Efficient Verification of Parallel Real–Time Systems

Tomohiro Yoneda[1], Atsufumi Shibayama[1],
Bernd–Holger Schlingloff[2], Edmund M. Clarke[3]

[1] Tokyo Institute of Technology ({yoneda,sibayama}@cs.titech.ac.jp)
[2] Technische Universität München (schlingl@informatik.tu-muenchen.de)
[3] Carnegie Mellon University (emc@cs.cmu.edu)

Abstract. This paper presents an efficient model checking algorithm for one–safe time Petri nets and a timed temporal logic. The approach is based on the idea of (1) using only differences of timing variables to be able to construct a finite representation of the set of all reachable states and (2) further reducing the size of this representation by exploiting the concurrency in the net, i.e. only one of several equivalent interleavings being generated for the evaluation of the given formula. This reduction of the state space is possible, because the considered linear–time temporal logic is stuttering invariant. In this paper the concrete model checking algorithm is developed and some experimental results which demonstrate the efficiency of the method are given.

1 Introduction

Model checking has proved to be useful for the automatic verification of finite state systems; see, e.g. [CES86] and others. Unfortunately, the verification of large parallel systems suffers from the so called *state explosion problem*: the number of states to be checked is exponential in the size of the system. An approach to confine this problem is to use partial orders and thus to avoid the construction of equivalent states reachable by different interleaving of atomic events. Several methods [Val90, God90] based on this approach have been proposed for reachability analysis and various other properties of Petri nets.

Those *untimed* verification techniques are suitable to check qualitative timing properties. Recently, the demand for correctness proofs of real–time systems increases rapidly. In real–time systems, the system correctness depends not only on the functional results of the system but also on the time at which these results are produced.

Such systems are often represented by finite automata, whose transitions are labeled by time intervals [AH89, and others], or which have a finite number of clocks [ACD90]. However, concurrency can not be modeled directly by such timed state graphs. On the other hand, *time Petri nets* were considered in [MF76]. Time Petri nets are an adequate model of timed concurrent systems, which generalizes other models in a natural way. Using time Petri nets, it is very easy to model, for example, logic gates with bounded delays or network protocols.

In order to specify and verify real–time systems, languages for reasoning about quantitative timing properties are necessary. Many timed temporal logics have been proposed to express such properties [AH89, ACD90, and others]. But again, for practical applications, state explosion is a big problem. There are only a few reports on the avoidance of state explosion in the case of real–time systems. Reachability analysis techniques for time Petri nets using partial orders have been reported in [YTK91].

In this paper, we develop an efficient model checking algorithm for the verification of real–time systems based on the partial order approach. The given real–time system is modeled by a time Petri net. For the specification of properties and time constraints of the time Petri nets we use a suitably extended linear temporal logic. The language is designed such that it fits to the partial order analysis. Automatic verification is achieved by generating a reduced state space of the net, which is big enough to evaluate the given formula, and by traversing the reduced state space with the given formula.

The rest of this paper is organized as follows. In the next section, several definitions concerning time Petri nets are given. In Sect. 3, we introduce our logic. Both the basic model checking algorithm and its partial order improvement are developed in the following two sections. In Sect. 6, some experimental results are presented which demonstrate the efficiency of the proposed method. Finally, we summarize our discussion.

2 Time Petri Nets

Time Petri nets were first defined in [MF76], and used for timing verification in [BD91, RB86].

Let $\mathbf{Q}$ be the set of rational numbers, and $\mathbf{Q}^+$ the set of nonnegative rational numbers. A *time Petri net* N is six–tuple, $N = (P, T, F, \mathit{Eft}, \mathit{Lft}, \mu_0)$, where

- $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of *places*;
- $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of *transitions* ($P \cap T = \emptyset$);
- $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*;
- Eft, Lft : $T \to \mathbf{Q}^+$ are functions for the *earliest* and *latest firing times* of transitions, satisfying $\mathit{Eft}(t) \leq \mathit{Lft}(t)$ for all $t \in T$;
- $\mu_0 \subseteq P$ is the *initial marking* of the net.

For any transition t, $\bullet t = \{p \in P \mid (p,t) \in F\}$ and $t\bullet = \{p \in P \mid (t,p) \in F\}$ denote the *preset* and the *postset* of t, respectively. To simplify the presentation, we require that $\bullet t \cap t\bullet = \emptyset$ for every transition t; however, this requirement is not essential for our results.

A *marking* μ of N is any subset of P. A transition is *enabled* in a marking μ if $\bullet t \subseteq \mu$ (all its input places have tokens in μ); otherwise, it is *disabled*. Let $\mathit{enabled}(\mu)$ be the set of transitions enabled in μ.

A *state* σ of a time Petri net is a pair (μ, clock), where μ is a marking and *clock* is a function $T \to \mathbf{Q}^+$. The *initial state* σ_0 is $(\mu_0, \mathit{clock}_0)$, where $\mathit{clock}_0(t) = 0$ for all $t \in T$.

The states of time Petri nets change, if time passes or if a transition fires. In state $\sigma = (\mu, \mathit{clock})$, time $\tau \in \mathbf{Q}^+$ *can pass*, if for all $t \in \mathit{enabled}(\mu)$, $\mathit{clock}(t) + \tau \leq \mathit{Lft}(t)$. In this case, state $\sigma' = (\mu', \mathit{clock}')$ is *obtained by passing* τ from σ, if

1. $\mu = \mu'$, and
2. for all $t \in T$, $clock'(t) = clock(t) + \tau$.

In state $\sigma = (\mu, clock)$, transition $t \in T$ *can fire*, if $t \in enabled(\mu)$, and $clock(t) \geq Eft(t)$. In this case, state $\sigma' = (\mu', clock')$ is *obtained by firing* t from σ, if

1. $\mu' = (\mu - \bullet t) \cup t\bullet$, and
2. for all $\hat{t} \in T$, $clock'(\hat{t}) = \begin{cases} 0 & \text{if } \hat{t} \in enabled(\mu'), \hat{t} \notin enabled(\mu) \\ clock(\hat{t}) & \text{else} \end{cases}$.

Intuitively, this can be interpreted as follows : Firing a transition t consumes no time, but updates μ and *clock* such that the clocks associated with newly enabled transitions (i.e. transitions which are enabled in μ' but not in μ) are reset to 0. Clock values of other transitions (i.e. transitions not affected by t) are left unchanged.

A *run* $\rho = (\sigma_0, \sigma_1, \sigma_2, \ldots)$ of N is a finite or infinite sequence of states such that σ_0 is the initial state, and σ_{i+1} is obtained from σ_i by passing time τ and then firing transition t. We write $\sigma_i(\rho)$ for the i–th state of ρ, and similarly $\mu_i(\rho)$ and $clock_i(\rho)$, and omit the argument (ρ) whenever appropriate. A run is *maximal*, if it is infinite or in its last state there is no enabled transition. The *behavior* $B(N)$ of N is the set of all maximal runs of N.

Given any run ρ and $i \geq 0$, we define $time_i(\rho)$ to be the sum of all times τ passed between $\sigma_0(\rho)$ and $\sigma_i(\rho)$; that is, $time_0(\rho) = 0$ and $time_{i+1}(\rho) = time_i(\rho) + clock_{i+1}(t) - clock_i(t)$ for some t which is not newly enabled in μ_{i+1}.

A state σ is *reachable* if there exists a finite run whose last state is σ. A time Petri net is *one–safe*, if for every state $\sigma = (\mu, clock)$ obtained by passing time from any reachable state σ', and for every transition t which can fire in σ, $t\bullet \cap \mu = \emptyset$. The restriction to one–safe nets simplifies both the analysis of time Petri nets and the reduced state space generation.

Further, for the proof of the finiteness of the graphs introduced in Sect. 4, we need the following *progress condition* [AH89]: The sum of earliest firing times of transitions forming any loop in N is positive. More precisely, for every set $\{t_1, t_2, \ldots, t_n\}$ of transitions such that $t_1 \bullet \cap \bullet t_2 \neq \emptyset$, $t_2 \bullet \cap \bullet t_3 \neq \emptyset$, ..., $t_n \bullet \cap \bullet t_1 \neq \emptyset$ it holds that $Eft(t_1) + Eft(t_2) + \cdots + Eft(t_n) > 0$. This guarantees that in any infinite run time is increasing beyond any bound.

In the sequel, a *net* will always be a one–safe time Petri net satisfying the progress condition.

Fig. 1 shows an example net N_x. Pairs of numbers after transition names represent earliest and latest firing times, respectively. Since, for example, t_2 can fire at any time between 1 and 3 after being enabled, the behavior $B(N_x)$ contains an infinite number of runs. Furthermore, since $Eft(t_0) > Lft(t_1)$, t_0 can never fire, and thus every run of N_x is infinite.

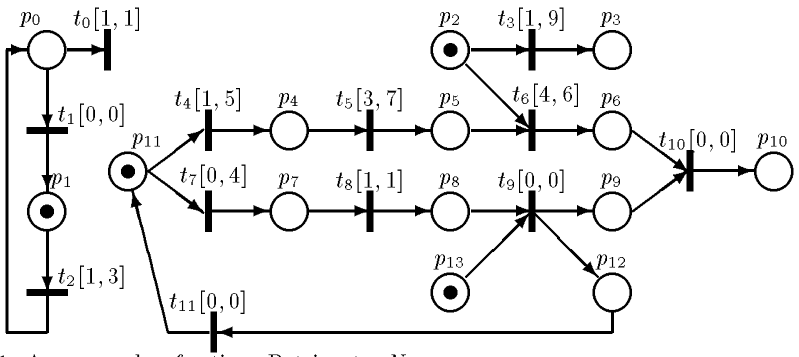


Fig. 1. An example of a time Petri net : N_x

3 TNL, a Timed Temporal Logic for Nets

In this section, we propose a temporal logic for the specification of net properties. On one hand, every such logic should be expressive enough to be capable of formalizing "interesting" properties including quantitative time requirements, and on the other hand there should exist an efficient model checking algorithm for the logic avoiding the state explosion problem. Since with branching time logics such as CTL[CES86] it is by now not possible to use the parallelism in the net to reduce the average time complexity of the model checking problem, we focus on linear time temporal logic.

Given a net N and formula φ, we want to find whether there exists a run ρ of N satisfying φ (written $\rho \models \varphi$). In general there are infinitely many runs of N, therefore we group these into a finite number of equivalence classes $[\rho_1]$, $[\rho_2]$, $\ldots$, $[\rho_c]$, such that the existence of a satisfying run ρ implies that every element of the equivalence class $[\rho]$ satisfies φ. Thus we only have to check a finite number of equivalence classes, and a coarser partition yields a better algorithm.

Consider a set of atomic propositions $\{p_1, \ldots, p_k\}$ of a logic, such that the notion of *validity* ($(\rho, i) \models p_j$) of an atomic proposition p_j in a state σ_i of a run ρ is defined. Two runs ρ and ρ' are *strongly equivalent* with respect to $\{p_1, \ldots, p_k\}$, if $(\rho, i) \models p_j$ iff $(\rho', i) \models p_j$ for all $i \geq 0$ and all atomic propositions $p_j \in \{p_1, \ldots, p_k\}$.

A state σ_{i+1} in a run ρ is *stuttering* with respect to $\{p_1, \ldots, p_k\}$, if $(\rho, i) \models p_j$ iff $(\rho, i+1) \models p_j$ for all $p_j \in \{p_1, \ldots, p_k\}$. Two runs ρ and ρ' are *stuttering equivalent* w.r.t. $\{p_1, \ldots, p_k\}$, if the two sequences obtained by eliminating all stuttering states from ρ and ρ' are strongly equivalent w.r.t. $\{p_1, \ldots, p_k\}$. Define a formula φ to be *stuttering invariant*, if for any two runs ρ and ρ' which are stuttering equivalent with respect to the atomic propositions in φ it holds that $\rho \models \varphi$ iff $\rho' \models \varphi$.

Stuttering invariance allows to group all stuttering equivalent runs into the same equivalence class, thereby reducing the average complexity of the model checking. In particular, all runs which differ only in the interleaving of independent transitions are stuttering equivalent with respect to places not connected to these transitions.

Unfortunately, most formulas of existing real–time logics are not stuttering invariant. Firstly, uncautious use of a "next–state" operator inhibits stuttering invariance. Moreover, if the logic allows to directly refer to the time associated with a state in a run, then a similar effect as with a "next–state" operator can result. In other words, classical real–time logics are inappropriate for our purpose. Therefore, our logic only refers to *differences* of firing times of transitions.

Our logic, which we call **TNL**, is formally defined as follows. Given any net $N = (P, T, F, \mathit{Eft}, \mathit{Lft}, \mu_0)$, let $\mathcal{P} = \{p^\bullet \mid p \in P\} \cup \{p^\circ \mid p \in P\}$ be the set of *time variables*. The set of *propositional variables* is P. The *formulas* of **TNL** are defined inductively:

- If $x, y \in \mathcal{P}$ and $c \in \mathbf{Q}$, then $x - y \le c$ is a formula.
- Every propositional variable is a formula.
- *false* is a formula.
- If φ_1 and φ_2 are formulas, then $(\varphi_1 \rightarrow \varphi_2)$ and $(\varphi_1 \,\mathcal{U}\, \varphi_2)$ are formulas.

false, propositional variables, and $x - y \le c$ for $x, y \in \mathcal{P}$ and $c \in \mathbf{Q}$ are called *atomic propositions*. Additional boolean connectives *true*, $\neg$, $\wedge$, $\vee$, $\rightarrow$, and temporal connectives $\Box$, $\Diamond$ can be defined as usual. Also formulas $x - y \sim c$, where $\sim$ is any relation from $\{<, =, \ge, >\}$, can be defined in an obvious way.

In order to define the semantics of **TNL**, the value of time variables in a state of a run has to be defined. Intuitively, $p^\bullet$ and $p^\circ \in V$ represent the time when the place p got or lost the latest token, respectively.

Let ρ be a run of N, $i \ge 0$, and let $x \in \mathcal{P}$.

$$eval_i(x) = \begin{cases} 0 & \text{if } i = 0 \\ time_i(\rho) & \text{if } x = p^\bullet,\ p \in \mu_i,\ p \notin \mu_{i-1} \\ time_i(\rho) & \text{if } x = p^\circ,\ p \notin \mu_i,\ p \in \mu_{i-1} \\ eval_{i-1}(x) & \text{otherwise} \end{cases}$$

Validity of a **TNL** formula φ in a run ρ at point $i \ge 0$, denoted by $(\rho, i) \models \varphi$, is now defined by induction on φ as usual:

1. $(\rho, i) \models x - y \le c$ iff $eval_i(x) - eval_i(y) \le c$
2. $(\rho, i) \models p$ iff $p \in \mu_i$ for $p \in P$
3. $(\rho, i) \not\models \mathit{false}$
4. $(\rho, i) \models (\varphi_1 \rightarrow \varphi_2)$ iff $(\rho, i) \models \varphi_1$ implies $(\rho, i) \models \varphi_2$
5. $(\rho, i) \models (\varphi_1 \,\mathcal{U}\, \varphi_2)$ iff there exists $j \ge i$ such that $(\rho, j) \models \varphi_2$, and for all k such that $i \le k < j$, $(\rho, k) \models \varphi_1$

ρ *satisfies* φ, denoted by $\rho \models \varphi$, if $(\rho, 0) \models \varphi$. φ is *satisfiable* in N if there exists a (maximal) run $\rho \in B(N)$ such that $\rho \models \varphi$.

Consider our example net from Fig. 1. Then the formula $\Diamond p_{10}$ is satisfiable if the place p_{10} is reachable, which is the case, and $\Diamond(p_{10} \wedge p_{10}^\bullet - p_{10}^\circ \le 8)$ is satisfiable if it can be reached within 8 time units, which is not the case (Note that $eval_i(p_{10}^\circ) = 0$ for all i). $\Box\Diamond(p_1^\circ - p_1^\bullet > 2)$ means that t_2 may infinitely often need more than 2 time units to fire.

4 Model Checking for Nets and TNL

In general, there exist infinitely many runs of a given net N. In this section, we will construct a finite graph G such that the paths through G represent exactly the runs of N, and that every node in G determines the truth value of all atomic propositions appearing in the given **TNL** formula. Thus, the **TNL** model checking problem is reduced to the LTL model checking problem, for which an algorithm can be found in [LP85].

Basically, we use a set of inequalities to represent a number of different clock functions. By an *inequality* we mean any string of the form "$x - y \sim c$", where x and y are from a designated set of variables, $c \in \mathbf{Q}$ and $\sim$ is a relation symbol from $\{\leq, <, =, >, \geq\}$. If I is a set of inequalities, then $var(I)$ denotes the set of variables that I contains; we say that I is a set of inequalities *over* $var(I)$.

Let I be a set of inequalities over $\{x_1, x_2, \ldots, x_n\}$. A *feasible vector* for I is a tuple $(c_1, c_2, \ldots, c_n)$ of constants $c_i \in \mathbf{Q}$, such that every inequality obtained by replacing every x_i by c_i $(1 \leq i \leq n)$ in any inequality from I holds in the theory of rational numbers. The *solution set* of I is the set of feasible vectors for I. A set of inequalities is *consistent* if its solution set is nonempty. Two sets of inequalities are *isomorphic*, if they have the same solution set.

The *closure* of a **TNL**–formula φ, denoted by $Cl(\varphi)$, is the smallest set of inequalities such that for every inequality "$x - y \leq c$" appearing in φ, both "$x - y \leq c$"$\in Cl(\varphi)$ and "$x - y > c$"$\in Cl(\varphi)$. A *maximal consistent set* of φ is a maximal set $F \subseteq Cl(\varphi)$ of inequalities which is consistent. Given any set I of inequalities, a *complete extension* $\hat{I}$ of I and φ is any consistent set $\hat{I} = I \cup I'$, such that I' is a maximal consistent set of φ. $CE(I, \varphi)$ denotes the set of all complete extensions of I and φ. Note that for consistent I, $CE(I, \varphi)$ is nonempty and finite.

In the previous section, time variables representing times when the corresponding places got or lost its latest token were introduced. In order to grasp the future behavior of the net, we introduce another sort of time variables, called *transition variables*, representing the next firing time of (enabled) transitions. Since there is no confusion, we use the set T to denote transition variables as well as transitions; all inequalities in this section will therefore use variables from $V = \mathcal{P} \cup T$. $\mathcal{P}_\varphi$ denotes the set of time variables appearing in φ.

An *atom* is a pair $\alpha = (\mu, I)$, where μ is a marking and I is a set of inequalities. The *initial* atom is $\alpha_0 = (\mu_0, \widehat{I_0})$, where μ_0 is the initial marking of the net, and $\widehat{I_0}$ is the unique complete extension of the following set I_0 of inequalities:

$$\begin{aligned} I_0 = \{&\text{"}x - y = 0\text{"} \mid x, y \in \mathcal{P}\} \cup \\ \{&\text{"}t - x \geq \mathit{Eft}(t)\text{"} \mid t \in \mathit{enabled}(\mu_0), x \in \mathcal{P}\} \cup \\ \{&\text{"}t - x \leq \mathit{Lft}(t)\text{"} \mid t \in \mathit{enabled}(\mu_0), x \in \mathcal{P}\} \end{aligned}$$

The first line defines the initial values of all time variables to be equal. The second and third line give the timing constraints on the next firing of transitions enabled in the initial marking.

We are now going to describe how the set of successor atoms α' of an atom α can be computed. To this end we need the notion of *deletion* of a set U of variables from a set I of inequalities. For every such I and U there exists an (up to isomorphism) unique set $I' = delete(I, U)$ of inequalities over $var(I) - U$, such that the solution set of I' is equal to the solution set of I, projected on $var(I) - U$. For example, if $I = \{$"$y - x \geq 2$", "$y - x \leq 7$", "$y - z < 3$", "$z - y \leq 11$"$\}$, then $delete(I, \{y\}) = \{$"$x - z < 1$", "$z - x \leq 18$"$\}$. As shown in [JM87], I' can be computed by a graph–based algorithm in time $O(|var(I)|^3)$.

If $\alpha = (\mu, I)$ is an atom, then $\mathit{firable}(\alpha) = \{t_f \mid t_f \in enabled(\mu), I \cup \{$"$t - t_f \geq 0$" $\mid t \in enabled(\mu)\}$ is consistent$\}$ is the set of transitions that can fire earlier than all other transitions in the given marking and timing properties. Let t_f be a transition in $\mathit{firable}(\alpha)$, $\mu' = (\mu - \bullet t_f) \cup t_f\bullet$, and $U_f = \{p^\circ \mid p \in \bullet t_f\} \cup \{p^\bullet \mid p \in t_f\bullet\}$. We define the following sets of inequalities:

- $J_1 = I \cup \{$"$t - t_f \geq 0$" $\mid t \in enabled(\mu)\}$
- $J_2 = delete(J_1, U_f)$
- $J_3 = J_2 \cup \{$"$x - t_f = 0$" $\mid x \in U_f\}$
- $J_4 = delete(J_3, \{t \mid t \notin enabled(\mu')\})$
- $J_5 = J_4 \cup \{$"$t - x \geq Eft(t)$" $\mid t \in enabled(\mu'),\ t \notin enabled(\mu),\ x \in U_f\}$
  $\cup \{$"$t - x \leq Lft(t)$" $\mid t \in enabled(\mu'),\ t \notin enabled(\mu),\ x \in U_f\}$
- $J_6 = delete(J_5, \mathcal{P} - \mathcal{P}_\varphi)$

Intuitively, this can be read as follows: J_1 describes that t_f fires first, i.e. earlier than other enabled transitions. J_2 is obtained from J_1 by eliminating all time variables U_f which have to be updated. This updating is then done in J_3 by fixing the value of these variables to be equal to the firing time of t_f. In J_4 the transition variables of disabled transitions are deleted. J_5 relates the transition variables of newly enabled transitions to the updated time variables. Finally, all irrelevant time variables are removed. Note that our definition of the J_i's contains some redundancies; e.g. J_6 can be computed by using the operation $delete(I, U)$ only once. For any α and t_f, J_6 is uniquely determined (up to isomorphism); we say J_6 is *obtained by firing t_f from α*. $\alpha' = (\mu', I')$ is a *successor atom* of α, if $I' \in CE(J_6, \varphi)$ for some J_6 obtained by firing an enabled transition t_f from α. An *atom sequence* ϱ is a finite or infinite sequence $\varrho = (\alpha_0, \alpha_1, \alpha_2, \ldots)$, such that α_0 is the initial atom and α_{i+1} is a successor atom of α_i for any $i \geq 0$. The *atom graph* $G_\alpha(N, \varphi)$ consists of all atoms reachable by a finite atom sequence.

Given any atom sequence ϱ, satisfaction of φ in ϱ ($\varrho \Vdash \varphi$) is defined in an obvious way. Moreover, it can be proved that for any run ρ there exists an atom sequence ϱ such that $\rho \models \varphi$ iff $\varrho \Vdash \varphi$ and vice versa. Thus the question of whether there exists a run of N satisfying φ can be reduced to the question of whether there exists a satisfying atom sequence.

If φ contains no time variables, then $G_\alpha(N, \varphi)$ is finite as shown in [BD91]. Otherwise, however, an infinite number of different atoms may be reachable from the initial atom, because the difference $x - y$ between some time variables may become arbitrarily large, e.g. $\alpha_1 = (\mu, I \cup \{$"$x - y > 5$"$\})$, $\alpha_2 = (\mu, I \cup \{$"$x - y > 17$"$\})$, $\alpha_3 = (\mu, I \cup \{$"$x - y > 99$"$\})$, and so on. In this case, however, every atomic proposition $x - y \leq c$ and $y - x \leq c$ will eventually become constantly false and true, respectively, and thus all α_i in which $x - y$ surpasses a certain threshold value can be considered to be equivalent.

Let *max_const* be the absolute value of the maximal constant appearing in any subformula of φ, and let I be a set of inequalities. A time variable $x \in \mathcal{P}_\varphi$ is called *saturated in* I, if there is no transition variable $t \in var(I)$ such that the set $I \cup \{\text{“}t - x \leq max\_const\text{”}\}$ is consistent. For any two atoms $\alpha_1 = (\mu, I_1)$ and $\alpha_2 = (\mu, I_2)$, let $D = \{x \mid x \text{ is saturated in } I_1 \text{ and } I_2\}$. α_1 and α_2 are *equivalent*, denoted by $\alpha_1 \simeq \alpha_2$, if $I_1 \cap Cl(\varphi) = I_2 \cap Cl(\varphi)$ and $delete(I_1, D) = delete(I_2, D)$, that is, if the same maximal consistent set of φ is a subset of both I_1 and I_2 and the timing relations of I_1 and I_2 with respect to unsaturated variables are isomorphic.

From these definitions we can prove, using similar techniques as in [ACD90]:

Theorem 1. *1. $\simeq$ is a bisimulation; that is, for any α_1 and α_2 such that $\alpha_1 \simeq \alpha_2$, and for any α'_1 which is a successor of α_1 there exists a successor α'_2 of α_2 such that $\alpha'_1 \simeq \alpha'_2$.*
2. $\simeq$ is an equivalence relation of finite index.

Therefore, there exists a finite set G of representative atoms such that for any atom α reachable from the initial atom there is an equivalent atom $\alpha' \in G$, and for any atom sequence $\varrho_1 = (\alpha_0, \alpha_1, \alpha_2, \ldots)$ there is a corresponding sequence $\varrho_2 = (\alpha'_0, \alpha'_1, \alpha'_2, \ldots)$ in G such that $\alpha_i \simeq \alpha'_i$ $(i \geq 0)$ and thus $\varrho_1 |\!|\!\!- \varphi$ iff $\varrho_2 |\!|\!\!- \varphi$. G is constructed by depth–first–search from the initial atom, where the equivalence of atoms can be checked efficiently using hash–tables. Note, however, that in general the size of G is exponential in the size of the net.

Now, model checking can be performed by building the product of G with the set of all sets γ of subformulas of φ, eliminating from this product all pairs (α, γ) inconsistent with φ, and decomposing the resulting graph into maximal strongly connected components. φ is satisfiable by N iff there is a self–fulfilling strong component, i.e. one which contains with any pair (α_1, γ_1) and any formula $(\varphi_1 \,\mathcal{U}\, \varphi_2) \in \gamma_1$ also an pair (α_2, γ_2) such that $\varphi_2 \in \gamma_2$.

5 Efficiency Improvement by Partial Orders

In this section we show how to reduce the size of the atom graph of a given net and formula without affecting the correctness of the model checking procedure. The reduced state space is obtained by considering a coarser equivalence on atoms than the one defined in the previous section. It satisfies the requirement that for any atom sequence in G there exists a stuttering equivalent (w.r.t. atomic propositions in φ) atom sequence in the reduced state space, and vice versa.

Given any atom α_0 and transitions t, t' enabled in α_0, we say that t is *independent from* t' with respect to α and φ, if for any atom sequence $\varrho' = (\alpha_0, \alpha'_1, \alpha'_2, \ldots)$ such that α'_1 is obtained by firing t' from α_0 there exists a stuttering equivalent (w.r.t. atomic propositions in φ) atom sequence $\varrho = (\alpha_0, \alpha_1, \alpha_2, \ldots)$ such that α_1 is obtained by firing t from α_0. Otherwise, t is called *dependent on* t' (w.r.t. α, φ). Note that this relation is *not* symmetric!

If t is independent from t' we do not have to consider the firing of t' when generating the successors of α in the depth–first–search; there will be a stuttering equivalent sequence constructed by the firing of t.

However, the above definition is not effective; there is no efficient way to compute the set of independent transitions for a given t and α. Therefore, subsequently we give an algorithm to compute an approximation, that is a set $\mathit{dependent}(t, \alpha, \varphi)$, or $\mathit{dependent}(t, \alpha)$ in short, such that t is independent from all transitions not in this set. This idea is similar to the *stubborn set theory* of [Val90] and the *interleaving set temporal logic* of [KP90]

Of course, $\mathit{dependent}(t, \alpha)$ should be as small as possible. For example, if the net N consists of two unconnected subnets N_1 and N_2, and φ mentions only places from N_1, then certainly all transition in N_1 are independent from any transition in N_2. E.g, we don't have to consider the different interleavings of t_2 with t_3, t_4 and t_7 in our example net N_x (shown in Fig. 1) for the formulas given at the end of Sect. 3.

On the other hand, if for some t, t' which are in *conflict* (i.e. $\bullet t \cap \bullet t' \neq \emptyset$), both t and t' are firable in α, then the firing of t' inhibits that of t; thus t is not independent from t', and t' should be in the dependent set of t. So, in N_x, for every firing of t_4 also the alternative of firing t_7 should be considered.

Furthermore, disabled conflicting transitions t' can inhibit the firing of t if they can become enabled by the firing of other (enabled) transitions, and the firing of t' can overlap with that of t. In the example, although t_6 (in conflict with t_3) is disabled, it may inhibit the firing of t_3, since it can become enabled by the firing of t_4 and t_5. However, t_6 will not inhibit the firing of t_3 if t_6 becomes enabled too late. Thus, the dependency relation has to respect the timing in the net. This can be checked by examining the minimal time difference between the next firing times of t_4 and t_3. It takes at least $\mathit{Eft}(t_5) + \mathit{Eft}(t_6)$ $(= 7)$ time units to fire t_6 after the firing of t_4. Thus, t_4 can only inhibit the firing of t_3, if t_4 can fire 7 time units earlier than t_3. Hence, we include t_4 in the dependent set of t_3 only if $I \cup \{$"$t_3 - t_4 \geq \mathit{diff}(t_4, t_6)$"$\}$ is consistent, where $\mathit{diff}(t, t')$ is the minimal value of sums of earliest firing times in the paths from t to t' ($\mathit{Eft}(t)$ is not included).

Given any atom $\alpha = (\mu, I)$, firable transition t_f and disabled transition t, we therefore have to find a set of firable transitions such that the firing of any transition in this set could make t fire before t_f fires, and that the firing of t is preceded by the firing of at least one transition in this set. A set Υ of transitions is *necessary* for t, if $\Upsilon = \{t' \mid p \in t'\bullet\}$ for some $p \in \bullet t - \mu$. $\mathit{necessary}^*(t, \alpha)$ is any set of transitions containing t which is transitively closed under necessity, that is, for any $t' \in \mathit{necessary}^*(t, \alpha)$ such that t' is disabled in μ there exists a set Υ of transitions necessary for t' with $\Upsilon \subseteq \mathit{necessary}^*(t, \alpha)$. For example, $\mathit{necessary}^*(t_6, \alpha_0) = \{t_6, t_5, t_4\}$ in Fig. 1.

A transition t_h in $\mathit{necessary}^*(t, \alpha)$ is *harmful for* t_f, if it is firable, and $I \cup \{$"$t_f - t_h \geq \mathit{diff}(t_h, t)$"$\}$ is consistent. If t is in conflict with t_f, then all harmful transitions for t_f in $\mathit{necessary}^*(t, \alpha)$ have to be fired as alternatives to the firing of t_f. The only transition which is harmful for t_3 in our above example is t_4.

There is still another class of dependent transitions. We want to obtain stuttering equivalence with respect to the atomic propositions of φ. Usually, φ contains only a few propositional and time variables. A transition t is *visible for* φ if $\bullet t \cup t\bullet$ contains any place p such that p or $p^\bullet$ or p° appears in φ. If t is visible, the firing order with other visible transitions is important. For example, both t_2 and t_3 are visible for the formula $(p_1 \mathcal{U} p_3)$ in the example net, thus the firing order between t_2 and t_3 is relevant for the evaluation of $(p_1 \mathcal{U} p_3)$, and t_2 should be in the dependent set of t_3, and vice versa. A visible transition can be regarded as being in conflict with all other visible transitions. Let $\mathit{conflict}^+(t)$ be the set $\{t' \mid \bullet t' \cap \bullet t \neq \emptyset\} \cup \{t\}$, if t is not visible, else $\mathit{conflict}^+(t)$ is $\{t' \mid \bullet t' \cap \bullet t \neq \emptyset\} \cup \{t' \mid t' \text{ is visible }\}$. Then $\mathit{dependent}(t_f, \alpha)$ is any set of transitions such that for every $t \in \mathit{conflict}^+(t_f)$ there exists a set $\mathit{necessary}^*(t, \alpha)$ such that all harmful transitions for t_f in $\mathit{necessary}^*(t, \alpha)$ are contained in $\mathit{dependent}(t_f, \alpha)$.

Finally, the set of transitions which are fired should be transitively closed under dependency; e.g., in our example, since t_4 is in the dependent set of t_3 and t_7 is in the dependent set of t_4, we have to fire t_7 as an alternative whenever we fire t_3 (p_{10} is only reachable by *first* firing t_7 and *then* t_4). Thus, let $\mathit{ready}(\alpha)$ be a smallest set of firable transitions, such that for any $t_f \in \mathit{ready}(\alpha)$, $\mathit{dependent}(t_f, \alpha) \subseteq \mathit{ready}(\alpha)$.

Now, we can prove:

Theorem 2. *For any atom sequence $\varrho \in G$ there exists a stuttering equivalent atom sequence $\varrho' = (\alpha_0, \alpha_1, \alpha_2, \ldots)$ such that for any $i \geq 0$, α_{i+1} is obtained by firing some transition in $\mathit{ready}(\alpha_i)$.*

Therefore, during the construction of the set of successor atoms of an atom we can neglect all firable transitions which are not ready. This results in a considerable average case reduction: For example, in Fig. 1, $\mathit{firable}(\alpha_0) = \{t_2, t_3, t_4, t_7\}$, whereas $\mathit{ready}(\alpha_0) = \{t_2\}$.

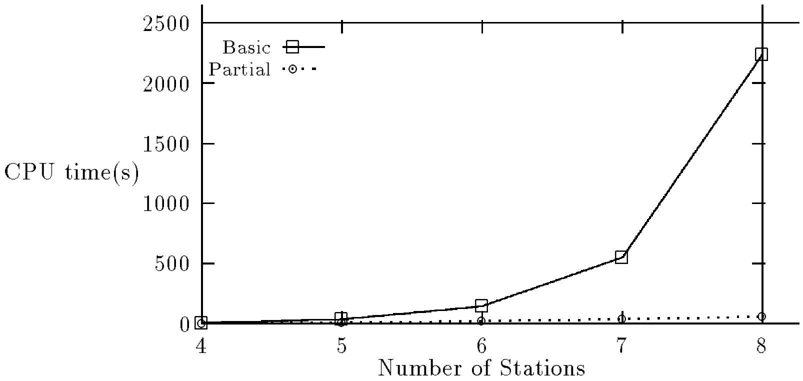
The formal description for constructing the reduced atom graph is shown in the Appendix. Though the worst case complexity of the construction of the set $\mathit{ready}(\alpha)$ is $O(|P| \cdot |T|^2)$, usually this takes only $O(|T|)$ steps with a small constant of about two or three.

6 Experimental Results

In this section, the performance of the basic model checking algorithm and its partial order improvement is demonstrated. We have implemented both the basic model checking algorithm and its partial order improvement on a 17 MIPS UNIX workstation in C++. In this section, the performance of both algorithms with an example from [RB86] is demonstrated.

The verified system called PROWAY is a local area network linking stations by a shared hardware bus. The bus allocation procedure is based on a token bus access technique. As example property, we verify if the next activity will always occur within some constant time units, say *max*, after a station finishes sending its message. This property holds in the system if the **TNL** formula $\neg\Box[\mathit{finish} \rightarrow (\neg \mathit{activity})\,\mathcal{U}\,(\mathit{activity}^{\bullet} - \mathit{finish}^{\bullet} \leq \mathit{max})]$ is not satisfiable.

The following Figure shows the CPU times for both implemented algorithms with this example. The size of the net is linear in the number n of stations; thus the basic algorithm is exponential in n. Since all stations operate more or less independently, parallelism also increases with n; therefore, the partial order method succeeds in reducing the complexity. This result is typical for a number of similar examples.



7 Conclusion

In this paper, we have proposed a timed temporal logic for time Petri nets which is expressive enough to formalize quantitative timing properties and yet it is stuttering invariant, so that the parallelism in the nets can be used to avoid the state explosion problem during verification.

Then, we have developed a model checking algorithm for our logic. We constructed for the infinite state space of the net a finite representation, the atom graph, such that every atom sequence represents a set of runs, and satisfies the formula iff the corresponding runs satisfy the formula.

Since the complexity of the consequent model checking algorithm depends on the number of atoms, we have shown how to reduce this number by elimination of redundant interleavings. In our method, for every firable transition all dependent sets, i.e. sets of firable transitions whose firings are relevant for the evaluation of the given formula, are computed. From the smallest set of firable transitions which is closed under dependency the reduced atom graph is generated. Since this set is usually much smaller than the set of all firable transitions, a considerable reduction of the state space is achieved.

Although the worst case complexity of the problem is exponential, experimental results from several examples show that the proposed algorithm successfully reduces the average complexity of the model checking.

In the future we intend to combine our method with symbolic model checking techniques (which represent state spaces as binary decision diagrams), and to find similar efficient model checking algorithms for other kinds of temporal logics such as branching time temporal logics and timed μ–calculi.

Acknowledgment: We would like to thank K.McMillan for many helpful discussions.

Appendix

Let $\alpha = (\mu, I)$ be an atom, t_f a transition in *firable*(μ), $\mu' = (\mu - \bullet t_f) \cup t_f \bullet$, and $U_f = \{p^\circ \mid p \in \bullet t_f\} \cup \{p^\bullet \mid p \in t_f \bullet\}$. We define the following sets:

- $J_1 = I \cup \{\text{“}t - t_f \geq 0\text{”} \mid t \in ready(\alpha)\}$
- $J_2 = delete(J_1, U_f)$
- $J_3 = J_2 \cup \{\text{“}x - t_f = 0\text{”} \mid x \in U_f\}$
- $J_4 = delete(J_3, \{t \mid t \notin enabled(\mu')\})$
- For $t \in T$ and $x \in \mathcal{P}$,
  $T_1(t, x) = \{\text{“}t - x \geq Eft(t)\text{”}, \text{“}t - x \leq Lft(t)\text{”}\} \cup \{\text{“}y^\bullet \leq x\text{”} \mid y \in \bullet t\}$
- $p \in P$ is called a *candidate of true parents* of a transition t_c in μ and I, if $p \in \mu \cap \bullet t_c$ and for some enabled transition t in μ, $I \cup \{\text{“}p^\bullet \geq t + diff(t, t_c) - Eft(t_c)\text{”}\}$ is consistent.
- $T_2 = \prod_{t \in enabled(\mu'), t \notin enabled(\mu)} \{T_1(t, p^\bullet) \mid p$ is a candidate of true parents of t in μ' and $J_4\}$, where $\prod$ represents the Cartesian product.
- $J_5 = \{\hat{I} \mid \hat{I} = J_4 \cup a_1 \cup a_2 \cup \cdots \cup a_i, (a_1, a_2, \cdots, a_i) \in T_2,\ \hat{I}$ is consistent$\}$
- $J_6 = \{delete(\hat{I}, \mathcal{P} - \mathcal{P}_\varphi - D) \mid \hat{I} \in J_5\}$, where $D = \{p^\bullet \mid p$ is a candidate of true parents of some disabled transition in μ' and $\hat{I}\}$

$\alpha' = (\mu', I')$ is a *successor atom* of α, if $I' \in \bigcup_{\hat{I} \in J_6} CE(\hat{I}, \varphi)$. The finite reduced atom graph G is constructed in the same way as shown in the end of Sect. 4.

References

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. *Proc. 5th IEEE LICS*, 1990.

[AH89] R. Alur, T. Henzinger. A really temporal logic. *Proc. 30th FOCS*, 1989.

[BD91] B. Berthomieu, M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. on Soft. Eng.*, 17(3):259–273, 1991.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite–state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, 1986.

[God90] P. Godefroid. Using partial orders to improve automatic verification methods. *Proc. 1st CAV*, 1990.

[JM87] F. Jahanian and A. K. Mok. A graph–theoretic approach for timing analysis and its implementation. *IEEE Trans. Comput.*, C-36(8):961–975, 1987.

[KP90] S. Katz and D. Peled. Defining conditional independence using collapses. *Semantics for concurrency, BCS–FACS Workshop, Springer*, 1990.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. *Proc. 12th POPL*, 1985.

[MF76] P. Merlin and D. J. Faber. Recoverability of communication protocols. *IEEE Trans. on Communication*, COM-24(9), 1976.

[RB86] J-L. Roux and B. Berthomieu. Verification of a local area network protocol with Tina, a software package for time Petri nets. *7th European Workshop on Application and Theory of Petri Nets*, pages 183–205, 1986.

[Val90] A. Valmari. A stubborn attack on state explosion. *Proc. 1st CAV*, 1990.

[YTK91] T. Yoneda, Y. Tohma, Y. Kondo. Acceleration of timing verification method based on time Petri nets. *Syst. and Computers in Japan*, 22(12):37–52, 1991.