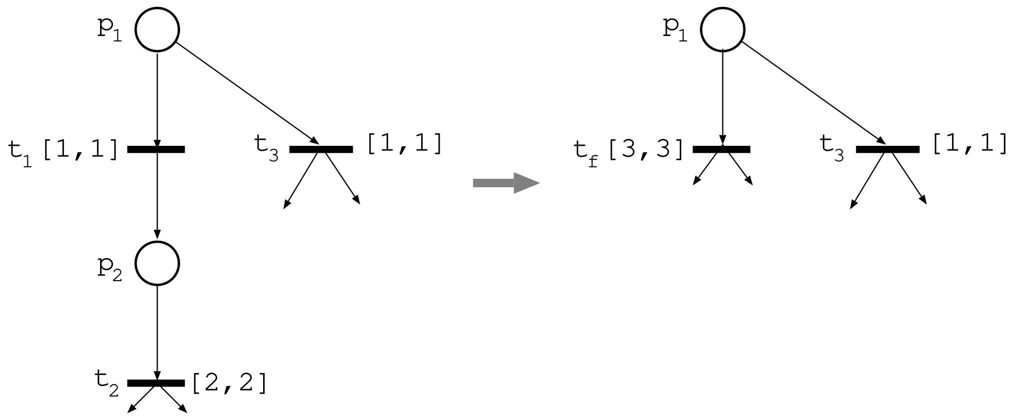

Figure 2: Illustration of necessity of fourth condition of the serial fusion reduction rule.

to obtain the fragment on the right, then, once a token arrives in p_1 in the reduced net, transition t_3 must fire. Thus, the nondeterministic choice between t_3's firing and t_f's firing would be lost in the reduced net.

Using the serial fusion rule, we can illustrate why we require that the underlying Petri net of a time Petri net be safe. Consider the application of the serial fusion rule to the net fragment on the left side of Figure 3, which would result in the fragment on the right side. Note that the unreduced time Petri net on the left is safe because the token deposited in place p_1 by the firing of transition t_4 is removed before transition t_5 fires. The underlying untimed Petri net is not safe, however. The net fragment shown on the right is unsafe as a time Petri net, since two tokens accumulate in p_1 before transition t_f fires. Since any practical application of our net reduction technique will consist of repeated applications of the reduction rules, it is crucial that each rule preserve safeness.

We now move on to prove that the serial fusion rule, when applied to time Petri nets whose underlying Petri nets are safe, does create equivalent nets.

Theorem 1 *Let time Petri net N_R be derived from time Petri net N by applying the serial fusion rule to transitions t_1 and t_2 and place p of N to obtain transition t_f in N_R. Then N_R is equivalent to N.*

Proof: We must show both that $L(N) - \{t_1, t_2\} \subseteq L(N_R) - \{t_f\}$ and that $L(N_R) - \{t_f\} \subseteq L(N) - \{t_1, t_2\}$. First we show that $L(N) - \{t_1, t_2\} \subseteq L(N_R) - \{t_f\}$, by proving the following (stronger) claim by induction.

Claim: For every schedule $\sigma \in L(N)$ there is a schedule $\sigma_R \in L(N_R)$ such that (a) $\sigma - \{t_1, t_2\} = \sigma_R - \{t_f\}$, (b) if each firing of t_1 in σ is followed by a firing

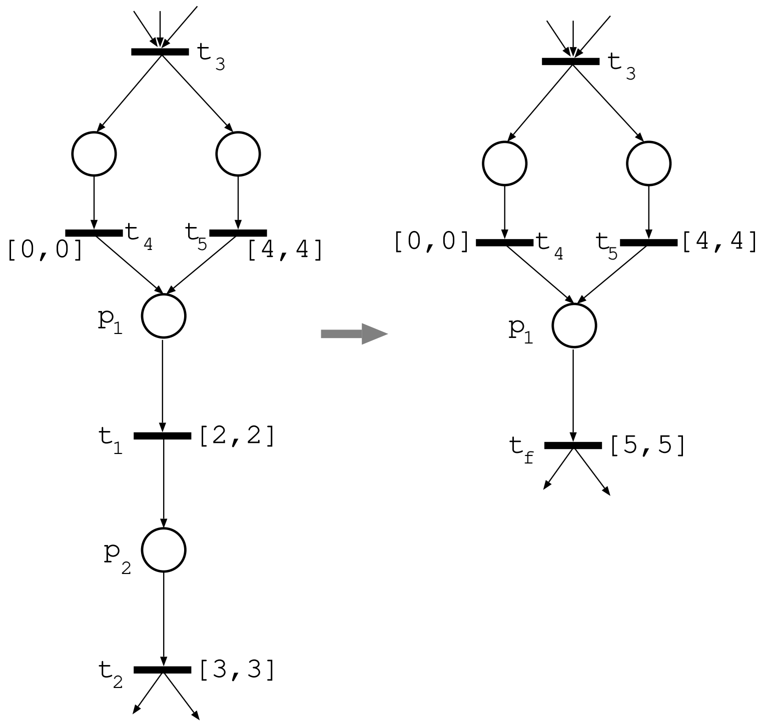

Figure 3: Illustration of requirement that underlying untimed Petri nets be safe.

of t_2, then $\mathcal{S}(N, \sigma) - \{t_1, t_2, p\} = \mathcal{S}(N_R, \sigma_R) - \{t_f\}$, and (c) if t_1 is enabled in $\mathcal{S}(N, \sigma)$, then t_f is enabled in $\mathcal{S}(N_R, \sigma_R)$ and the time since enablement of t_1 in $\mathcal{S}(N, \sigma)$ is the same as the time since enablement of t_f in $\mathcal{S}(N_R, \sigma_R)$.

The induction is on the number of firings of t_2 in schedule σ. Note that $N - \{t_1, t_2, p\} = N_R - \{t_f\}$. For the base case, let σ be a schedule containing no firing of t_2. If $t_1 \notin \sigma$, then $\sigma \in L(N - \{t_1, t_2, p\})$, so putting $\sigma_R = \sigma$ satisfies the claim.

Consider now the case where t_1 is in σ but $t_2 \notin \sigma$. Obviously part (b) of our claim is vacuously satisfied. Since N is safe, there must be only one firing of t_1 in σ. Thus let $\sigma = \sigma_1(t_1, \theta)\sigma_2$, where t_1 and t_2 do not appear in either σ_1 or σ_2. Note that t_1 cannot be enabled at the end of σ_2, because if it were, then the underlying untimed Petri net would not be safe. Thus (c) is vacuously satisfied. If $\mathcal{I}(t_2) = [0, 0]$, then $\mathcal{I}(t_1) = \mathcal{I}(t_f)$, so we can use $\sigma_1(t_f, \theta)\sigma_2$ as a schedule of N_R that satisfies part (a) of the claim.

Otherwise, if $\mathcal{I}(t_2) \neq [0, 0]$, use the schedule $\sigma_1\sigma_2$. This schedule obviously satisfies part (a) of the claim; we just need to be sure that it is a legal schedule for N_R. All transitions in σ_1 are clearly still fireable in N_R. As we will now explain in detail, transitions in σ_2 are fireable in N_R because of two key facts: transition t_f does not share any input places with other transitions; and transition t_1 has no output places other than p.

The only problem that could arise with schedule σ_2 in net N_R is that because t_f has a later firing interval than t_1, the tokens sitting in the input places of t_f after σ_1 has been completed could force some transition not in σ_2 to fire before the completion of σ_2. However, since t_1, and thus t_f, do not share input places with any other transition, the tokens in the input places of t_f cannot enable in N_R, at any time during σ_2, any transition that is disabled in N. Thus, σ_2 is feasible in N_R.

Now assume the claim is true for any firing schedule of N with up to k firings of t_2, and consider any schedule σ of N involving $k + 1$ firings of t_2. Since $\{p\} = {}^\bullet t_2 = t_1{}^\bullet$ and place p is initially unmarked, t_1 must fire once before each firing of t_2. Because the underlying net is safe, t_1 cannot fire twice without an intervening firing of t_2. Therefore σ must contain either $k + 1$ or $k + 2$ firings of t_1.

Consider the case where σ contains $k + 1$ firings of t_1. Write

$$\sigma = \sigma_1(t_1, \theta_{1,1})\upsilon_1(t_2, \theta_{1,2})\sigma_2(t_1, \theta_{2,1})\upsilon_2(t_2, \theta_{2,2}) \cdots \sigma_{k+1}(t_1, \theta_{k+1,1})\upsilon_{k+1}(t_2, \theta_{k+1,2})\sigma_{k+2},$$

where neither t_1 nor t_2 appears in any σ_j or υ_j. Put $\sigma' = \sigma_1(t_1, \theta_{1,1}) \cdots \sigma_{k+1}$. By the inductive hypothesis, there is a firing schedule σ'_R for N_R such that $\mathcal{S}(N, \sigma') - \{t_1, t_2, p\} = \mathcal{S}(N_R, \sigma'_R) - \{t_f\}$, and such that at the end of these two schedules the time since enablement of t_1 in N is the same as the time since enablement of t_f in N_R.

If transition t_1 does not share any of its input places, then its final firing in σ must be in the interval $\mathcal{I}(t_1)$ after its enablement, and transition t_2 must fire in the interval $\mathcal{I}(t_2)$ after $\theta_{k+1,1}$, so tokens must appear in all the output places of t_2 in the interval $\mathcal{I}(t_1) + \mathcal{I}(t_2)$ after the enablement of t_1. By part (c) of the inductive hypothesis, we conclude that all the times in that interval are precisely when tokens may appear in the equivalent output places of t_f in the reduced net N_R. Thus, we can again complete the schedule of N_r by appending $\upsilon_{k+1}(t_f, \theta_{k+1,2})\sigma_{k+2}$ to σ'_R, again satisfying part (a) of our claim. If $\mathcal{I}(t_2) = [0, 0]$, then $\mathcal{I}(t_1) = \mathcal{I}(t_f)$, we can complete the schedule for N_R by appending $\upsilon_{k+1}(t_f, \theta_{k+1,2})\sigma_{k+2}$ to σ'_R (because in this case all transitions in υ_{k+1} and t_2 fire at the same time). This also satisfies part (a) of our claim. Since tokens arrive in the same time interval in $t_2{}^\bullet$ and $t_f{}^\bullet$, this schedule satisfies parts (b) and (c) of our claim. The case of firing schedules containing $k + 2$ firings of t_1 but only $k + 1$ firings of t_2 is similar.

Now we show that $L(N_R) \subseteq L(N)$. This time we prove that for every schedule ρ of N_R, there is a schedule σ of N such that (if we take t_2 and t_f to be the same label) $\rho = \sigma - \{t_1\}$ and $\mathcal{S}(N_R, \rho) = \mathcal{S}(N, \sigma) - \{t_1, p\}$. Moreover, if the last transition of ρ enables t_f, then t_1 is enabled in $\mathcal{S}(N, \sigma)$ with the same enablement time. We proceed by induction on the number of firings of t_f in ρ.

The case where transition t_f never fires is clear. Now consider a schedule

$$\rho = \rho_1(t_f, \theta_1)\rho_2(t_f, \theta_2) \cdots \rho_k(t_f, \theta_k)\rho_{k+1}(t_f, \theta_{k+1})\rho_{k+2}$$

of N_R that involves $k + 1$ firings of t_f, where no ρ_j contains t_f. Let $\rho' = \rho_1(t_f, \theta_1)\rho_2(t_f, \theta_2) \cdots \rho_k(t_f, \theta_k)\hat{\rho}_{k+1}$, where $\hat{\rho}_{k+1}$ is ρ_{k+1} truncated immediately after the transition firing which enables the $(k + 1)$st firing of t_f. By the inductive hypothesis, there is a schedule σ' of N corresponding to ρ', and transition t_1 is enabled at the end of σ'.

Let l be the time from the end of ρ' to the firing of the last t_f in ρ. We can split l into two portions, $l = l_1 + l_2$ such that $l_1 \in \mathcal{I}(t_1)$ and $l_2 \in \mathcal{I}(t_2)$. The schedule σ' can be extended so that t_1 fires at time l_1 after the end of σ'. If t_1 does not share any of its input places, then this is clear. Otherwise, we know that $\mathcal{I}(t_f) = \mathcal{I}(t_1)$. Thus since t_f can fire at time l_1, so can t_1.

Thus, we can extend the schedule σ' to the schedule σ of N as follows. First we append the remainder of ρ_{k+1}, together with t_1 inserted at time l_1 after the end of σ'. To this we append $(t_2, \theta_{k+1})\rho_{k+2}$. Since ${}^\bullet t_f = {}^\bullet t_1$, if the last transition of ρ_{k+1} enables t_f, then it also enables t_1.

Our serial fusion rule preserves the safeness of the underlying untimed Petri net because this rule is a special case of Berthelot's post-fusion rule, which is known to preserve safeness [3]. □

3.2 Pre-fusion

This is an extension of Berthelot's transition pre-fusion rule to time Petri nets [3].

Reduction Rule 2 (Pre-fusion) *Preconditions: There exists a place p, a transition t_1, and a set of transitions S such that:*

1. *Place p is unmarked in the initial marking.*
2. *${t_1}^\bullet = \{p\}$. (Place p is the unique output place of t_1.)*
3. *Place $p \in {}^\bullet t$ for every $t \in S$.*
4. *Place p is disconnected from all transitions other than t_1 and those in S.*
5. *$({}^\bullet t_1)^\bullet = \{t_1\}$. (Transition t_1 does not share any of its input places.)*
6. *$\mathcal{I}(t_1) = [d, d]$ for some constant d.*
7. *Either $\mathcal{I}(t_1) = [0, 0]$, or, if $\mathcal{I}(t_1) = [d, d]$ with $d > 0$, then for every transition $t \in S$, for every $q \in {}^\bullet t$ with $q \neq p$, we have* nice-tree(t, q, d),

where nice-tree(t, q, d) *is defined by:*

1. *$q^\bullet = \{t\}$. (Transition t does not share input place q. In general, no input places are shared except for the place called p in the statement of the rule, which is shared among all transitions in S.)*
2. *$({}^\bullet q)^\bullet = \{q\}$. (Place q does not share any of its input transitions.)*
3. *For each $t' \in {}^\bullet q$, either $d \leq$ eft(t'), or for each place $q' \in {}^\bullet t'$,* nice-tree$(t', q', d -$ eft$(t'))$, *where* eft(t') *denotes the static earliest firing time of t'.*

Rule: Remove place p and transition t_1. Make every place in ${}^\bullet t_1$ an input place of every transition in S. If $\mathcal{I}(t_1) = [d, d]$ with $d \neq 0$, then add d to the time interval of every transition in S, and adjust the times of the transitions in the trees found by the nice-tree *condition as follows: For each $t \in {}^\bullet({}^\bullet S)$, subtract $m = \min(d, \text{eft}(t))$ from $\mathcal{I}(t)$. If $m < d$, then for each $t' \in {}^\bullet({}^\bullet t)$, subtract $\min(d - m, \text{eft}(t'))$ from $\mathcal{I}(t')$. Continue subtracting out times along the transitions in the tree in this fashion until the leaves are reached (which will occur when the full d has been subtracted out along each tree path starting at q).*

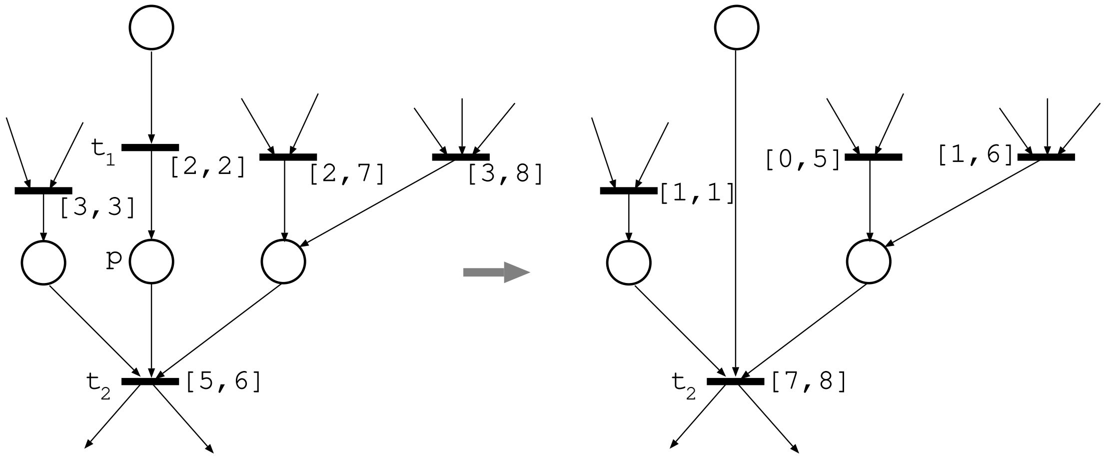

Figure 4: Example of pre-fusion reduction rule, with $S = \{t_2\}$.

The pre-fusion rule is illustrated in Figure 4.

The nice-tree condition says that for every transition $t_2 \in S$ we can find a tree embedded in the time Petri net with root t_2 and leaves that are transitions, such that the sum of the earliest firing times of the transitions along any root-to-leaf path is at least d. This tree must include as nodes all input places of any non-leaf transition except for the place p itself, and all output places of every transition in the tree.

Theorem 2 *Let time Petri net N_R be derived from time Petri net N by applying the pre-fusion rule to transition t_1, place p and transition set S of N. Then N is equivalent to N_R.*

Proof: We restrict our attention to the case where $S = \{t_2\}$; the case of $|S| > 1$ is similar. Let t_f be the transition of N_R that was formed by fusing t_1 and t_2 of N. The general skeleton of the proof is the same as the proof of Theorem 1. The part that is different is showing how the induction step works for each direction of the proof. We begin with this step for showing $L(N)|U \subseteq L(N_R)|U$, where U is the set of unaltered transitions.

Thus we need to show, given a schedule of N with $k+1$ firings of t_1, how to extend the equivalent schedule of N_R from k to $k+1$ firings of t_f. Let θ_1 be the time at which transition t_1 becomes enabled for the final time in the schedule of N, and let θ_2 be the time that p receives a token for the final time in the schedule of N. Let θ_3 be the latest time when a token arrives in any of ${}^\bullet t_2 - \{p\}$ for the final enablement of t_2. Note that if ${}^\bullet t_2 = \{p\}$, then the simpler serial fusion rule would apply. (See Figure 5.)

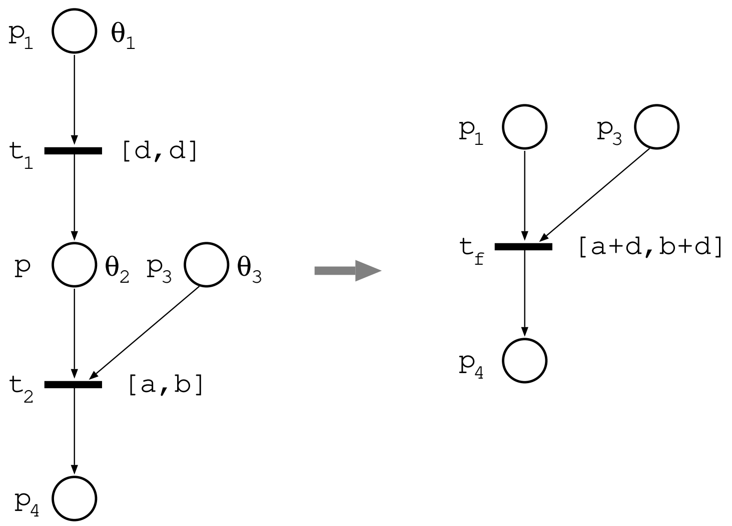

Figure 5: Illustration of pre-fusion reduction rule for proof of Theorem 2. Time of arrival of token in place p_i is θ_i.

We wish to compute τ_4, the interval of time in which t_2 fires, and show how to extend a schedule of N_R in such a way that t_f can fire exactly in the interval τ_4 as well. Let $\mathcal{I}(t_1) = [d, d]$ with $d \geq 0$. Now $\theta_2 = \theta_1 + d$, and

$$\begin{aligned}
\tau_4 &= \max(\theta_2, \theta_3) + \mathcal{I}(t_2) \\
&= \max(\theta_1 + d, \theta_3) + \mathcal{I}(t_2) \\
&= \begin{cases} \theta_1 + d + \mathcal{I}(t_2) & \text{if } \theta_1 \geq \theta_3 - d \\ \theta_3 + \mathcal{I}(t_2) & \text{otherwise} \end{cases}
\end{aligned} \tag{1}$$

Let τ_f be the interval of time in which t_f fires in N_R. The input places of t_f are ${}^\bullet t_1 \cup {}^\bullet t_2 \setminus \{p\}$ in N, and the time of arrival of the last token to appear in ${}^\bullet t_1$ is still θ_1. (The arrival at θ_1 in both nets is guaranteed by the inductive hypothesis in the omitted skeleton of the proof.) Let θ_3' be the last time of arrival of any token in the other input places of t_f. Then

$$\tau_f = \max(\theta_1, \theta_3') + \mathcal{I}(t_f). \tag{2}$$

Now if $d = 0$, then $\mathcal{I}(t_f) = \mathcal{I}(t_2)$, and all transitions other than t_1, t_2, and t_f are identical in N and N_R, so $\theta_3 = \theta_3'$. Thus we have that $\tau_f = \tau_4$ as desired.

If $d > 0$, then the pre-fusion rule has altered N_R so that $\theta_3' = \theta_3 - d$, because of subtraction along the "nice-tree." Also recall that $\mathcal{I}(t_f) = \mathcal{I}(t_2) + d$. By

Equation 2 this means

$$\tau_f = \max(\theta_1, \theta_3 - d) + (\mathcal{I}(t_2) + d).$$

Now if $\theta_1 \geq \theta_3 - d$, then $\tau_f = \theta_1 + \mathcal{I}(t_2) + d$. Otherwise, if $\theta_1 < \theta_3 - d$, then $\tau_f = \theta_3 - d + (d + \mathcal{I}(t_2)) = \theta_3 + \mathcal{I}(t_2)$. Thus the range of τ_f is precisely the range given for τ_4 in Equation 1 above.

The opposite direction, showing that schedules of N_R can be extended to equivalent schedules of N, follows in a symmetric fashion.

As with the serial fusion rule, the pre-fusion rule preserves safeness of the underlying untimed net because it is a special case of Berthelot's pre-fusion rule, which is known to preserve safeness [3]. $\square$

3.3 Post-fusion

To translate Berthelot's post-fusion rule to the timed setting, we need a more restrictive assumption about static firing intervals than we did for the serial fusion rule.

Reduction Rule 3 (Post-fusion) *Preconditions: There exist two nonempty sets of transitions A and B and a place p satisfying:*

1. *Place p is unmarked in the initial marking.*
2. *For every $a \in A$, place $p \in a^\bullet$, and $p \notin {}^\bullet a$.*
3. *For every $b \in B$, ${}^\bullet b = \{p\}$ and $p \notin b^\bullet$.*
4. *Place p is disconnected from all other transitions.*
5. *Every $b \in B$ has $\mathcal{I}(b) = [0, 0]$.*

Actions: Remove place p. For every $a \in A$ and $b \in B$, create a new transition t_{ab} with ${}^\bullet t_{ab} = {}^\bullet a$ and $t_{ab}{}^\bullet = (a^\bullet \setminus \{p\}) \cup b^\bullet$, and $\mathcal{I}(t_{ab}) = \mathcal{I}(a)$. Remove all transitions in A and B.

We give an example of the application of the post-fusion rule in Figure 6. All the preconditions not having to do with firing intervals are the same as for the untimed version of the reduction rule. The restriction that all the transitions in B have static firing intervals of $[0, 0]$ is necessary so the time at which output places of transitions in A (other than p) receive tokens is not changed.

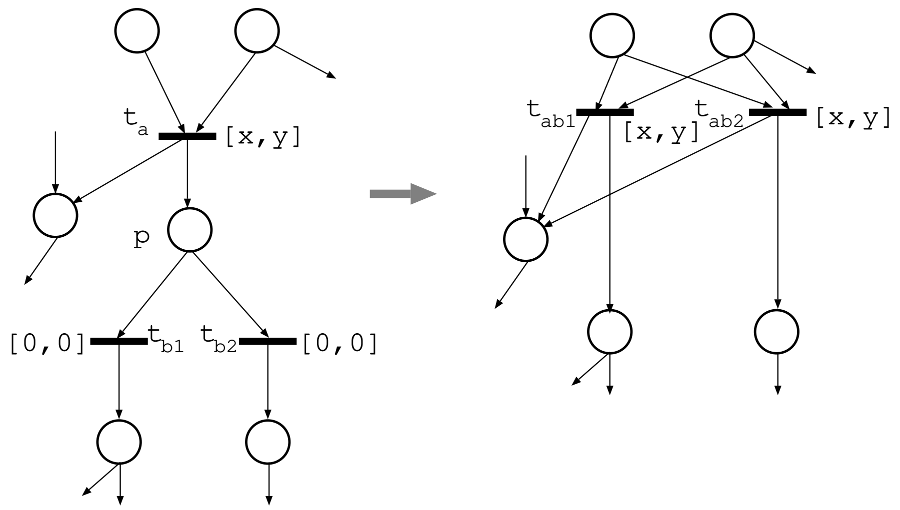

Figure 6: Illustration of post-fusion reduction rule. Here the set A is $\{t_a\}$ and the set B is $\{t_{b1}, t_{b2}\}$.

Theorem 3 *Let time Petri net N_R be derived from time Petri net N by applying the post-fusion rule to sets of transitions A and B and place p of N. Then N_R is equivalent to N.*

Proof Sketch: The argument is again similar to the proof of Theorem 1. Let S be the set of all the fused transitions added to the net. To show that $L(N) - (A \cup B) \subseteq L(N_R) - S$, we show how for any firing schedule σ of N we can find a corresponding schedule σ_R of N_R. The statement that can be proved by induction is that for each schedule σ of N there is a schedule σ_R of N_R such that (a) $\sigma - (A \cup B) = \sigma_R - S$, (b) for each firing of a transition $a \in A$ that is followed by the firing of a transition $b \in B$ in σ, schedule σ_R contains a firing of t_{ab} at the same time and position in the schedule as a, (c) for each transition $a \in A$ enabled at the end of σ, every transition t_{ab} for all $b \in B$ is enabled at the end of σ_R with equal amounts of time since enablement, and (d) if place p is unmarked at the end of σ, then $\mathcal{S}(N, \sigma) - A = \mathcal{S}(N_R, \sigma_R) - S$.

In the other direction we may show that each firing schedule in N_R can be transformed into a schedule of N by replacing each firing of a transition $t_{a,b}$ at a time θ with firings of t_a followed by t_b both at time θ.

Finally, this reduction rule preserves net safeness because it is a special case of Berthelot's post-fusion rule, which preserves safeness [3]. □

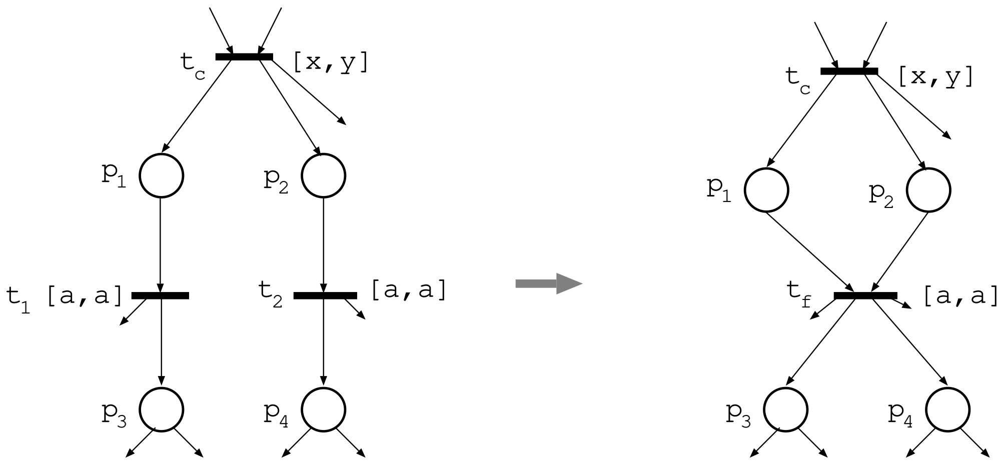

Figure 7: Illustration of lateral fusion reduction rule.

3.4 Lateral fusion of transitions

This rule is applied to a set S of transitions such that all transitions in S are preceded by a common transition not in S [3].[4] The transitions in S can be replaced by a unique transition, t_f. Figure 7 illustrates the case in which two transitions, t_1 and t_2, are preceded by the common transition t_c. We now give the formal definition of the lateral fusion rule.

Reduction Rule 4 (Lateral fusion) *Preconditions: A set of transitions S is laterally fusible into a transition t_f if there is a transition $t_c \notin S$, and a distinct place p_i for each t_i in S such that:*

1. *For every place p_i we have ${}^\bullet p_i = \{t_c\}$.*
2. *For every place p_i we have $p_i{}^\bullet = \{t_i\}$.*
3. *For every transition $t_i \in S$ we have ${}^\bullet t_i = \{p_i\}$.*
4. *All places p_i have the same initial marking.*

[4]Berthelot actually gives two versions of this rule. The first version is similar to the rule we discuss here. In the second, symmetric version, the common transition t_c follows the set S of transitions. In order to preserve net liveness, the second version requires an additional precondition that Berthelot fails to mention [3]. We do not discuss this version of the rule, because it cannot be extended to time Petri nets even after the additional precondition is included.

5. *All transitions in S have the same static firing interval, and this interval is of the form $[a, a]$.*

Actions: Every transition in S is replaced by a single transition, t_f, that has the same static firing interval as the replaced transitions. The input and output sets of t_f are the union of the corresponding sets for each transition in S.

Remark: After this rule has been applied, the parallel redundant place rule (see Section 4.1) will then apply to remove all but one of the places p_i. For instance, in the example given in Figure 7, either one of the two places p_1 and p_2 could be removed.

The lateral fusion preconditions that do not involve time are essentially the same as for the untimed version of this rule. It is obvious that the transitions to be fused must have the same static firing interval. We now explain why the transitions in set S must have static firing "points" rather than intervals. Consider the example in Figure 8. The net fragment on the left, with $S = \{t_1, t_2\}$, satisfies all other preconditions of the lateral fusion rule. In this net, transition t_3 is fireable whenever transition t_2 fires at least one time unit after transition t_1. If the reduction rule is applied, the net on the right is obtained. In this net, however, transition t_3 can never be fired.

Now we prove that the lateral fusion rule with this restriction does result in an equivalent net.

Theorem 4 *If time Petri net N_R is derived from time Petri net N by applying the lateral fusion rule, then N_R is equivalent to N.*

Proof: Let N_R be derived from N by applying the lateral fusion rule to all the transitions in some set S of transitions of N, and let t_f be the fused transition of N_R. For each $t_i \in S$, let p_i be the place of N specified by the preconditions of the lateral fusion rule. We first prove that $L(N) - S \subseteq L(N_R) - \{t_f\}$, and then will prove that $L(N_R) - \{t_f\} \subseteq L(N) - S$.

In general, all places p_i receive tokens at the same time (and either all or none are marked in the initial marking), so all the transitions in S must always fire at the same global time. Let us remark that, since the underlying net is safe, no transition in S can be enabled for its $(k+1)$st firing until every transition in S has fired k times.

To show that $L(N) - S \subseteq L(N_R) - \{t_f\}$, we prove that for each schedule σ of time Petri net N, there is a schedule σ_R of net N_R such that (a) $\sigma - S = \sigma_R - \{t_f\}$, (b) if all the transitions in S are enabled at the end of σ in N, then t_f is enabled at the end of σ_R in N_R, with the same time since enablement, and (c) if all transitions

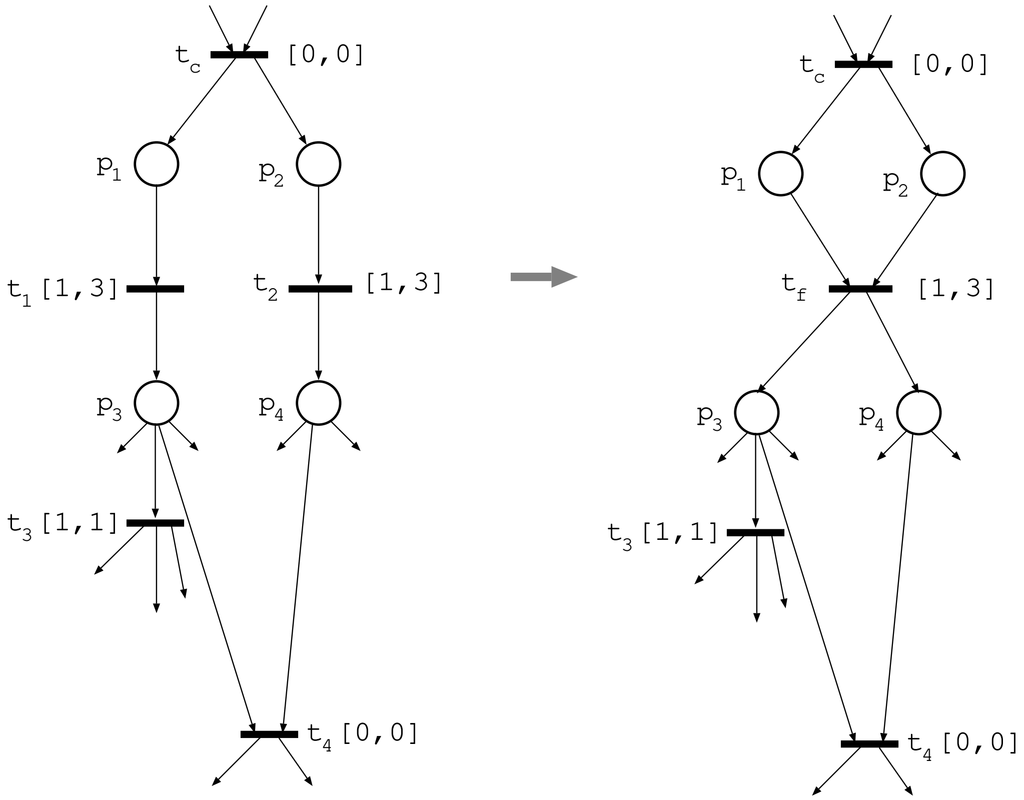

Figure 8: Example illustrating why transitions affected by lateral fusion rule must have point, rather than interval, delays.

in S appear the same number of times in σ, then $\mathcal{S}(N, \sigma) - S = \mathcal{S}(N_R, \sigma_R) - \{t_f\}$. The proof is by induction on the maximum number of times that any transition in S appears in the firing schedule σ.

The base case is that no transition in S appears in σ. Since transitions not in S are not affected by this reduction rule, they are fireable in N_R whenever they are in N. Consequently, the schedule σ_R, which is identical to σ, is feasible in net N_R, satisfying (a) and (c). Part (b) follows from the fact that ${}^\bullet t_f = {}^\bullet S$.

Now let σ be a schedule of N such that the maximum number of firings of any transition in S is $k + 1$. Let σ' be σ truncated immediately before the first firing of any S transition for the $(k + 1)$st time, and let $\sigma = \sigma'\omega$. By our remarks above, every transition in S must appear exactly k times in σ'. By the inductive hypothesis, there is a firing schedule σ'_R of N_R such that $\sigma' - S = \sigma'_R - \{t_f\}$, and $\mathcal{S}(N, \sigma') - S = \mathcal{S}(N_R, \sigma'_R) - \{t_f\}$. Moreover, the time since enablement of t_f at the end of σ'_R is the same as the common time since enablement of all transitions in S at the end of σ'.

Schedule ω begins with the firing of some transition in S. Let ω_R be ω with that transition firing replaced by a firing of t_f, and all other transitions from S deleted. Now $\sigma - S = \sigma'_R\omega_R - \{t_f\}$, so schedule $\sigma'_R\omega_R$ satisfies part (a) of our claim.

For (c), if schedule σ contained exactly $k + 1$ firings of every transition in S, then in ω every transition in S fires exactly once at the same time, say θ. The firing of t_f in N_R at time θ will put tokens in the same places as the firing of all S transitions in N. Therefore, since $\mathcal{S}(N, \sigma') - S = \mathcal{S}(N_R, \sigma'_R) - \{t_f\}$, it must be that $\mathcal{S}(N, \sigma) - S = \mathcal{S}(N_R, \sigma'_R\omega_R) - \{t_f\}$. For (b), notice that if all transitions in S are enabled at the end of σ, then σ contained exactly $k+1$ firings of every transition from S, so by (c), which we just showed, $\mathcal{S}(N, \sigma) - S = \mathcal{S}(N_R, \sigma'_R\omega_R) - \{t_f\}$. Since ${}^\bullet t_f = {}^\bullet S$, this satisfies (b).

We now consider the opposite case, that is, given a schedule $\sigma_R \in N_R$ find a schedule $\sigma \in N$ such that $\sigma_R - \{t_f\} = \sigma - S$. This case is very simple, because whenever t_f is fireable in N_R all S transitions are fireable in N. Thus, σ can be obtained from σ_R by replacing the firing of t_f with the firing of all transitions in S in an arbitrary order at the same time as the firing of t_f. The firing of all S transitions results in the same marking as the firing of t_f in N_R.

Our lateral fusion rule is a special case of the corresponding rule by Berthelot, which is known to preserve safeness [3]. We conclude that our rule preserves safeness of the underlying untimed net. $\square$

4 Reduction rules removing places

In this section, we present a number of rules that can be used to remove places from a net.

4.1 Redundant places

We begin by defining the notion of a redundant place. Intuitively, place p is redundant in Petri net N if p does not have any effect on the behavior of N. Thus, if p is removed from N along with the arcs incident on p, the reachability graph of the resulting net is the same as N's reachability graph.

Our first rule was originally defined by Berthelot [3]. This rule translates without alteration from the untimed to the timed case. (Incidentally, Berthelot presents two other rules for removing places, but one depends on behavioral as well as structural properties of the Petri net, and the other fails to preserve safeness, so they are not of interest to us.)

Reduction Rule 5 (Redundant Places) *Preconditions: A place p and a set (possibly empty) of places I not containing p and a weighting function $w : I \cup \{p\} \to \mathbb{N} - \{0\}$ such that*

1. $\sum_{q \in I} w(q) M_0(q) \leq w(p) M_0(p)$.
2. *For every $t \in p^\bullet$, $\sum_{q \in I \cap {}^\bullet t} w(q) \geq w(p)$.*¶
3. *For every transition t, we have $\sum_{q \in I} w(q) C(q,t) \leq w(p) C(p,t)$, where the function $C(q,t)$ is 0 if either q is unconnected to t or q is both an input and output place of t, 1 if $q \in t^\bullet$, and -1 if $q \in {}^\bullet t$.*

Rule: Remove place p and all arcs in and out of it.

Berthelot's proof that this rule preserves the language of the net (as well as safety and liveness) carries over directly to time Petri nets [2, 3]. However, this rule by itself is not useful for an automated reduction system, because there is no efficient way to find the set I or the weighting function w.

However, it is often possible to find commonly occurring special cases of this rule [9, 20]. Two examples follow.

¶In fact, the condition $\sum_{q \in I \cap {}^\bullet t} w(q) \geq (1 - M_0(p)) w(p) + \sum_{q \in I} w(q) M_0(q)$, which may be weaker when p is initially marked, is sufficient.

Reduction Rule 6 (Parallel Redundant Places) *If two places have identical input and output transitions, and identical initial markings, then either one of these places can be removed [20].*

Reduction Rule 7 (Removal of useless end places) *An initially unmarked place with no output transitions may be removed.*

4.2 Unnecessary begin places

We conclude our list of reduction rules with some straightforward rules for removing places with no inputs.

Reduction Rule 8 (Removal of empty begin places) *If place p has no input transitions and is initially unmarked, then it may be removed together with all its output transitions.*

Theorem 5 *If net N_R is obtained from net N by an application of the removal of empty begin places rule at place p of N, then N_R is equivalent to N.*

Proof: Since place p can never contain a token, no transition in $p^\bullet$ can ever fire in N. □

Reduction Rule 9 (Unnecessary marked begin places) *Preconditions: A place p with no input transitions such that*

1. *Place p is initially marked.*
2. *Place p has exactly one output transition, t.*
3. *There is an initially unmarked place $p' \in {}^\bullet t$ such that, if we view the time Petri net as a directed graph, then p' is not in any cycle, and no transition from which p' is reachable is in any cycle.*

Actions: Delete place p.

Observe first that this rule can be applied in polynomial time. Precondition 3 can be computed by computing the transitive closure of the time Petri net.

Although this rule seems highly specialized, we have occasionally obtained a significant increase in the speed of the analysis of a time Petri net by applying it.

Theorem 6 *Let time Petri net N_R be derived from time Petri net N by applying the unnecessary marked begin places reduction rule to place p. Then N is equivalent to N_R.*

Proof: Let $p^\bullet = \{t\}$. In the original net, p ensures that t can fire at most once, at $\mathcal{I}(t)$ after all t's other input places receive tokens. Once p is removed, t can fire $\mathcal{I}(t)$ after *each time* all its remaining input places receive tokens. However, Precondition 3 ensures that t has another input place p' that can contain a token at most once. $\square$

5 Conclusions and future work

In this paper, we have presented a set of reduction rules for time Petri nets. Our work is based on the work of Berthelot for ordinary Petri nets [2, 3]. Berthelot's results have been successfully applied to automated deadlock analysis in Ada programs [9]. In a similar fashion, the ultimate goal of our reduction technique is to facilitate real-time program analysis.

The introduction of time often complicates the applicability of reduction rules. In general, our rules require additional preconditions beyond the preconditions of Berthelot's corresponding rules. An exception is the set of conditions required for a place to be redundant in a Petri net. As we noted in Section 4, these conditions and the reduction rules that are based on these conditions translate directly from ordinary Petri nets to time Petri nets.

Despite the complications imposed by the introduction of time, we have reason to believe that the net reduction technique presented here will be as successful in real-time analysis as it is in concurrency analysis. The reduction rules discussed in this paper have been applied to real-time system analysis using STP nets as the underlying system model [7]. Our reduction rules can be applied directly to STP nets because these nets are a special case of time Petri nets.

Preliminary experiments with a toolset for the analysis of STP nets have shown that net reduction can be quite effective even when applied to hand-generated nets [7]. More importantly, net reduction has always been effective when we applied it to nets generated automatically by our translator [6]. There are two reasons for this phenomenon. First, our method for automatic translation is aimed at modeling all task interactions contained in an Ada program, whereas the goal of reduction is to preserve only properties of interest in a net model. Thus, task interactions that do not affect properties of interest in an STP net can be eliminated during net reduction. Second, as we mentioned earlier, our automatically-generated nets consist of subnets modeling different statements contained in the underlying Ada code. Many reductions become possible when the translator connects different subnets in order to build a complete net model of the code. This situation is similar to the case of object code generated by a compiler—the resulting code is unnecessarily large, and therefore one can apply suitable code optimizations.

In the future we plan to investigate specializing the reduction technique we discussed here for the analysis of code written in Ada (and perhaps other high-level programming languages). We expect that this will lead to customizations of Rule 6 aimed at exploiting Ada-specific information to speed up application of the resulting rules. We also plan to investigate efficiency issues for the reduction activity. In particular, we will examine the effects of the order of application of the rules on the overall efficiency of the reduction. Finally, we plan to run experiments with examples of real-time programs, and to compare our empirical results with results obtained using alternative approaches to real-time analysis.

References

[1] P. Azema, G. Juanole, E. Sanchis, and M. Montbernard. Specification and verification of distributed systems using Prolog interpreted Petri nets. In *Proceedings of the Seventh International Conference on Software Engineering*, pages 510–518, Orlando, Florida, 1984.

[2] G. Berthelot. *Transformations et analyse de réseaux de Petri: Application aux protocoles*. PhD thesis, Université de Paris 6, 1983.

[3] G. Berthelot. Checking properties of nets using transformations. In G. Rozenberg, editor, *Advances in Petri Nets 1985*, pages 19–40. Springer-Verlag, 1987.

[4] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. Softw. Eng.*, 17(3):259–273, Mar. 1991.

[5] U. Buy and R. Sloan. A Petri-net-based approach to real-time program analysis. In *Proc. Seventh Internat. Workshop on Software Specification and Design*, pages 56–60, Dec. 1993.

[6] U. Buy and R. H. Sloan. Automatic real-time analysis of Ada tasking programs for embedded systems. Submitted for publication to *Real Time Systems*. Available as Technical Report 95-1, Department of Electrical Engineering and Computer Science, University of Illinois at Chicago.

[7] U. Buy and R. H. Sloan. Analysis of real-time programs with simple time Petri nets. In *Proc. 1994 Internat. Sympos. on Software Testing and Analysis*, pages 228–239, 1994.

[8] J. Desel. Reduction and design of well-behaved concurrent systems. In J. C. M. Beaten and J. W. Klop, editors, *CONCUR '90: Theories of Concurrency: Unification and Extension*, LNCS 458, pages 166–181. Springer-Verlag, 1990.

[9] S. Duri, U. Buy, R. Devarapalli, and S. M. Shatz. Application and experimental evaluation of state space reduction methods for deadlock analysis in Ada. *ACM Trans. Software Engineering and Methodology*, 3(4):340–380, Oct. 1994.

[10] J. Esparza and M. Silva. Top-down synthesis of live and bounded free choice nets. In *Proceedings of the 11th Conference on Petri Nets*, pages 63–83, Paris, France, 1990.

[11] G. Estrin, R. S. Fenchel, R. R. Razouk, and M. K. Vernon. SARA (System ARchitects Apprentice): Modeling, analysis, and simulation support for design of concurrent systems. *IEEE Trans. Softw. Eng.*, 12(2):293–311, Feb. 1986.

[12] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezzé. A general way to put time in Petri nets. In *Proc. 5th Internat. Workshop on Software Specifications and Design*, pages 60–67, Pittsburgh, Pennsylvania, May 1989.

[13] P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In K. G. Larsen and A. Skou, editors, *Computer-Aided Verification: 3rd Internat. Conf., CAV '91*, LNCS 575, pages 332–342. Springer-Verlag, 1992.

[14] S. Haddad. A reduction theory for coloured Petri nets. In G. Rozenberg, editor, *Advances in Petri nets*, volume 424 of *Lecture Notes in Computer Science*, pages 209–235. Springer Verlag, New York, New York, 1990.

[15] P. Merlin and D. Faber. Recoverability of communication protocols—implications of a theoretical study. *IEEE Trans. Communications*, COM-24(9):381–404, Sept. 1976.

[16] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

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

[18] S. M. Shatz, K. Mai, C. Black, and S. Tu. Design and implementation of a Petri net-based toolkit for Ada tasking analysis. *IEEE Transactions on Parallel Distributed Systems*, 1(4):424–441, Oct. 1990.

[19] P. H. Starke. Reachability analysis of Petri nets using symmetries. *Syst. Anal. Model. Simul.*, 8:293–303, 1991.

[20] S. Tu, S. M. Shatz, and T. Murata. Applying Petri net reduction to support Ada-tasking deadlock analysis. In *Proc. Eleventh Internat. Conference on Distributed Computing Systems*, pages 96–103, Paris, France, 1990.

[21] A. Valmari. A stubborn attack on state explosion. In E. M. Clarke and R. P. Kurshan, editors, *Computer-Aided Verification: 2nd Internat. Conf., CAV '90*, LNCS 531, pages 156–165. Springer-Verlag, 1991.

[22] T. Yoneda, A. Shibayama, B.-H. Schlingloff, and E. M. Clarke. Efficient verification of parallel real-time systems. In C. Courcoubetis, editor, *Computer-Aided Verification: 5th Internat. Conf., CAV '93*, LNCS 697, pages 321–332. Springer-Verlag, 1993.