Stubborn Sets for Real-Time Petri Nets

Robert H. Sloan* Ugo Buy†
Dept. of Electrical Engineering and Computer Science
University of Illinois at Chicago

Abstract

The stubborn-set method attempts to reduce the number of states that must be explicitly enumerated during reachability analysis of a concurrent system, while preserving crucial concurrency properties in the reduced state space. Here we extend the method to the analysis of certain timed models. We also prove that timing properties of interest, such as minimum and maximum delays between events, are preserved in the reduced model. Finally, we report on some experimental results that we have obtained with our extension. These results show that real-time stubborn sets can often provide significant reductions in state space size.

1 Introduction

Both formal analysis and testing of real-time programs are very difficult, and yet at the same time very important, since there are so many safety-critical real-time programs. Formal analysis methods are conceptually appealing because these methods, unlike testing, can *guarantee* that a real-time system satisfies given timing properties. Of course, in order to effectively support the development of real-time software, formal methods must lead to the

*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`.

creation of practical tools that automatically perform significant portions of an analysis.

Our approach to the automatic analysis of real-time software uses an extension of Petri nets for modeling real-time systems. Our *Simple Time Petri nets* [4], or STP nets, use a relatively simple notion of time compared to other time models, but are especially well-suited for efficient real-time analysis.

Our general approach is based on the following paradigm: First, a real-time program is translated into an STP-net model. If possible, this net is then shrunk to a smaller net with identical behavior [13]. Next, the net's state space is constructed. This state space is represented as a reachability graph whose nodes correspond to reachable program states, and whose arcs correspond to state transitions. Finally, suitable graph algorithms are applied to the reachability graph in order to answer typical questions about the program's real-time behavior (e.g., maximum and minimum delays between given events in a real-time program). As with most state-space-based methods, our approach suffers from the *state explosion* problem, meaning that the number of reachable states tends to be quite large even for relatively small real-time programs. The overall goal of this research project is to develop many different heuristic methods for limiting the state explosion problem, and to use as many as possible in combination [3, 4].

Here we develop in detail and prove the correctness of one such technique, *real-time stubborn sets*. In addition, we also report on some experimental results that we have obtained from the PARTS toolkit (Petri-net-based Analyzer of Real-Time Software), which implements the most relevant aspects of our approach. We will also compare and contrast our work to similar work by Yoneda et al. [16].

Valmari originally defined the stubborn set method for the analysis of (untimed) concurrent programs [15]. In the past, we have obtained dramatic speedups in Petri-net-based concurrency analysis of Ada programs by integrating various state space reduction methods, including stubborn sets [6]. Our goal is to do something similar for real-time analysis by extending the techniques we used for concurrency analysis to real-time analysis [2, 4]. Real-time stubborn sets is one of several techniques that we use to enhance the tractability of reachability-based real-time analysis in our PARTS system.

Stubborn sets are one of several *partial-order* methods for reduced state-space explosion. These methods consider only one particular ordering of a set of n simultaneous events, instead of all $n!$ possible orderings. Each particular partial-order method gives some algorithm for finding portions of the state space where this may be done without losing any relevant information about the state space.

In the case of stubborn sets, in each state, we find a set of transitions, hopefully small, that are *stubborn*. The method tells us that if we meet certain conditions, instead of exploring all fireable transitions, we need to explore only those fireable transitions in the stubborn set. Of course, answers to relevant questions about program behavior are the same in the reduced and the full state space.

A weaker version of our real-time stubborn set method can be derived directly from Valmari's work on stubborn sets for variable transition systems [15], because one can view STP nets as a particular kind of variable transition system. Here we make two main theoretical contributions. First, we exploit timing properties of STP nets in order to obtain smaller stubborn sets than one obtains by simply viewing them as variable transition systems. Secondly, we prove that the stubborn set techniques give proper answers to timing queries, as well as to untimed safety and liveness queries. (Valmari's method turns out to give correct answers to timing queries, but he did not consider real-time analysis at all, and he proves only that his method preserves untimed concurrency properties.)

The work we present here was developed independently of and slightly later than Yoneda et al.'s partial-order method for a timed version of Petri nets [16]. They use a different partial order method, and their work has two other important differences from ours. They use time Petri nets [9], which have a much richer notion of time than our STP nets. However, their method is inherently $\Omega(n^3)$, where n is the number of processes, whereas our method is linear in the number of processes. That is to say, even if the reduced state space constructed by the stubborn set has size $\Theta(n)$, Yoneda's method will still require $\Omega(n^4)$ time to construct it. We discuss these matters at length in Section 5 below.

The rest of this paper is organized as follows. In the next section we present the formal definitions of STP nets and stubborn sets for them. In Section 3, we present the theorems that show that our new real-time stubborn-set method is valid, and in Section 4 we present some empirical results showing the speedup in analysis we have obtained by using stubborn sets. In Section 5 we discuss other timed Petri net models and, in particular, compare our results to the method proposed by Yoneda et al. [16].

2 Preliminaries

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 [12], and in the excellent survey article by Murata [10].

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* 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.

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 $t \in T$.

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

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

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

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. If firing transition t takes the net from marking M to M', we write $M - t \rightarrow M'$.

2.2 STP nets

Many timed extensions of Petri nets have been proposed. We believe, however, that most of those extensions are too general to permit efficient use of partial-order methods for state space reduction. We discuss this matter further in Section 5 below. We will restrict our attention to *Simple Time Petri Nets*, or *STP nets* [2], which are a special case of time Petri nets [1, 9]. An STP net is a 5-tuple (P, T, F, M_0, D) where (P, T, F, M_0) is an ordinary Petri net, and D is a length $|T|$ vector of nonnegative rational numbers that associates a *static delay*, $\tau(t)$, with each transition t. In this article, we impose the additional technical restriction that there is no multiple enablement of the underlying ordinary Petri net; that is, the Petri net has no reachable states where some transition has at least two tokens in every one of its input places.[2]

Static delays change the behavior of an STP net with respect to an ordinary Petri net in the following way. Whenever a transition t becomes enabled (i.e., when all its input places have a token), that transition must fire precisely $\tau(t)$ time units later, unless it becomes disabled by the removal of

[2]In theory, verifying this condition for an *arbitrary* net might require building the net's entire state space. In practice, if our system is to be of any use, then the STP nets being analyzed will be the output of an automatic translation from some programming language such as Ada or Concurrent C. It is easy to build a translator that enforces this condition.

tokens from an input place in the meantime. Observe that ordinary, untimed Petri nets are are now a special case of STP nets with all static delays being zero. At any given global time, each transition t also has a *dynamic delay*, $d(t)$, which tells how much time is now left until t fires. Note that for every transition t, at all times we have $0 \leq d(t) \leq \tau(t)$.

The state of an STP net is a pair (M, I), where M is the marking of the underlying untimed Petri net, and I is a vector containing the dynamic firing time of each transition enabled by M and $\perp$ for disabled transitions. The marking M of state $s = (M, I)$ is denoted $\mathrm{mark}(s)$. The initial state of STP net (P, T, F, M_0, D) consists of M_0 and a vector containing the static firing interval of each transition enabled by M_0 and $\perp$ for those transitions disabled by M_0.

We say that a transition t is *fireable* in state (M, I) if it is enabled (M associates at least one token with all of ${}^\bullet t$), and it is legal to fire t next. This will be true if and only if transition t has minimal dynamic delay among all enabled transitions.

Let s' be the state derived from firing fireable transition t in state s. We write $s - t \rightarrow s'$, and also $s \Rightarrow s'$. State s' has the marking obtained by considering only the underlying untimed Petri net; its vector of dynamic delays is as follows. For every transition u that was disabled in s but is enabled in s', transition u has dynamic delay equal to $\tau(u)$. If transition $u \neq t$ was enabled in s and remains enabled in s', then the dynamic delay of u in state s' is its previous dynamic delay minus the dynamic delay of transition t in state s. If u is not enabled in s', then in s' we set $d(u) = \perp$. Finally, if t itself is again enabled in s', then $d(t) = \tau(t)$; otherwise $d(t) = \perp$.

As usual, we say that *state s leads to state s'*, denoted $s \overset{*}{\Rightarrow} s'$, whenever there exists a finite sequence of states $s_1, s_2, \ldots, s_n$ such that

$$s \Rightarrow s_1 \Rightarrow s_2 \Rightarrow \cdots \Rightarrow s_n \Rightarrow s'.$$

The *state space* of STP net N is the set of all states s of N such that $s_0 \overset{*}{\Rightarrow} s$, where s_0 is the initial state of N.

A *(firing) schedule of STP net N from state s_0* is a finite sequence of ordered pairs (t_i, s_i), for $1 \leq i \leq n$, such that for all $1 \leq i \leq n$ we have $s_{i-1} - t_i \rightarrow s_i$. We will often denote such a schedule by

$$s_0 - t_1 \rightarrow s_1 - t_2 \rightarrow \cdots - t_{n-2} \rightarrow s_{n-2} - t_{n-1} \rightarrow s_{n-1} - t_n \rightarrow s_n.$$

The *time* of such a sequence is the sum of the dynamic delays of all the transitions fired.

Let us say that transition t is *current* in state s if either t is fireable or t has a static delay of 0. Otherwise t is *delayed*. The idea is that any transition that is delayed cannot possibly fire until some positive time after the currently fireable transition(s).

2.3 Stubborn sets

In this subsection we define real-time stubborn sets.

Definition. We say that the set A of transitions of STP net $N = (P, T, F, M_0, D)$ is *semi-stubborn* if every $t \in A$ is current and in addition:

1. If t is not enabled, then for some empty $p \in {}^{\bullet}t$, each transition in ${}^{\bullet}p$ is either in A or delayed.

2. If t is fireable, then for each $t' \in T - A$, either ${}^{\bullet}t \cap {}^{\bullet}t' = \emptyset$, or t' is

delayed.

By this definition, the empty set is semi-stubborn; therefore we need the following.

DEFINITION. A set of transitions is *stubborn* if it is semi-stubborn and contains at least one fireable transition.

We assume that for every STP net N we have a *stubborn set generator*, a function TS that will pick out a stubborn set for us. Formally TS is a map from the state space of N to the power set of the transition set of N such that $TS(s)$ is a stubborn set at state s. If transition $t \in TS(s)$, then we call *t stubborn in state s*, or simply stubborn when s is clear from context.

The success of the stubborn-set method can vary idiosyncratically with the choice of TS, although in general smaller stubborn sets are better than larger stubborn sets [14]. We give one method for generating stubborn sets below, in Section 4, where we present some experimental results.

Fix an STP net N and a stubborn set generator TS. Our intention is to explore only the fraction of N's state space reached by firing only stubborn fireable transitions. We say *state s leads to state s' in the reduced state space of N*, denoted $s \overset{\mathrm{ST}}{\Rightarrow} s'$, if there is some transition $t \in TS(s)$ such that $s - t \rightarrow s'$. The *reduced state space of N* is the set of all states s such that there exists a finite sequence of states $s_1, s_2, \ldots s_n$ such that

$$s_0 \overset{\mathrm{ST}}{\Rightarrow} s_1 \overset{\mathrm{ST}}{\Rightarrow} s_2 \cdots \overset{\mathrm{ST}}{\Rightarrow} s_n \overset{\mathrm{ST}}{\Rightarrow} s$$

where s_0 is the initial state of N.

2.4 Visible transitions

When we wish to verify timing properties, we will in general be asking questions of the form "What is the longest time (or shortest time) path from given STP net state s_0 to any state whose marking is in some specified set of markings $\mathcal{C}$?" We must obviously be careful that when we generate only part of the net's state space that we do not discard the desired path. This concern motivates the definition of vis($\mathcal{C}$) that we now develop.

A *specification* of the set $\mathcal{C}$ of markings of STP net $N = (P, T, F, M_0, D)$ is a logical combination of (in)equalities over the number of tokens in places of N. Equalities and inequalities must be of the form number of tokens in place $p_i \gtreqless k$ for some integer k $geq 0$. For instance,

$$(p_1 \geq 1 \text{ or } p_3 \geq 1) \text{ and } p_7 = 0$$

is a specification of a particular set of markings.

For any specified set of markings $\mathcal{C}$ we assume we are provided with a set of *visible transitions*, denoted vis($\mathcal{C}$). The set vis($\mathcal{C}$) may be any superset of all the transitions in ${}^\bullet p_i \cup p_i{}^\bullet$ for all places p_i mentioned in some specification of $\mathcal{C}$. (The set vis($\mathcal{C}$) can be, but does not have to be, the smallest such set.) This definition is essentially similar to Valmari's definition of visibility [15], although the settings are different.

We will call any transition $t \in$ vis($\mathcal{C}$) *visible*, and any transition $t' \notin$ vis($\mathcal{C}$) *invisible*. For any schedule σ of N, we use vis(σ) to denote the ordered sequence of visible transitions of σ.

2.5 Comparison to Valmari's definition of stubborn sets

Valmari's definition of stubborn sets [15] does not distinguish between current and delayed transitions, since his stubborn sets were invented for untimed analysis. As a result, if we view STP nets as variable transition systems and apply Valmari's definition, we get larger stubborn sets. In particular, if a stubborn set includes fireable transition t, then it must also include *all* transitions that share any input places with t. With our technique, we do not include any transition that is delayed.

Of course, Valmari did not consider real-time analysis at all, so the proofs of correctness of timing analysis we present below are new, even if one uses the definition of stubborn sets derived directly from variable transition systems.

3 Technical results

We begin with a technical lemma that says that any schedule that takes nonzero time after its first transition must include a stubborn transition.

Lemma 1 *Fix an STP net $N = (P, T, F, M_0, D)$, reachable state s_0, and a schedule of N*

$$\sigma = s_0 - t_1 \rightarrow s_1 - t_2 \rightarrow \cdots - t_{n-2} \rightarrow s_{n-2} - t_{n-1} \rightarrow s_{n-1} - t_n \rightarrow s_n$$

such that the time of σ is strictly greater than the dynamic delay of transition t_1 in state s_0. If A is any stubborn set of transitions of state s_0, then there must be a transition in $A \cap \sigma$.

Proof Assume to the contrary that $A \cap \sigma = \emptyset$, and let t be any element of A that is fireable in state s_0. Let t_i be the first transition after t_1 with a nonzero dynamic delay. In state s_{i-1}, transition t must no longer be fireable; otherwise firing time constraints would force it to fire before t_i could fire. Since t was fireable in state s_0, some transition t_j with $1 \leq j < i$ must have removed at least one token from ${}^\bullet t$. That transition t_j must have been current in state s_0 (or else it could not have fired before t_i) and shared an input place with stubborn transition t. Thus, by the definition of stubborn sets, $t_j \in A$, a contradiction. $\square$

The next lemma is the heart of the stubborn set theory for STP nets. It says that changing a firing schedule to fire a stubborn transition first changes neither the end state nor the total time of the schedule.

Lemma 2 *Fix an STP net $N = (P, T, F, M_0, D)$. Let A be a stubborn set of N in state s_0 and*

$$\sigma = s_0 - t_1 \rightarrow s_1 - t_2 \rightarrow \cdots - t_{n-2} \rightarrow s_{n-2} - t_{n-1} \rightarrow s_{n-1} - t_n \rightarrow s_n$$

be a schedule of N such that $t_n \in A$ but $t_i \notin A$ for $i < n$. Then transition t_n is fireable in state s_0 and the schedule

$$\sigma' = s_0 - t_n \rightarrow s'_1 - t_1 \rightarrow s'_2 - t_2 \rightarrow s'_3 \rightarrow \cdots - t_{n-3} \rightarrow s'_{n-2} - t_{n-2} \rightarrow s'_{n-1} - t_{n-1} \rightarrow s'_n$$

is legal, and has the same ending state (i.e. $s'_n = s_n$) and total time as σ.

Proof First we show that transition $t_n \in \sigma$, and, in general, the first stubborn transition of any schedule, must be fireable in the initial state s_0 of σ. Since t_n is semi-stubborn in s_0, it must be current, and therefore all the other transitions of σ must also be current.

Assume t_n is disabled. By the definition of semi-stubborn sets, there is some empty place $p \in {}^\bullet t_n$ such that all current transitions that could fill p are in A. Some transition t_i of σ with $i < n$ must in fact place a token in p before t_n can fire. But $t_i \in A$, a contradiction. Therefore t_n must be enabled, and hence fireable.

Since t_n is fireable, and all transitions in σ are current but not in A, we have ${}^\bullet t_j \cap {}^\bullet t_n = \emptyset$ for $1 \leq j < n$, and in state s_0 it must be that $d(t_1) = d(t_n)$. Therefore all t_j for $1 \leq j \leq n$ can still be fired in sequence even if t_n is fired first. Moreover, schedule σ' will indeed lead to the same net marking as schedule σ Finally, both schedules take $d(t_1)$ time. Consequently, $s_n = s'_n$. $\square$

The following lemma tells us that two different schedules with the same starting state and sequence of visible transitions with respect to a class of markings $\mathcal{C}$ are in some sense equivalent with respect to $\mathcal{C}$.

Lemma 3 *Fix an STP net N, a class of markings $\mathcal{C}$, and a state s_0 of N such that* $\mathrm{mark}(s_0) \notin \mathcal{C}$. *Let*

$$\sigma = s_0 - t_1 \rightarrow s_1 - t_2 \rightarrow \cdots - t_{n-2} \rightarrow s_{n-2} - t_{n-1} \rightarrow s_{n-1} - t_n \rightarrow s_n$$

be any schedule of N such that $\mathrm{mark}(s_n) \in \mathcal{C}$ *but* $\mathrm{mark}(s_i) \notin \mathcal{C}$ *for $i < n$. Then*

1. $t_n \in \mathrm{vis}(\mathcal{C})$, *and*

2. *for any other schedule ω of N that starts in state s_0 such that* $\mathrm{vis}(\omega) = \mathrm{vis}(\sigma)$, *exactly those states of ω that come after the (last) occurrence*

of t_n have a marking in $\mathcal{C}$.

Proof The fact that $t_n \in \mathrm{vis}(\mathcal{C})$ follows immediately from the definition of visible.

Now consider any other schedule ω starting in state s_0 with $\mathrm{vis}(\omega) = \mathrm{vis}(\sigma)$. We can prove by induction that in the states reached at the end of firing each schedule through the kth visible transition, for any $k \leq n$, all the places in the specification of $\mathcal{C}$ have the same marking. Condition 2 follows easily from this. □

Our main theorem tells us that it is sufficient to check only the reduced state space to verify timing properties.

Theorem 1 *Fix an STP net N with stubborn set generator TS and an arbitrary class of markings $\mathcal{C}$ of N. Let*

$$\sigma = s_0 - t_1 \rightarrow s_1 - t_2 \rightarrow \cdots - t_{n-2} \rightarrow s_{n-2} - t_{n-1} \rightarrow s_{n-1} - t_n \rightarrow s_n$$

be any schedule of N such that $\mathrm{mark}(s_n) \in \mathcal{C}$ *but* $\mathrm{mark}(s_i) \notin \mathcal{C}$ *for* $0 \leq i < n$. *Then there is a schedule*

$$\sigma' = s_0 - t'_1 \rightarrow s'_1 - t'_2 \rightarrow \cdots - t'_{m-2} \rightarrow s'_{m-2} - t'_{m-1} \rightarrow s'_{m-1} - t'_m \rightarrow s'_m$$

of the same total time as σ such that transition $t'_i \in TS(s'_{i-1})$ for $1 \leq i \leq m$, and $\mathrm{mark}(s'_m) \in \mathcal{C}$ *but* $\mathrm{mark}(s'_i) \notin \mathcal{C}$ *for $0 \leq i < m$ whenever all the following conditions are met.*

1. *The reduced state space of N is finite.*

2. *The reduced state space of N contains no cycles of total time zero.*

3. *For each state s of N either*

   (a) *there are no fireable transitions in* $\mathrm{vis}(\mathcal{C}) \cap TS(s)$, *or*

   (b) *all current transitions in* $\mathrm{vis}(\mathcal{C})$ *are in* $TS(s)$.

Proof We proceed by induction on the number of transitions in the schedule σ. Consider first the base case where $\sigma = s_0 - t \rightarrow s_1$. If $t \in TS(s_0)$, then we are done.

Otherwise, transition $t \notin TS(s_0)$. Obviously t is visible, so since $t \notin TS(s_0)$, it must be that every fireable transition in $TS(s_0)$ is invisible. Let $t' \in TS(s_0)$ be fireable, and let $s_0 - t' \rightarrow s'$. By the definition of semi-stubborn sets, transition t must still be fireable in state s'. Since $d(t) = d(t')$, the schedule consisting of transition t' followed by transition t takes the same total time as σ. Since t' is invisible, this schedule of t' followed by t also takes us from s_0 to a state whose marking is in $\mathcal{C}$. However, we do not know whether $t \in TS(s')$. Since t is visible, if $t \notin TS(s')$, it must still be true that every fireable transition in $TS(s')$ is invisible and leaves t fireable after it fires. Since the reduced state space of the net is finite, and contains no cycles of time 0, if we start in state s_0 and fire a sequence of (invisible) transitions consuming zero total time not including t, we must eventually come to a state where only transition t and transitions that share input places with t are fireable. In that state t must be stubborn. Thus we can find a schedule consisting exclusively of stubborn transitions that has the same total time as σ and leads from s_0 to some state with a marking in $\mathcal{C}$.

Assume now that the theorem is true for any schedule with fewer than n transitions and consider a schedule σ containing exactly n transitions for any $n > 1$. There are two cases, depending on whether any transition in σ

is in $TS(s_0)$.

We begin with the case where some transition of σ is in $TS(s_0)$. Let t_i be the first transition of σ that is in $TS(s_0)$. If $i = 1$, then by the inductive hypothesis we are done.

Otherwise, by Lemma 2, transition t_i, the first stubborn transition, is fireable in state s_0. Consider now the case $i = n$. Transition t_n is visible, and if all other transitions in σ are not stubborn, then it must be by Condition 3 of the theorem that all other transitions in σ are invisible. The schedule consisting of t_n alone is legal and has the same total time as σ, and by Lemma 3 it must take us to a state in $\mathcal{C}$.

Now if $1 < i < n$, put

$$\begin{aligned}\sigma_i &= s_0 - t_1 \rightarrow s_1 - t_2 \rightarrow \cdots - s_{i-1} - t_i \rightarrow s_i \\ \sigma_i' &= s_0 - t_i \rightarrow s_1' - t_1 \rightarrow \cdots - s_{i-1}' - t_{i-1} \rightarrow s_i\end{aligned}$$

By Lemma 2, state s_i is indeed the last state of σ_i', and both σ_i and σ_i' take the same total time. If we knew that $\text{mark}(s_1'), \text{mark}(s_2'), \ldots, \text{mark}(s_{i-1}')$, were all not in $\mathcal{C}$, then by the inductive hypothesis the schedule σ_i' followed by transitions $t_{i+1}, t_{i+2}, \ldots, t_n$ would complete the induction step.

By Lemma 3, it is sufficient to show that $\text{vis}(\sigma_i') = \text{vis}(\sigma_i)$. If t_i is invisible, then obviously $\text{vis}(\sigma_i') = \text{vis}(\sigma_i)$, since σ_i and σ_i' differ only in the position of t_i. Transition t_i is fireable, so all transitions t_j with $j < i$ must be current. If t_i is visible, then all t_j with $j < i$ are invisible, because of Condition 3 of the theorem. In this case, $\text{vis}(\sigma_i') = \text{vis}(\sigma_i) = t_i$. This concludes the case where there is some transition that is stubborn in state s_0.

Now consider the case where $TS(s_0) \cap \sigma = \emptyset$. Let $t' \in TS(s_0)$ be fireable, and let $s_0 - t' \rightarrow s'_1$. By Lemma 1, we know that schedule σ consumes zero time after its first transition, so all transitions of σ are current. Therefore, transition t' must be invisible, because otherwise by Condition 3 of the theorem, transition t_n, which is certainly visible, would have to be stubborn. By the definition of semi-stubborn sets, t' must not share input places with any transition of σ. If $t_1 \in TS(s')$, then we are done. Otherwise, as with the base case, we know that additional stubborn transitions cannot consume any more time, and cannot share input places with any transition of σ. Again, we must eventually get to a state with t_1 stubborn. $\square$

Remark: We must have something like Condition 2 outlawing cycles of states taking zero total time. Otherwise our algorithm might fail by having as its stubborn set for every state on such a cycle only the fireable (zero time) transition that leads around that cycle. We impose Condition 2 because we feel that a zero time cycle of states likely reflects an error in the STP net (or the system it is modeling). It is easy for an algorithm performing reduced state space exploration to test for such cycles and report their presence if any are found.

There are at least two other ways one could avoid this problem. One is to simply strengthen Condition 3 of the theorem to require that for every state s, all current visible transitions are in $TS(s)$. Another, in the spirit of Valmari [15], is to require that in every such cycle of states there is one state s such that $TS(s)$ contains all current visible transitions.

Notice that Theorem 1 also allows to verify safety properties of the form "from state s we never encounter a state with a marking in $\mathcal{C}$," and liveness properties of the form "from state s we must eventually reach a state with markings in $\mathcal{C}$."

Corollary 2 *Fix an STP net N with stubborn set generator TS, an arbitrary state s_0 of N and a class of markings $\mathcal{C}$. Then, under the same set of conditions as specified for Theorem 1, it suffices to explore only the reduced state space of N to determine whether starting from state s_0 one ever encounters a state with marking in $\mathcal{C}$, and also whether starting from state s_0 one must eventually reach a state with marking in $\mathcal{C}$.*

Proof If there is a state in the full state space with marking in $\mathcal{C}$ reachable from state s_0, then there must also be such a state in the reduced state space. Otherwise the shortest time path would go from some fixed time in the full state space to infinity in the reduced state space, contradicting Theorem 1. For the liveness question, "Are we we forced to eventually reach a state in $\mathcal{C}$?", the reduced state space must give the same answer as the full state space, or the longest time path would be different in the reduced state space. □

4 Empirical results

To help us assess the effectiveness of our extension of stubborn sets, we have considered several examples of real-time systems available in the literature and we have run experiments with our PARTS toolset. The examples presented here are not intended to provide a definitive conclusion about the usefulness of real-time stubborn sets. Rather, these examples highlight some of the strengths and limitations of this method.

4.1 Methodology

We used the following algorithm for constructing stubborn sets. First we look for one invisible fireable transition that is not in conflict with any other transition. If we find one, then it forms a singleton stubborn set. Next we check to see whether there were exactly two fireable transitions in the current state. If not, then for each fireable transition, we built a stubborn set including that transition, and we use the stubborn set with the smallest number of fireable transitions (stopping early if we ever construct a stubborn set with exactly two fireable transitions).

All our experiments were run on a Sun SPARC Station 10 with 32 Mbytes of memory.

4.2 Computer simulations

Here we report the experimental results we obtained for a simple *divide-and-conquer* example [5], for a real-time version of Leveson's *railroad crossing* example [7, 8], and for an *Ethernet protocol* [11].

4.2.1 Divide and conquer example

In the divide-and-conquer example [5], n computing elements are given. When a problem must be computed, the first element nondeterministically chooses whether to conquer the problem (which requires a "long" time), or to pass it along to the next computing element (which requires a "short" time on the part of the first element). The next $(n-2)$ elements have a behavior similar to the first element; however, the last element is required to solve the problem. An STP net modeling this problem is shown in Figure 1.

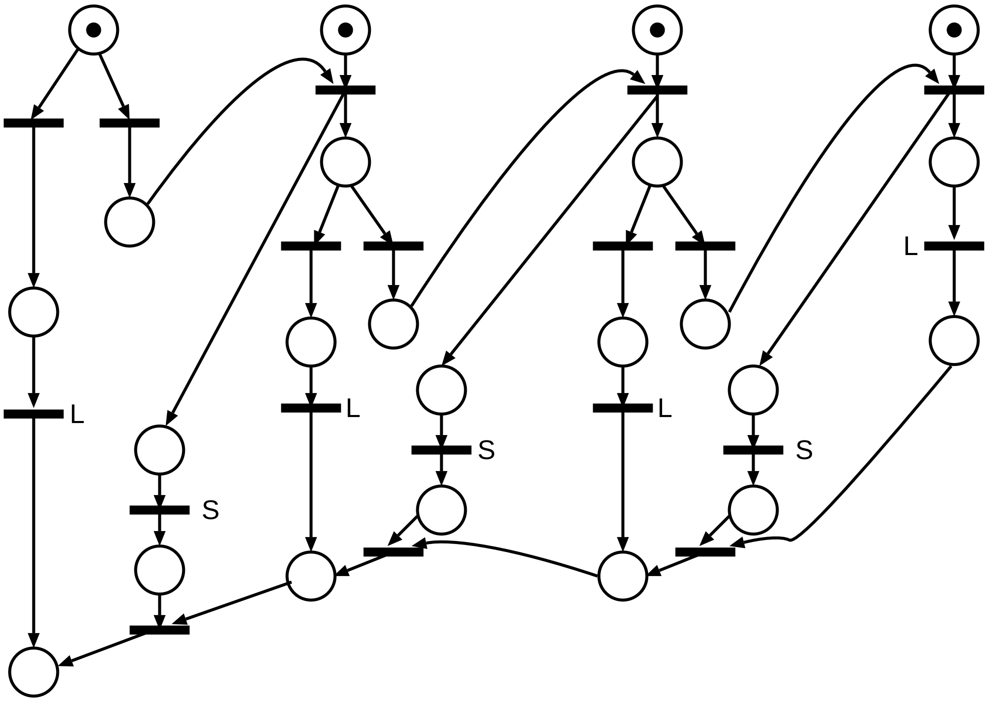

Figure 1: STP net modeling divide-and-conquer example for $n = 4$ computing units. The delay of each transition is either "long" (labeled L), "short" (labeled S), or zero (unlabeled).

We have analyzed this example for different values of n. In Table 1 we report experimental results both for full and reduced state space generation.

Note that the growth in state space size is exponential when stubborn sets are not used; however, this growth becomes quadratic when stubborn sets are used. To be precise, with stubborn sets we get $(n^2 + 3n - 2)$ states.

4.2.2 GRC and ethernet examples

The generalized railroad crossing (GRC) system operates a gate at a railroad crossing containing n. tracks [7]. The area of the crossing between the railroad tracks and the road for cars is a known distance beyond a

	Full State Space			Reduced State Space		
Problem	States	Arcs	Sec	States	Arcs	Sec
D&C 10	1,096	4,179	8.5	128	127	0.4
D&C 20	—	—	—	458	457	2.1
D&C 40	—	—	—	1,718	1,717	13.9
D&C 60	—	—	—	3,778	3,777	44.0
D&C 80	—	—	—	6,638	6,637	107.2
D&C 100	—	—	—	10,298	10,297	221.7

Table 1: State space reduction for the divide-and-conquer example with stubborn sets. These simulations were run using 5 as the "long" time and 1 as the "short" time. A — indicates that the experiment could not be completed within 60 minutes of CPU time. Sec indicates seconds of CPU time on a Sun SPARC station 10.

	Full State Space			Reduced State Space		
Problem	States	Arcs	Sec	States	Arcs	Sec
GRC 1	207	417	0.7	83	94	0.4
GRC 1 crash	159	204	0.4	140	156	0.4
GRC 2	5,058	11,836	20.5	2,008	2,320	8.7
GRC 2 crash	5,911	8,767	16.8	4,585	5,376	13.9
Ethernet	642	1,406	2.2	329	481	1.3

Table 2: Experimental results showing state space reduction for stubborn sets. Sec indicates seconds of CPU time on a Sun SPARC station 10.

warning sensor. This sensor notifies the GRC system whenever a train passes by. Sensors are also used to signal the GRC system when a train leaves the crossing. Inputs for the problem include the shortest possible time for any train to travel from the warning sensor to the crossing, and upper and lower bounds on the time it takes for the gate barring the crossing to open, and to close. We have verified a safety property and a utility property for this example. The safety property stipulates that the gate is always closed when a train is in the crossing. The utility property states that the gate should be open as much as possible, given the safety property.

In Table 2 we report data for a correct and an incorrect version of the GRC example. In each case, we automatically analyzed one-track and two-track versions. In the correct versions the two properties are satisfied. In the incorrect version, the erroneous GRC system would not start lowering the gate while it was in the process of opening. Thus the gate could only start closing after becoming completely open. As a result, the safety property is not satisfied, because a train that arrives at the warning sensor while the gate is opening (right after a previous train has left the crossing) can get to the crossing without the gate being completely closed. This fact is correctly reported by our toolset. The use of stubborn sets causes a substantial reduction in state space size; however, the growth in the number of states remains exponential in the number of tracks even after the stubborn set method is applied.

For the GRC2 example we also generated a reduced state space using the more limited stubborn set method that follows from Valmari's definitions for for arbitrary variable transition systems. In this case we obtained 4,057 states and 5,221 arcs in 12.0 sec of CPU time. We conclude that our enhancements to the stubborn set method have improved the effectiveness of this method with respect to Valmari's original definition.

The last example we considered is the well-known CSMA/CD (carrier sense, multiple access with collision detection) protocol for an Ethernet bus [11]. This example has n stations sending packets through a bus. The bus does not perform buffering of packets, and there is a propagation delay σ on the sending of packets. Suppose that the bus is available for some time before a sender S_1 sends a packet at time θ_0. For the next σ time units, the other senders sense that the bus is still available. At time $(\theta_0 + \sigma)$ the bus signals the other senders that the bus is busy. If a sender S_2 decides to send a packet but senses that the bus is busy, then S_2 waits for a random time between 0 and 2σ before attempting to send again. If, however, S_2 senses that the bus is still available and sends a packet after S_1 has sent a packet, a collision occurs. When this happens, the bus waits for σ units and then signals all senders that there was a collision and that the packets were lost. In this case, each sender randomly waits for up to 2σ time units before retrying. Suppose that no collision occurs after S_1 sends a packet. In this case, the packet is delivered λ time units after the packet was sent, with $\lambda \gg \sigma$. Moreover, the bus waits for σ time units after the completion of the send before signaling the other senders that the bus is free.

We analyzed a version of the bus protocol example with two senders and delays $\lambda = 10$ and $\sigma = 1$. Our experimental results are reported in the last row of Table 2. As the data indicate, state space construction for this example is relatively fast even without the stubborn set method; however, this method substantially reduces both the size of the state space and the time required for state space construction.

4.3 Evaluation of empirical results

First of all, we observe that the real-time stubborn set method usually increases the efficiency of an analysis. In general, the size of the state space and the time for state space generation are substantially lower when stubborn sets are used.

Second, the effectiveness of the method varies depending on the system being analyzed. In some cases, such as the divide-and-conquer example, the method can dramatically reduce state space size from exponential to polynomial in the number of processes in the system. This result mirrors similar results obtained by Valmari in the (untimed) analysis of certain concurrent systems [15]. In other cases, such as the Ethernet protocol and the railroad crossing examples, the method can achieve good reductions; however, state space size remains exponential in the number of processes. We conclude that stubborn sets are a useful reduction technique; however, they should be combined with additional state space reduction methods in order to make reachability-based analysis practical for real-time systems. We are currently investigating several additional state space reduction methods [4, 13].

Third, our stubborn set method is more effective than the method one would obtain by treating STP nets as variable transition systems and using Valmari's definitions for variable transition systems. Only for the divide-and-conquer example would one get equally good results. For both the Ethernet protocol and all versions of the GRC example, we obtained much smaller reductions when we tried that stubborn set method instead of our stubborn set method.

5 STP net's notion of time

STP nets assign each atomic action a fixed time. This leads to some minor speedup with respect to untimed concurrency analysis, because certain actions are sequenced by timing considerations. These minor benefits are reflected in the *full* state space section of Tables 1 and 2 of experimental results in Section 4; note that all those state spaces are very large. In fact, whenever we have used STP nets to model a real-time version of a system whose untimed concurrent version has an exponential state space, our STP net model has had an exponential state space.

When we model atomic actions that intuitively have variable duration, the size of our STP nets and their state spaces increase because we must model this variable duration with many transitions. For example, when modeling the Ethernet protocol where a sender waits from 0 to 2σ time units to resend, we have in our STP net $2\sigma + 1$ transitions with delays $0, 1, \ldots, 2\sigma$.

The simplicity of STP nets does give a significant advantage in the speed of unreduced state space generation. If instead of fixed durations we allowed intervals for each action, we would have time Petri nets [9]. Time Petri nets (TPNs) are identical to STP nets except that they allow for firing *intervals* instead of firing delays. That is, each transition t is associated with an interval $[a, b]$ with $0 \leq a \leq b \leq +\infty$ such that once t becomes enabled it must fire from a to b time units later, unless it becomes disabled in the meantime.

The states of a TPN are more complicated than those of an STP net; one definition of the state space for TPNs is given by Berthomieu and Diaz [1]; subsequently, Yoneda et al. [16] provided a new definition that includes more information useful for verification in each state. Among other information, every state includes a dynamic firing *interval* for each transition.

Calculating the new state obtained by firing an enabled transition from a given state—and translating this new state to some canonical form so one can tell whether the new state has been seen previously—is a basic operation performed many, many times in any state-space exploration method. For

STP nets, this operation is extremely fast, linear in all relevant parameters.

The key parameter here is ℓ, the number of simultaneously enabled transitions.[3] Typically ℓ might be roughly the number of processes times the number of nondeterministic choices that any process can make. For time Petri nets, this basic operation is much slower. Berthomieu and Diaz leave it as an open problem whether finding the next state is even *polynomial* in ℓ [1]; we have developed an algorithm to perform this operation in time $O(\ell^3)$ for their model. Yoneda et al. use an algorithm with a $\Omega(\ell^3)$ lower bound for this operation [16].

In short, time Petri nets are preferable to STP nets because they have greater modeling capacity. However, the price for this modeling capacity is that the generation of a single state becomes cubic in the number of processes.

5.1 Comparison to model of Yoneda

Our stubborn set techniques were developed independently of but several months subsequent to a partial-order technique for time Petri nets developed by Yoneda et al. [16]. In this subsection we compare our efforts to theirs.

As discussed in the previous subsection, we believe that for systems with many processes their time Petri net methods will not be practical, because the generate-next-state operation, which is at the center of the innermost loop of any state-space-based analysis, is at least cubic in their system, whereas it is linear for our method.

For instance, for Corbett and Avrunin's divide-and-conquer example dis-

[3] "Enabled" still means having a token in every input place; there are usually more enabled than fireable transitions.

cussed in Section 4, both partial-order methods result in a $\Theta(n^2)$ reduced state space. However, for our method, no state requires more than linear time to generate, whereas for Yoneda's method, there will be $\Theta(n^2)$ states that require $\Omega(n^3)$ time to generate. Thus, our overall time to analyze this problem is $\Theta(n^3)$, and we can comfortably analyze the $n = 100$ case on a workstation. It seems unlikely that Yoneda's method, with an overall time of $\Omega(n^5)$ for the divide-and-conquer example, can analyze the $n = 100$ case.

We conclude that Yoneda's method is probably preferable for analyzing systems with a small-to-medium number of processes because of its richer notion of time, and our method is probably preferable for analyzing systems with a large number of processes.

6 Concluding remarks

The stubborn set method seeks to optimize reachability-based verification of finite state systems by reducing the number of states that must be explicitly enumerated. Valmari showed that the method can be profitably applied to the analysis of safety and liveness properties of untimed concurrent systems [15]. We have defined an extended version of this method for STP nets, our model of real-time systems, and shown that it can be usefully applied to real-time analysis. Our extended version of stubborn sets takes advantage of the timing properties of an STP net to obtain additional state space reductions with respect to Valmari's method. The resulting method has given encouraging experimental results on several examples of real-time systems. Finally, we have compared our approach to real-time stubborn sets to the related approach of Yoneda et al. [16]. Our future work will be aimed at performing experiments with additional examples of real-time systems, and at integrating stubborn sets with other state space optimization techniques, such as symmetry-based methods and compositional methods.

References

[1] 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.

[2] 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.

[3] U. Buy and R. H. Sloan. Automatic real-time analysis of Ada tasking programs for embedded systems. Submitted for publication to *Real Time Systems*.

[4] 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.

[5] J. C. Corbett and G. S. Avrunin. A practical technique for bounding the time between events in concurrent real-time systems. In *Proc. 1993 Internat. Sympos. on Software Testing and Analysis*, pages 110–116. ACM, June 1993.

[6] S. Duri, U. Buy, R. Devarapalli, and S. M. Shatz. Using state space reduction methods for deadlock analysis in Ada tasking. In *Proc. 1993 Internat. Sympos. on Software Testing and Analysis*, pages 51–60. ACM, June 1993.

[7] C. Heitmeyer, R. Jeffords, and B. Labaw. A benchmark for comparing different apporaches for specifying and verifying real-time systems. In *Proc. Tenth Internat. Workshop on Real-Time Operating Systems and Software*, May 1993.

[8] F. Jahanian and D. A. Stuart. A method for verifying properties of Modechart specifications. In *Proc. Real-Time Systems Sympos.*, pages 12–21, Dec. 1988.

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

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

[11] X. Nicollin, J. Sifakis, and S. Yovine. Compiling real-time specifications into extended automata. *IEEE Trans. Softw. Eng.*, 18(9):794–804, Sept. 1992.

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

[13] R. H. Sloan and U. Buy. Reduction rules for time Petri nets, 1994. Submitted for publication to *Acta Informatica.*

[14] A. Valmari. Heuristics for lazy state generation speeds up analysis of concurrent systems. In *Proc. Finnish Artificial Intelligence Symposium*, volume 2, page 640 650, Helsinki, Finland, 1988.

[15] 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.

[16] 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.