Fundamenta Informaticae 190(2-4) : 63–107 (2023)
Available at IOS Press through:
`https://doi.org/10.3233/FI-232167`

Waiting Nets: State Classes and Taxonomy

Loïc Hélouët[*]
University Rennes, Inria, CNRS, IRISA, France
loic.helouet@inria.fr

Pranay Agrawal
ENS-Paris-Saclay, France
pranay.agrawal@ens-paris-saclay.fr

Abstract. In time Petri nets (TPNs), time and control are tightly connected: time measurement for a transition starts only when all resources needed to fire it are available. Further, upper bounds on duration of enabledness can force transitions to fire (this is called *urgency*). For many systems, one wants to decouple control and time, i.e. start measuring time as soon as a part of the preset of a transition is filled, and fire it after some delay and when all needed resources are available. This paper considers an extension of TPN called *waiting nets* that dissociates time measurement and control. Their semantics allows time measurement to start with incomplete presets, and can ignore urgency when upper bounds of intervals are reached but all resources needed to fire are not yet available. Firing of a transition is then allowed as soon as missing resources are available. It is known that extending bounded TPNs with stopwatches leads to undecidability. Our extension is weaker, and we show how to compute a finite state class graph for bounded waiting nets, yielding decidability of reachability and coverability. We then compare expressiveness of waiting nets with that of other models w.r.t. timed language equivalence, and show that they are strictly more expressive than TPNs.

Keywords: Time, Petri nets, State Classes.

[*]Address for correspondence: University Rennes, Inria, CNRS, IRISA, France.

1. Introduction

Time Petri nets (TPNs) [1] are an interesting model to specify cyber-physical systems. They allow for the specification of concurrent or sequential events (modeled as transitions occurrences), resources, time measurement, and urgency. In TPNs, time constraints are modeled by attaching an interval $[\alpha_t, \beta_t]$ to every transition t. If t has been enabled for at least α_t time units it *can* fire. If t has been enabled for β_t time units, it is *urgent*: time cannot elapse, and t *must* either fire or be disabled. Urgency is an important feature of TPNs, as it allows for the modeling of strict deadlines, but gives them a huge expressive power. In their full generality, TPNs are Turing powerful. A consequence is that most properties that are decidable for Petri Nets [2] (coverability [3], reachability [4], boundedness [3]...) are undecidable for TPNs. Yet, for the class of bounded TPNs, reachability [5] and coverability are decidable. The decision procedure relies on a symbolic representation of states with *state classes* and then on the definition of abstract runs as paths in a so-called state class graph [6, 7].

There are many variants of Petri nets with time. An example is *timed Petri nets* (TaPN) [8], where tokens have an age, and time constraints are attached to arcs of the net. In TaPNs, a token whose age reaches the upper bound of constraints becomes useless. The semantics of TaPNs enjoys some monotonicity, and well-quasi-ordering techniques allow to solve coverability or boundedness problems [9, 10]. However, reachability of a given marking remains undecidable for TaPNs [11]. Without any notion of urgency, TaPN cannot model delay expiration. In [12], a model mixing TaPN and urgency is proposed, with decidable coverability, even for unbounded nets. We refer readers to [13] for a survey on TaPNs and their verification, and to [14] for a survey on different ways to introduce time in Petri nets.

Working with bounded models is enough for many cyber-physical systems. However, bounded TPNs suffer another drawback: time measurement and control are too tightly connected. In TPNs, time is measured by starting a new clock for every transition that becomes enabled. By doing so, measuring a duration for a transition t starts only when all resources needed to fire t are available. Hence, one cannot stop and restart a clock, nor start measuring time while waiting for missing resources. To solve this problem, [15] equips bounded TPNs with stopwatches. Nets are extended with read arcs, and the understanding of a read arc from a place p to a transition t is that when p is filled, the clock attached to t is frozen. Extending bounded TPNs with stopwatches leads to undecidability of coverability, boundedness and reachability. This is not a surprise, as timed automata with stopwatches are already a highly undecidable model [16]. For similar reasons, time Petri nets with preemptable resources [17], where time progress depends on the availability of resources cannot be formally verified.

This paper considers *waiting nets*, a new extension of TPN that decouples time measurement and control. Waiting nets distinguish between enabling of a transition and enabling of its firing, which allows rules of the form "start measuring time for t as soon as p is filled, and fire t within $[\alpha, \beta]$ time units when p and q are filled". Syntactically, this model is a simple extension of TPNs: the model is built by adding a particular type of *control place* to TPNs. Waiting nets allow clocks of enabled transitions to reach their upper bounds, and wait for missing control to fire. A first contribution of this paper is a formal definition of the semantics of this new model. An immediate result is that waiting nets are more expressive than TPNs, as TPNs are a simple syntactic restriction of waiting nets. A former attempt called Timed Petri nets with Resets (TPNR) distinguishes some delayable transitions that can fire later than their upper bounds [18]. For bounded TPNR, reachability and TCTL model checking are decidable. However, delayable transitions are never urgent, and once delayed can only fire during a maximal step with another transition fired on time. Further, delayable transitions start measuring time as soon as their preset is filled, and hence do not allow decoupling of time and control as in waiting nets.

As a second contribution of this paper, we define state class graphs for waiting nets. We build on the work of [6] to define domains, i.e. symbolic representations for the values of clocks that measure time elapsed since enabling of transitions. We then define a successor relation via transition firing among domains. This allows for the construction of a state class graph depicting reachable markings and domains of a waiting net. We show that the state class graphs of bounded waiting nets are finite, yielding decidability of reachability and coverability (which are PSPACE-complete). This is a particularly interesting result, as these properties are undecidable for *stopwatch Petri nets*, *even in the bounded case*. The table 1 summarizes known decidability results for reachability, coverability and boundedness problems for time variants of Petri nets, including the new results for waiting nets proved in this paper. Another interesting result shown in this paper is that a state class may have more that one successor via a transition. This shows that state class graphs of waiting nets are not deterministic, and that TPNs and waiting nets are hence different models.

Table 1. Decidability and complexity results for time(d) variants of Petri nets.

	Reachability	coverability	Boundedness
Time Petri Nets	Undecidable [19]	Undecidable [19]	Undecidable [19]
(bounded)	Decidable	Decidable	–
Timed Petri nets	Undecidable [11]	Decidable [20, 9]	Decidable [20]
(bounded)	Decidable	Decidable	–
Restricted Urgency	Undecidable [12]	Decidable [12]	Decidable [12]
(bounded)	Decidable	Decidable	–
Stopwatch Petri nets	Undecidable [15]	Undecidable [15]	Undecidable [15]
(bounded)	Undecidable [15]	Undecidable [15]	–
TPNR	Undecidable [18]	Undecidable [18]	Undecidable [18]
(bounded)	Decidable [18]	Decidable [18]	–
Waiting Nets	Undecidable (Rmk. 3.6)	Undecidable (Rmk. 3.6)	Undecidable (Rmk. 3.6)
(bounded)	PSPACE-Complete (Thm. 4.12)	PSPACE-Complete (Thm. 4.12)	–

Our last contribution is a study of the expressiveness of waiting nets w.r.t. timed language equivalence. We compare the timed languages generated by timed automata, TPNs, waiting nets. We consider variants of these models with injective or non-injective labelings, and with $\epsilon-$transitions. Interestingly, the expressiveness of bounded waiting nets lays between that of bounded TPNs and timed automata.

This paper is an extended version of an article originally published in [21]. The rest of the paper is organized as follows: we recall the formal background needed later in the paper in Section 2. We recall in particular definitions of clock constraints, timed automata and timed languages. We define the syntax and semantics of waiting nets in Section 3. Section 4 shows a state class graph construction for waiting nets and proves finiteness of the set of domains of waiting nets. Finiteness of the obtained state class graph is then used to show PSPACE completeness of reachability and coverability for bounded waiting nets. Section 5 compares the expressive power of waiting nets with that of other models and with other timed variants of Petri nets before conclusion. For readability, several technical proofs of Section 4 are only sketched, but are provided in Appendix.

2. Preliminaries

We denote by $\mathbb{R}^{\geq 0}$ the set of non-negative real values, and by $\mathbb{Q}$ the set of rational numbers. A *rational interval* $[\alpha, \beta]$ is the set of values between a lower bound $\alpha \in \mathbb{Q}$ and an upper bound $\beta \in \mathbb{Q}$. We also consider intervals without upper bounds of the form $[\alpha, \infty)$, to define values that are greater than or equal to α.

A *clock* is a variable x taking values in $\mathbb{R}^{\geq 0}$. A variable x_t will be used to measure the time elapsed since transition t of a net was last newly enabled. Let X be a set of clocks. A *valuation* for X is a map $v : X \to \mathbb{R}^{\geq 0}$ that associates a positive or zero real value $v(x)$ to every variable $x \in X$. Intervals alone are not sufficient to define the domains of clock valuations met with TPNs and timed automata. An *atomic constraint* on X is an inequality of the form $a \leq x$, $x \leq b$, $a \leq x - y$ or $x - y \leq b$ where $a, b \in \mathbb{Q}$ and $x, y \in X$. A *constraint* is a conjunction of atomic constraints. We denote by $Cons(X)$ the set of constraints over clocks in X. We will say that a valuation v satisfies a constraint ϕ, and write $v \models \phi$ iff replacing x by $v(x)$ in ϕ yields a tautology. A constraint ϕ is *satisfiable* iff there exists a valuation v for X such that $v \models \phi$. Constraints over real-valued variables can be efficiently encoded with Difference Bound Matrices (DBMs) and their satisfiability checked in $O(n^3)$ [22]. For completeness, we give a formal definition of DBMs in Appendix C. We also show in Appendix B that checking satisfiability of a constraint amounts to detecting negative cycles in a weighted graph, which can be checked using a variant of the Floyd-Warshall algorithm. The *domain* specified by a constraint ϕ is the (possibly infinite) set of valuations that satisfy ϕ.

Given an alphabet Σ, a *timed word* is an element of $(\Sigma \times \mathbb{R}^+)^*$ of the form $w = (\sigma_1, d_1)(\sigma_2, d_2)\dots$ such that $d_i \leq d_{i+1}$. A timed language is a set of timed words. Timed automata [23] are frequently used to recognize timed languages. In the last section of this paper, we compare expressiveness of several variants of time(d) Petri nets and timed automata. For completeness, we recall the definition and semantics of timed automata.

Definition 2.1. (Timed Automaton)
A *Timed Automaton* $\mathcal{A}$ is a tuple $\mathcal{A} = (L, \ell_0, X, \Sigma, Inv, E, F)$, where L is a set of locations, $\ell_0 \in L$ is the initial location, X is a set of clocks, Σ is an alphabet, $Inv : L \to Cons(X)$ is a map associating an invariant to every location. The set of states $F \subseteq L$ is a set of final locations, and E is a set of edges. Every edge is of the form $(\ell, g, \sigma, R, \ell') \in L \times Cons(X) \times \Sigma \times 2^X \times L$.

Intuitively, the semantics of a timed automaton allows elapsing time in a location ℓ (in which case clocks valuations grow uniformly) if this does not result in a valuation that violates $Inv(\ell)$, or firing a discrete transition $(\ell, g, \sigma, R, \ell')$ from location ℓ with clock valuation v if v satisfies guard g, and the valuation v' obtained by resetting all clocks in R to 0 satisfies $Inv(\ell')$. One can notice that invariants can prevent firing a transition. Every run of a timed automaton starts from (ℓ_0, v_0), where v_0 is the valuation that assigns value 0 to every clock in X.

Let $\mathcal{A} = (L, \ell_0, X, \Sigma, Inv, E, F)$ be a timed automaton. A configuration of $\mathcal{A}$ is a pair (ℓ, v) where $\ell \in L$ is a location, and v a valuation of clocks in X. Let $v_{R:=0}$ denote a valuation v' such that $v'(x) = 0$ if $x \in R$ and $v'(x) = v(x)$ otherwise. A *discrete move* via transition $e = (\ell, g_e, \sigma_e, R, \ell')$ is allowed from (ℓ, v) iff $v \models g_e$ (the guard of the transition is satisfied) and $v_{R:=0} \models Inv(\ell')$. We will denote such moves by $(\ell, v) \xrightarrow{e} (\ell', v')$. Let $d \in \mathbb{R}^+$, and let $v + d$ denote a valuation such that $v + d(x) = v(x) + d$ for every $x \in X$. A *timed move* of d time units is allowed if $v + d' \models Inv(\ell)$ for every real value $d' \leq d$. We will denote such moves by $(\ell, v) \xrightarrow{d} (\ell', v + d)$.

A *run* of $\mathcal{A}$ is a sequence of discrete and timed moves that starts from configuration (ℓ_0, v_0), where v_0 is the valuation that associates value 0 to every clock in X. A run is accepting if it ends in a final location. Without loss of generality, we will assume that runs of timed automata alternate timed and discrete moves (possibly of duration 0). A run is hence a sequence of the form $\rho = (\ell_0, v_0) \xrightarrow{d_0} (\ell, v_0 + d_0) \xrightarrow{e_0} (\ell_1, v_1) \ldots$, where each $d_i \in \mathbb{R}$ is a duration, and each e_i is a transition. The *timed word* associated with run ρ is the word $w_\rho = (\sigma_{e_0}, dt_0) \ldots (\sigma_{e_i}, dt_i) \ldots$ where $dt_i = \sum_{k=0}^{k=i-1} d_k$. The *(timed) language* recognized by $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A})$, and is the set of timed words associated with accepting runs of $\mathcal{A}$. The definition of timed automata can be easily extended to include silent transitions. Timed automata then contain transitions of the form $e = (\ell, g_e, \epsilon, R_e, \ell')$ labeled by an unobservable letter denoted ϵ, and are hence automata over an alphabet $\Sigma \cup \epsilon$. In this case, the word recognized along a run ρ is the projection of w_ρ on pairs (σ_{e_i}, dt_i) such that $\sigma_{e_i} \neq \epsilon$.

In the rest of the paper, we will denote by TA the class of timed automata. We will be in particular interested by the subclass $TA(\leq, \geq)$ in which guards are conjunctions of atomic constraints of the form $x \geq c$ and invariants are conjunctions of atomic constraints of the form $x \leq c$. Several translations from TPNs to TAs have been proposed (see for instance [24, 7]). In particular, the solution of [7] uses the state class graph of a TPN to build a time-bisimilar timed automaton in class $TA(\leq, \geq)$. This shows that one needs not the whole expressive power of timed automata to encode timed languages recognized by TPNs.

3. Waiting nets

TPNs are a powerful model: they can be used to encode a two-counter machine (see for instance [10] for an encoding of counter machines), and can hence simulate the semantics of many other formal models. A counterpart to this expressiveness is that most problems (reachability, coverability, boundedness, verification of temporal logics...) are undecidable. Decidability is easily recovered when considering the class of bounded TPNs. Indeed, for bounded TPNs, one can compute a finite symbolic model called a state class graph [6], in which timing information is symbolically represented by firing domains. For many applications, working with bounded resources is sufficient. However, TPNs do not distinguish between places that represent control (the "state" of a system), and those that represent resources: transitions are enabled when all places in their preset are filled. A consequence is that one cannot measure time spent in a control state, when some resources are missing.

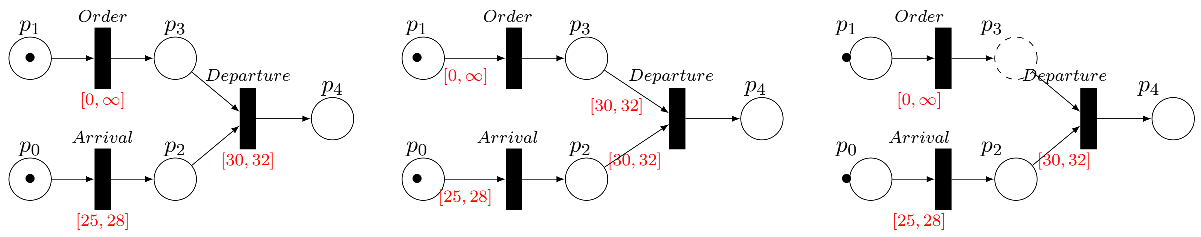


Figure 1. A TPN $a)$ a Timed Petri net $b)$, and a Waiting net $c)$

Consider the examples of Figure 1, that are designed to represent an arrival of a train followed by a departure. The arrival in a station is modeled by transition $Arrival$, that should occur between 25 and 28 minutes after the beginning of a run. The station is modeled by place p_2, and the departure of the train by transition $Departure$. A train can leave a station only if a departure order has been sent, which is modeled by transition $Order$. Assume that one wants to implement a scenario of the form "the train leaves the station between 30 and 32 minutes after its arrival if it has received a departure order".

Consider the TPN of Figure 1-a). The time constraint attached to $Departure$ is an interval of the form $[30, 32]$, which is the only way to impose a precise timing in TPNs. This TPN does not implement our scenario, but rather behaviors in which the train leaves the station between 30 and 32 minutes after the instant when it is in station **and** a departure order is received. This means that a train may spend $32 + d$ minutes in station, if the order is released d minutes after its arrival. Now, consider the second net in Figure 1-b). It is a timed Petri net, i.e. a variant of nets where tokens are given an age (the durations that have elapsed since they were created in a place) and flow relations constrain the age of consumed tokens. In a nutshell, the semantics of timed Petri nets allow consumption of tokens which age satisfy the timing constraints of flow relations. Tokens that become too old to satisfy a constraint are never consumed and can be ignored. In the net of Figure 1-b), a token can move from place p_1 to place p_3 at any time (i.e. a departure order can be issued at any instant), and a second token can move from place p_0 to place p_2 when its age is between 25 and 28 minutes, representing arrival of a train within a delay ranging from 25 to 28 minutes. Following the semantics of timed Petri nets, one can simply "forget" the token in place p_2 if the delay to produce a token in place p_3 exceeds $28 + 32$. In other words, in this model, one can forget a train arrived in station if the departure order arrives more than 32 minutes after its $Arrival$. This is not the scenario we want to model. Further, there is no timed Petri net that can specify this example, because timed Petri nets do not have a notion of urgency, and hence cannot enforce occurrence of an action after a predetermined delay (in our case a departure of a train at latest after 32 minutes).

We propose an extension of TPNs called *waiting nets* (WTPNs for short), that decouples control and resources during time measurement. We consider two types of places: *standard* places, and *control* places, with the following functions: Time measurement for a transition t starts as soon as t has enough tokens in the standard places of its preset. Then, t can fire if its clock value lays in its timing interval, **and** if it has enough tokens in the control places of its preset. To illustrate this extension, consider the example of Figure 1-c). This example is similar to the net of Figure 1-a), but with p_3 replaced by a control place (symbolized by a dashed circle). Let us detail the changes brought by control places. Let transition $Arrival$ fire at a date d between 25 and 28 minutes, filling place p_2. This immediately enables transition $Departure$, that is forced to fire at latest 32 time units later if transition $Order$ is fired at a date smaller than $d + 32$, or exactly at the date of firing of transition $Order$ if this transition is fired more than 32 minutes after date d. This net models a situation where a train can arrive at date $d \in [25, 28]$ minutes, and leave as soon as at least 30 minutes have elapsed since arrival, and a departure order was given. If the departure order is received at a date smaller than $d + 32$, then the train can leave at latest at date $d + 32$. If the departure order is given at a date $d' > d + 32$, then the train leaves at date d', i.e. when the departure order is given. This is exactly the scenario described above. We can now define waiting nets formally as follows :

Definition 3.1. A *waiting net* is a tuple $\mathcal{W} = \big(P, C, T, {}^\bullet(), ()^\bullet, \alpha, \beta, \lambda, (M_0.N_0)\big)$, where

- P is a finite set of standard places, C is finite set of control places, such that $P \cup C \neq \emptyset$ and $P \cap C = \emptyset$. A *marking* $M.N$ is a pair of maps $M : P \to \mathbb{N}$, $N : C \to \mathbb{N}$ that associate an integral number of tokens respectively to standard and control places.
- T is a finite set of transitions. Every $t \in T$ has a label $\lambda(t) \in \Sigma \cup \epsilon$,
- ${}^\bullet() \in (\mathbb{N}^{P \cup C})^T$ is the backward incidence function, $()^\bullet \in (\mathbb{N}^{P \cup C})^T$ is the forward incidence function,
- $(M_0.N_0) \in \mathbb{N}^{P \cup C}$ is the initial marking of the net,
- $\alpha : T \to \mathbb{Q}^+$ and $\beta : T \to \mathbb{Q}^+ \cup \infty$ are functions giving for each transition respectively its earliest and latest firing times. For every transition t, we have $\alpha(t) \leq \beta(t)$.

The labeling map λ is *injective* if $\lambda(t) = \lambda(t') \Leftrightarrow t = t'$, and may allow labeling transitions with a silent action label denoted ϵ. To differentiate standard and control places in the preset of a transition, we will denote by ${}^\circ(t)$ the restriction of ${}^\bullet(t)$ to standard places, and by ${}^c()$ the restriction of ${}^\bullet()$ to control places. We will write $M(p) = k$ (resp. $N(c) = k$) to denote the fact that standard place $p \in P$ (resp. control place $c \in C$) contains k tokens. Given two markings $M.N$ and $M'.N'$ we will say that $M.N$ is greater than $M'.N'$ and write $M.N \geq M'.N'$ iff $\forall p \in P, M(p) \geq M'(p)$ and $\forall c \in C, N(c) \geq N'(C)$.

Figure 2-a) is a waiting net modelling an online sale offer, with limited duration. Control places are represented with dashed lines. A client receives an ad, and can then buy a product up to 8 days after reception of the offer, or wait to receive a coupon offered to frequent buyers to benefit from a special offer at reduced price. However, this special offer is valid only for 3 days. In this model, a token in control place p_3 represents a coupon allowing the special offer. However, time measure for the deal at special price starts as soon as the ad is sent. Hence, if the coupon is sent 2 days after the ad, the customer still has 1 day to benefit from this offer. If the coupon arrives more than 3 days after the ad, he has to use it immediately. Figure 2-b) enhances this example to model expiration of the coupon after 3 days with a transition. Transition T_O consumes urgently a token from place p_6 exactly 3 time units after firing of transition Ad if it is still enabled, which means that the special offer expires within 3 days, and coupon arriving later than 3 days after the add cannot be used.

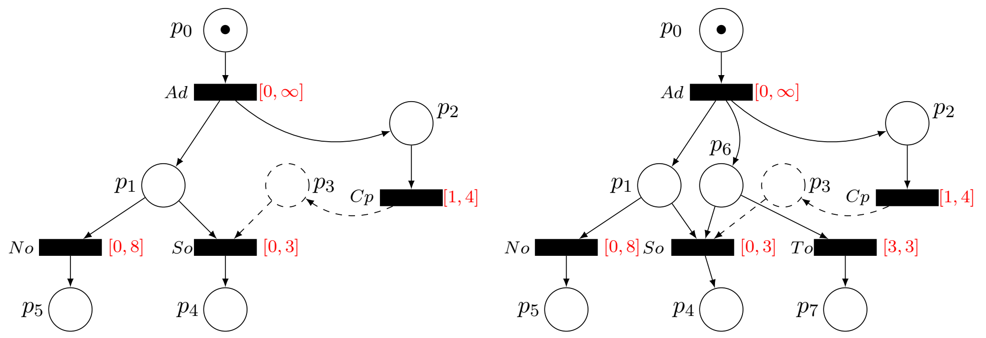


Figure 2. $a)$ Decoupled time and control in a waiting net. $b)$ ... with a timeout transition.

The semantics of waiting nets associates clocks to transitions, and lets time elapse if their standard preset is filled. It allows firing of a transition t if the standard and the control preset of t is filled.

Definition 3.2. (Enabled, Fully Enabled, Waiting transitions)

- A transition t is *enabled* in marking $M.N$ iff $M \geq {}^{\circ}(t)$ (for every standard place p in the preset of t, $M(p) \geq {}^{\circ}(t)(p)$). We denote by $\mathsf{Enabled}(M)$ the set of transitions which are enabled from marking M, i.e. $\mathsf{Enabled}(M) := \{t \mid M \geq {}^{\circ}(t)\}$.
- A transition t is *fully enabled* in $M.N$ iff, for every place in the preset of t, $M.N(p) \geq {}^{\bullet}(t)(p)$. $\mathsf{FullyEnabled}(M.N)$ is the set of transitions which are fully enabled in marking $M.N$, i.e. $\mathsf{FullyEnabled}(M.N) := \{t \mid M.N \geq {}^{\bullet}(t)\}$.
- A transition t is *waiting* in $M.N$ iff $t \in \mathsf{Enabled}(M) \setminus \mathsf{FullyEnabled}(M.N)$ (t is enabled, but is still waiting for the control part of its preset). We denote by $\mathsf{Waiting}(M.N)$ the set of waiting transitions.

Obviously, $\mathsf{FullyEnabled}(M.N) \subseteq \mathsf{Enabled}(M)$. For every enabled transition t, there is a clock x_t that measures for how long t has been enabled. For every fully enabled transition t, t can fire when $x_t \in [\alpha(t), \beta(t)]$. We adopt an *urgent* semantics, i.e. when a transition is fully enabled and $x_t = \beta(t)$, then this transition, or another one enabled at this precise instant *has to* fire without letting time elapse. Firing of a transition t from marking $M.N$ consumes tokens from all places in ${}^{\bullet}(t)$ and produces tokens in all places of $(t)^{\bullet}$. A consequence of this token movement is that some transitions are disabled, and some other transitions become enabled after firing of t.

Definition 3.3. (Transition Firing)

Firing of a transition t from marking $M.N$ is done in two steps. The first step computes an intermediate marking $M''.N'' = M.N - {}^{\bullet}(t)$ obtained by removing tokens consumed by the transition from its preset. Then, a new marking $M'.N' = M''.N'' + (t)^{\bullet}$ is computed. We will write $M.N \xrightarrow{t} M'.N'$ whenever firing of t from $M.N$ produces marking $M'.N'$. A transition t_i is *newly enabled* after firing of t from $M.N$ iff it is enabled in $M'.N'$, and either it is not enabled in $M''.N''$, or t_i is a new occurrence of t. We denote by $\uparrow \mathsf{enabled}(M.N, t)$ the set of transitions newly enabled after firing t from marking $M.N$.

$$\uparrow \mathsf{enabled}(M.N, t) := \{t_i \in T \mid {}^{\bullet}(t_i) \leq M.N - {}^{\bullet}(t) + (t)^{\bullet} \wedge \big((t_i = t) \vee ({}^{\bullet}(t_i) \geq M.N - {}^{\bullet}(t))\big)\}$$

As explained informally with the examples of Figure 2, the semantics of waiting nets allows transitions firing when some time constraints on the duration of enabling are met. Hence, a proper notion of state for a waiting net has to consider both place contents and time elapsed. This is captured by the notion of *configuration*. In configurations, time is measured by attaching a clock to every enabled transition. To simplify notations, we define valuations of clocks on a set $X_T = \{x_t \mid t \in T\}$ and write $x_t = \perp$ if $t \notin enabled(M)$. To be consistent, for every value $r \in \mathbb{R}$, we set $\perp + r := \perp$.

Definition 3.4. (Configuration)
A *Configuration* of a waiting net is a pair $(M.N, v)$ where $M.N$ is a marking and v is a valuation of clocks in X_T. The *initial configuration* of a waiting net is a pair $(M_0.N_0, v_0)$, where $v_0(x_t) = 0$ if $t \in \mathsf{enabled(M_0)}$ and $v_0(x_t) = \perp$ otherwise. A transition t is *firable* from configuration $(M.N, v)$ iff it is fully enabled, and $v(x_t) \in [\alpha(t), \beta(t)]$.

Slightly abusing our notation, we will write $v(t)$ instead of $v(x_t)$. The semantics of waiting nets is defined in terms of *timed* or *discrete* moves from one configuration to the next one. Timed moves increase the value of clocks attached to enabled transitions (when time elapsing is allowed) while discrete moves are transitions firings that reset clocks of newly enabled transitions. We give the formal operation definition of timed and discrete moves in Figure 3 and then detail these operational rules.

$$\frac{\begin{array}{l}\forall t \in \mathsf{Waiting}(M.N),\\ \qquad v'(t) = \min(\beta(t), v(t)+d)\\ \forall t \in \mathsf{FullyEnabled}(M.N),\\ \qquad v(t)+d \leq \beta(t)\\ \qquad\quad \text{and } v'(t) = v(t)+d\\ \forall t \in T \setminus \mathsf{Enabled}(M), v'(t) = \perp\end{array}}{(M.N, v) \xrightarrow{d} (M.N, v')}
\qquad
\frac{\begin{array}{c} M.N \geq {}^\bullet(t)\\ M'.N' = M.N - {}^\bullet(t) + (t)^\bullet\\ \alpha(t) \leq v(t) \leq \beta(t)\\ \forall t_i \in T, v'(t_i) = \begin{cases} 0 \text{ if } t_i \in\uparrow \mathsf{enabled}(M.N, t)\\ \perp \text{ if } t_i \notin \mathsf{enabled}(M)\\ v(t_i) \text{ otherwise}\end{cases}\end{array}}{(M.N, v) \xrightarrow{t} (M'.N', v')}$$

Figure 3. Operational semantics of Waiting Nets

A **timed move** from a configuration $(M.N, v)$ lets $d \in \mathbb{R}^{\geq 0}$ time units elapse, but leaves the marking unchanged. We adopt an *urgent semantics* that considers differently *fully enabled* transitions and *waiting* transitions. First of all, elapsing of d time units from $(M.N, v)$ is allowed only if, for every fully enabled transition t, $v(t) + d \leq \beta(t)$. If $v(t) + d > \beta(t)$ then t becomes urgent before d time units, and letting a duration d elapse from $(M.N, v)$ is forbidden. If we already have $v(t) = \beta(t)$, then no timed move is allowed, and firing of t is *urgent*: *t has to* be fired or disabled by the firing of another transition before elapsing time. On the other hand, if elapsing d time units does not violate urgency of any fully enabled transition, the new valuation for the clock x_t attached to a fully enabled transition after elapsing d time units is $v'(t) = v(t)+d$. Urgency does not apply to waiting transitions, which can let an arbitrary amount of time elapse when at least one control place in their preset is not filled. Now, as we model the fact that an event has been enabled for a sufficient duration, we let the value of clocks attached to waiting transitions increase up to the upper bound allowed by their time interval, and then freeze these clocks. So, for a waiting transition, we have $v'(t) = \min(\beta(t), v(t) + d)$. We will write

$v \oplus d$ to denote the valuation of clocks reached after elapsing d time units from valuation v. A timed move of duration d from configuration $(M.N, v)$ to $(M.N, v')$ is denoted $(M.N, v) \xrightarrow{d} (M.N, v')$. As one can expect, waiting nets enjoy time additivity (i.e. $(M.N, v) \xrightarrow{d_1} (M.N, v_1) \xrightarrow{d_2} (M.N, v_2)$ implies that $(M.N, v) \xrightarrow{d_1+d_2} (M.N, v_2)$, and continuity, i.e. if $(M.N, v) \xrightarrow{d} (M.N, v')$, then for every $d' < d$ $(M.N, v) \xrightarrow{d'} (M.N, v'')$.

A **discrete move** fires a fully enabled transition t whose clock x_t meets the time constraint of t (i.e., $\alpha(t) \leq x_t \leq \beta[t]$), and resets clocks attached to transitions newly enabled by token moves. A discrete move relation from configuration $(M.N, v)$ to $(M'.N', v')$ via transition $t_i \in \mathsf{FullyEnabled}(M.N)$ is denoted $(M.N, v) \xrightarrow{t_i} (M'.N', v')$. Overall, the semantics of a waiting net $\mathcal{W}$ is a timed transition system (TTS) with initial state $q_0 = (M_0.N_0, v_0)$ and which transition relation follows the time and discrete move semantics rules.

Definition 3.5. A *run* of a waiting net $\mathcal{W}$ from a configuration $(M.N, v)$ is a sequence

$$\rho = (M.N, v) \xrightarrow{e_1} (M_1.N_1, v_1) \xrightarrow{e_2} (M_2.N_2, v_2) \cdots \xrightarrow{e_k} (M_k.N_k, v_k)$$

where every e_i is either a duration $d_i \in \mathbb{R}^{\geq 0}$, or a transition $t_i \in T$, and every $(M_{i-1}.N_{i-1}, v_{i-1}) \xrightarrow{e_i} (M_i.N_i, v_i)$ is a legal move of $\mathcal{W}$.

We denote by $\mathsf{Runs}(\mathcal{W})$ the set of runs of $\mathcal{W}$ starting from $(M_0.N_0, v_0)$. A marking $M.N$ is *reachable* iff there exists a run from $(M_0.N_0, v_0)$ to a configuration $(M.N, v)$ for some v. $M.N$ is *coverable* iff there exists a reachable marking $M'.N' \geq M.N$. We will say that a waiting net is *bounded* iff there exists an integer K such that, for every reachable marking $M.N$ and every place $p \in P$ and $p' \in C$, we have $M(p) \leq K$ and $N(p') \leq K$. Given two markings $M_0.N_0$ and $M.N$ the *reachability* problem asks whether $M.N$ is reachable from $(M_0.N_0, v_0)$, and the *coverability* problem whether there exists a marking $M'.N' \geq M.N$ reachable from $(M_0.N_0, v_0)$.

Remark 3.6. A waiting net with an empty set of control places is a TPN. Hence, waiting nets inherit all undecidability results of TPNs: reachability, coverability and boundeness [19] are undecidable in general for unbounded waiting nets.

Given a run $\rho = (M_0.N_0, v_0) \xrightarrow{e_1} (M_1.N_1, v_1) \xrightarrow{e_2} (M_2.N_2, v_2) \cdots$, the timed word associated with ρ is the word $w_\rho = (t_1, d_1) \cdot (t_2, d_2) \cdots$ where the sequence $t_1 \cdot t_2 \ldots$ is the projection of $e_1 \cdot e_2 \cdots$ on T, and for every (t_i, d_i) such that t_i appears on move $(M_{k-1}.N_{k-1}, v_{k-1}) \xrightarrow{e_k} (M_k.N_k, v_k)$, d_i is the sum of all durations in $e_1 \ldots e_{k-1}$. The sequence $t_1.t_2 \ldots$ is called the *untiming* of w_ρ. The *timed language* of a waiting net is the set of timed words $\mathcal{L}(\mathcal{W}) = \{w_\rho \mid \rho \in \mathsf{Runs}(\mathcal{W})\}$. Notice that unlike in timed automata and unlike in the models proposed in [25], we do not define accepting conditions for runs or timed words, and hence consider that the timed language of a net is prefix closed. The *untimed language* of a waiting net $\mathcal{W}$ is the language $\mathcal{L}^U(\mathcal{W}) = \{w \in T^* \mid \exists w_\rho \in \mathcal{L}(\mathcal{W}),$ w is the untiming of $w_\rho\}$. These definitions can be adapted the obvious way to consider labels of transitions instead of transition themselves (i.e. timed words are sequences of pairs of the form $(\lambda(t_i), d_i)$, and labeling of transitions by ϵ. To simplify notations, we will consider runs alternating timed and discrete moves. This results in no loss of generality, since durations of consecutive timed moves can be summed up, and a sequence of two discrete move can be seen as a sequence of transitions with 0 delays between discrete moves. In the rest of the paper, we will write $(M.N, v) \xrightarrow{(d,t)} (M'.N', v')$ to denote the sequence of moves $(M.N, v) \xrightarrow{d} (M.N, v \oplus d) \xrightarrow{t} (M'.N', v')$.

Let us illustrate the semantics of waiting nets with the example in Figure 2-a). In this net, we have $P = \{p_0, p_1, p_2, p_4, p_5\}$, $C = \{p_3\}$, $T = \{Ad, No, So, Cp\}$, $\alpha(Ad) = \alpha(No) = \alpha(So) = 0$, $\alpha(Cp) = 1$, $\beta(Ad) = \infty$, $\beta(No) = 8$, $\beta(So) = 3$, $\beta(Cp) = 4$. We also have $^{\circ}(So) = p_1$ and $^{c}(So) = p_3$, $(So)^{\bullet} = p_4$ (we let the reader infer $^{\bullet}()$ and $()^{\bullet}$ for other transitions). For conciseness, we will denote by $\{p_i, p_j, ..\}.\{p_k, p_r, ...\}$ a marking where standard places $p_i, p_j, ...$ and control places $p_k, p_r, ...$ are marked. The net starts in an initial configuration $(M_0.N_0, v_0)$ where $M_0.N_0 = \{p_0\}.\{\}$, $v_0(Ad) = 0$ and $v_0(t) = \perp$ for all other transitions in T.

A possible run of the net of Figure 2-a) is :

$$(\{p_0\}.\{\}, v_0) \xrightarrow{(d_0,Ad)} (\{p_1, p_2\}.\{\}, v_1) \xrightarrow{(2,Cp)} (\{p_1\}.\{p_3\}, v_2) \xrightarrow{(1,No)} (\{p_5\}.\{p_3\}, v_3)$$

where d_0 is any arbitrary positive real value $v_1(ad) = \perp$ and $v_1(No) = v_1(So) = v_1(Cp) = 0$, $v_2(ad) = v_2(Cp) = \perp$ and $v_2(No) = v_2(So) = 2$. This run represents a situation where a advertisement is sent at some date d, a coupon 2 days later, and the customer decides to take the normal offer. Another possible run is :

$$(\{p_0\}.\{\}, v_0) \xrightarrow{(d_0,Ad)} (\{p_1, p_2\}.\{\}, v_1) \xrightarrow{(3,Cp)} (\{p_1\}.\{p_3\}, v_2') \xrightarrow{(1,So)} (\{p_4\}.\{\}, v_3')$$

with $v_2'(ad) = v_2'(Cp) = \perp$ and $v_2'(No) = v_2'(So) = 3$. This run represents a scenario where a customer decides to use the coupon sent. Notice that from $(\{p_1\}.\{p_3\}, v_2')$, time cannot elapse, because transition So is fully enabled, $v_2'(So) = 3$, and $\beta(So) = 3$. Notice also that firing of So can only occur after firing of Cp, but yet time measurement starts for So as soon as $^{\circ}(So) = \{p_1\}$ is filled, i.e. immediately after firing of Ad. This example is rather simple: the net is acyclic, and each transition is enabled/disabled only once. One can rapidly see that the only markings reachable are $M_0.N_0 = \{p_0\}.\{\}$, $M_1.N_0 = \{p_1, p_2\}.\{\}$, $M_2.N_2 = \{p_1\}.\{p_3\}$, $M_3.N_3 = \{p_5\}.\{p_3\}$ and $M_3'.N_3' = \{p_4\}.\{\}$ visited in the runs above, and two additional markings $M_4.N_4 = \{p_2, p_5\}.\{\}$ (reached when transition No is fired before transition Cp and $M_5.N_5 = \{p_5\}.\{p_3\}$ reachable immediately from $M_4.N_4$ by firing Cp. A normal order can be sent at most 8 time units after advertising, a special order must be sent at most 3 time units after advertising if a coupon was received, etc.

4. Reachability

In a configuration $(M.N, v)$ of a waiting net $\mathcal{W}$, v assigns real values to clocks. The timed transition system giving the semantics of a waiting net is hence in general infinite, even when $\mathcal{W}$ is bounded. In timed automata, clock valuations can be partitioned into equivalence classes called regions [23] to get a finite abstraction. For TPNs, these equivalence classes are called *domains*, and can be used to define an untimed transition systems called a *state class graph* [6], that recognizes firable sequences of transitions. In this section, we build similar graphs for waiting nets. We also prove that the set of domains in these graphs is always finite, and use this result to show that reachability and coverability are decidable for bounded waiting nets. The type of domain built for waiting nets is identical to that of TPNs, and the construction of successors is based on the same operations, namely verification of the emptiness of domains, variable substitution, variable elimination, and calculus of a normal form. The main change is that one has to consider time elapsed when computing a successor domain to take into account waiting transitions that have reached their upper bound. A consequence is that a domain can have several successors via the same transition firing. Yet, verification of coverability and reachability remain similar to that of TPNs, i.e. yield an exponential time complexity w.r.t. the number of transitions.

Let t_i be a fully enabled transition with $\alpha(t_i) = 3$ and $\beta(t_i) = 12$, and assume that t_i has been enabled for 1.6 time units. According to the semantics of WPNs, $v(t_i) = 1.6$, and t_i cannot fire yet, as $x_{t_i} < \alpha(t_i)$. Transition t_i can fire only after a certain duration θ_i such that $1.4 \leq \theta_i \leq 10.4$. Similar constraints hold for all enabled transitions. We will show later that they are not only upper and lower bounds on θ_i's of the form $a_i \leq \theta_i \leq b_i$, but can include dependencies of the form $\theta_i - \theta_j \leq c_{ij}$ on pairs of variables. These constraints form a *domain* of legal real values defining the time that can elapse before firing an enabled transition or reaching the upper bound of the interval attached to it.

Definition 4.1. (State Class, Domain)
A *state class* of a waiting net $\mathcal{W}$ is a pair $(M.N, D)$, where $M.N$ is a marking of $\mathcal{W}$ and D is a set of inequalities called *firing domain*. The inequalities in D are of two types:

$$\begin{cases} a_i \leq \theta_i \leq b_i, & \text{where } a_i, b_i \in \mathbb{Q}^+ \text{ and } t_i \in \mathsf{Enabled}(M) \\ \theta_j - \theta_k \leq c_{jk}. & \text{where } c_{jk} \in \mathbb{Q} \text{ and } t_j, t_k \in \mathsf{Enabled}(M). \end{cases}$$

A variable θ_i in a firing domain D over variables $\theta_1, \dots, \theta_m$ represents the time that can elapse before firing transition t_i if t_i is fully enabled, and the time that can elapse before the clock attached to t_i reaches the upper bound $\beta(t_i)$ if t_i is waiting. Hence, if a transition is fully enabled, and $a_i \leq \theta_i \leq b_i$, then t_i cannot fire before a_i time units, and cannot let more than b_i time units elapse, because it becomes urgent and has to fire or be disabled before b_i time units. Inequalities in D define an infinite set of possible values for $\theta_i's$. We denote by $[\![D]\!]$ the set of solutions for a firing domain D. Domains give upper and lower bounds on $\theta_i's$ with inequalities of the form $a_i \leq \theta_i \leq b_i$, but also contain diagonal constraint of the form $\theta'_j - \theta'_k \leq c_{jk}$. These constraints appear when two transitions t_i, t_j are enabled in a state class, and one has to add inequalities of the form $\theta_i \leq \theta_j$ to check firability of t_i. We will say that t_i is *firable* from $(M.N, D)$ if there exists $v \in [\![D]\!]$ such that $(M.N, v) \xrightarrow{t_i} (M'.N', v')$. We will say that $(M'.N', D')$ is a *successor* of $(M.N, D)$ via transition t_i iff t_i is firable from $(M.N, D)$ and D' is a domain describing the set of values for $\theta'_k s$ that are reachable when elapsing time and firing t_i from a valuation in D. The initial state class for a waiting net is the state class $(M_0.N_0, D_0)$ where $M_0.N_0$ is the usual initial marking of a waiting net, and D_0 is the domain

$$D_0 = \{\alpha(t_i) \leq \theta_i \leq \beta(t_i) \mid t_i \in \mathsf{enabled}(M_0)\}$$

The main reason to work with domains is to abstract time and bring back decidability issues for reachability or coverability on an infinite timed transition system to a similar problem on a finite untimed structure. This calls for three properties of the abstraction: first, one has to be able to compute effectively a successor relation $\mathsf{Succ}()$ among state classes. Second, the closure $\mathsf{Succ}^*(\mathsf{M_0}.\mathsf{N_0}, \mathsf{D_0})$ of the initial state class through this successor relation has to be finite. Last, the construction has to be sound and complete. These three properties hold for time Petri nets [26, 6]. In the rest of this paper, we will show that a sound, complete and effective construction of state class graphs for waiting nets exists, but that it slightly differs from existing constructions defined for TPNs.

The way to define a set of solutions $[\![D]\!]$ is not unique. We will say that D_1, D_2 are *equivalent*, denoted $D_1 \equiv D_2$ iff $[\![D_1]\!] = [\![D_2]\!]$. A set of solutions $[\![D]\!]$ is hence not uniquely defined, but fortunately, a unique representation called a *canonical form* exists [27].

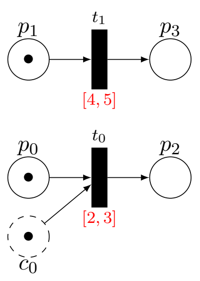
Definition 4.2. (Canonical Form)
The canonical form of a firing domain D is the unique domain

$$D^* = \begin{cases} a_i^* \leq \theta_i \leq b_i^* \\ \theta_j - \theta_k \leq c_{jk}^* \end{cases}$$

where $a_i^* = Inf(\theta_i)$, $b_i^* = Sup(\theta_i)$, and $c_{jk}^* = Sup(\theta_j - \theta_k)$

The canonical form D^* is the minimal set of constraints defining the set of solutions $[\![D]\!]$. If two sets of constraints are equivalent then they have the same canonical form. The constraints we consider are of the form $a_i \leq \theta_i \leq b_i$ and $\theta_i - \theta_j \leq c_{ij}$, where a_i, b_i, c_{ij} are rational values. This type of constraint can be easily encoded by *Difference Bound Matrices* [22], or by constraint graphs (i.e., graphs where vertices represent variables or the value 0, and an edge from θ_i to θ_j of weight $w_{x,y}$ represents the fact that $\theta_i - \theta_j \leq w_{x,y}$). The canonical form for a domain is obtained as a closure operation, that amounts to computing the shortest paths for each pair θ_i, θ_j in the constraint graph. This can be achieved using the Floyd-Warshall algorithm, in $O(n^3)$, where n is the number of variables in the domain. We refer interested readers to [28] for a survey on operations on constraints, specified using DBMs. Note that the set of constraints in a domain D may not be satisfiable, i.e. $[\![D]\!]$ can be empty. In particular, for TPNs and waiting nets, unsatisfiable domains may appear when checking whether a particular transition can be the first one to fire in a domain D. Satisfiability of a domain is decidable, and amounts to checking existence of negative cycles in the constraint graph of D, wich can again be achieved in polynomial time with Floyd-Warshall.

Consider for instance the example of Figure 4. This simple waiting net has two enabled transitions in its initial marking $M_0.N_0 = \{p_0, p_1\}.\{c_0\}$, and its initial domain is the pair of inequalities D_0 represented in the Figure. These constraint can alternatively be represented by a DBM (also shown in the Figure), or a constraint graph (restricted to plain edges) represented on the left of the Figure. Clearly, D_0 has a solution. For instance $\theta_0 = 2.2, \theta_1 = 4.5$ is a solution for the system of inequalities in D_0. Let us test if transition t_1 can fire before t_0. This amounts to adding the constraint $\theta_1 \leq \theta_0$ to domain D_0, or equivalently the dashed edge with weight 0 in the constraint graph. As expected, $D_0 \cup \{\theta_1 \leq \theta_0\}$ is not satisfiable, and one can verify that adding the dashed edge creates a negative cycle in the constraint graph.



$$D_0 = \begin{cases} 2 \leq \theta_0 \leq 3 \\ 4 \leq \theta_1 \leq 5 \end{cases}$$

	0	θ_0	θ_1
0	$(\leq, 0)$	$(\leq, -2)$	$(\leq, -4)$
θ_0	$(\leq, 3)$	$(\leq, 0)$	$(\leq, \infty)$
θ_1	$(\leq, 5)$	$(\leq, \infty)$	$(\leq, 0)$

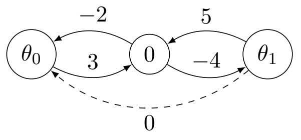


Figure 4. A Waiting net, its initial domain, an equivalent DBM representation, and its constraint graph.

Syntactically, state classes, domains and their canonical forms are identical for TPNs and waiting nets. Indeed, the fact that a transition is waiting or fully enabled does not affect the representation of constraints, as each θ_i represents a time remaining until the upper bound $\beta(t_i)$ is reached by clock x_{t_i}. However, there is a major difference in the construction of successors for state classes of TPNs and those of waiting nets. In TPNs, the maximal duration that can elapse in a state class is constrained by *all* enabled transitions. For waiting nets, the maximal duration that can elapse in a state class is constrained by fully enabled transitions only: elapsing δ time units before firing some transition t_i can be allowed even if a waiting transition t_j reaches its upper bound after $\delta' < \delta$ time units, because variable θ_j represents a *time to upper bound of interval* $[\alpha(t_j), \beta(t_j)]$ rather than a *time to fire*. On the other hand, when elapsing time, one has to consider which clocks attached to waiting transitions have reached their upper bound. A consequence is that a state class has several successors via the same transition, and hence state class graphs of waiting nets are not deterministic.

In state classes of TPNs, verifying that t_i can fire from $(M.N, D)$ is checked by verifying that $D^{+\theta_i} ::= D \cup \{\theta_i \leq \theta_j \mid t_j \in \mathsf{Enabled}(M.N)\}$ is satisfiable. Intuitively, satisfiability of $D^{+\theta_i}$ means that it is possible to fire t_i witout violating urgency of other transitions. Following the semantics of section 3, a transition t_i can fire from a configuration $(M.N, v)$ iff it is fully enabled and $v(t_i) \in [\alpha(t_i), \beta(t_i)]$. Hence, from configuration $(M.N, v)$, firing of t_i is one of the next discrete moves iff there exists a duration θ_i such that t_i can fire from $(M.N, v + \theta_i)$, i.e., after letting duration θ_i elapse, and no other transition becomes urgent before θ_i time units. One can check firability of a transition t_i as in TPNs through satisfiability of $D^{+\theta_i}$, but considering only fully enabled transitions. Note however that if there is a waiting transition t_j, D contains a pair of constraints of the form $\theta_j \leq b_j$, and $\theta_i - \theta_j \leq 0$, then the constraint on θ_j may over-constrain the possible values for θ_i. To avoid any conflict between fully enabled transitions and waiting transitions, we define a notion of *projection* of a domain on its fully enabled transitions, that erases all upper bounds of waiting transitions, and diagonal constraints involving variables associated with waiting transitions.

Definition 4.3. (Projection)
Let D be a firing domain with variables a_i, b_i, c_{jk} set as in def. 4.1. The *projection* of D on its fully enabled transitions is a domain

$$D_{|full} = \begin{array}{ll} & \{a_i \leq \theta_i \leq b_i \mid t_i \in \mathsf{FullyEnabled}(M.N)\} \\ \cup & \{a_i \leq \theta_i \leq \infty, \mid t_i \in \mathsf{Waiting}(M.N)\} \\ \cup & \{\theta_j - \theta_k \leq c_{jk} \in D \mid t_j, t_k \in \mathsf{FullyEnabled}(M.N)\}. \end{array}$$

In waiting nets, t_i is *firable* from a state class $(M.N, D)$ iff $M.N \geq {}^{\bullet}(t_i)$ and $D_{|full} \cup \{\theta_i \leq \theta_j \mid t_j \in \mathsf{FullyEnabled}(M.N)\}$ is satisfiable, i.e. there exists a value for θ_i and for every θ_j attached to a fully enabled transition t_j that does not violate the constraint that θ_j is greater than or equal to θ_i and does not exceed the upper bound b_j appearing in D.

The construction of the set of reachable state classes of a waiting net is an inductive procedure. Originally, a waiting net starts in a configuration $(M_0.N_0, v_0)$, so the initial state class of our system is (M_0, D_0), where $D_0 = \{\alpha(t_i) \leq \theta_i \leq \beta(t_i) \mid t_i \in \mathsf{Enabled}(M_0.N_0)\}$. Then, for every state class $(M.N, D)$, and every transition t firable from $(M.N, D)$, we compute all possible successors $(M'.N', D')$ reachable after firing of t. Note that we only need to consider $t \in \mathsf{FullyEnabled}(M.N)$, as t can fire only when $N > {}^{c}(t)$. Computing $M'.N'$ follows the usual firing rule of a Petri net: $M'.N' = M.N - {}^{\bullet}(t) + (t)^{\bullet}$ and we can hence also compute $\uparrow \mathsf{enabled}(M.N, t)$, $\mathsf{enabled}(M'.N')$ and $\mathsf{FullyEnabled}(M'.N')$. It remains to show the effect of transitions firing on domains to compute all possible successors of a class. Firing a transition t from $(M.N, D)$ propagates constraints of the firing domain D on variables attached to transitions that remain enabled. Variables associated to newly enabled transitions only have to meet lower and upper bounds on their firing times. We can now show that for waiting nets, the set of successors of a state class is finite and can be effectively computed despite waiting transitions and non-determinism.

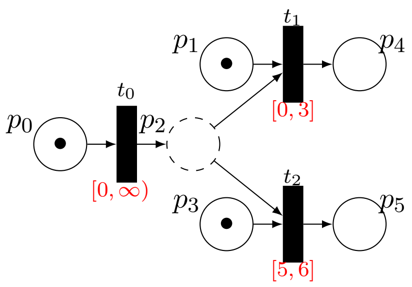


Figure 5. Upper bounds of transitions influence state classes of waiting Nets

Let us illustrate the influence of control places on the construction of state classes with the waiting net of Figure 5. This net starts in a configuration $C_0 = (M_0.N_0, v_0)$ with $M_0.N_0 = \{p_0, p_1, p_3\}.\{\}$. Transitions t_0, t_1, t_2 are enabled in this marking, but t_0 is the only fully enabled transition. From configuration C_0, one can let an arbitrary amount of time $\delta \in \mathbb{R}^{\geq 0}$ elapse, and then fire transition t_0. If t_0 fires from $C_0' = (M_0.N_0, v_0 + \delta)$, the net reaches a new configuration $(M'.N', v')$ where $M'.N' = \{p_1, p_3\}.\{p_2\}$, and $v'(t_0) = \perp, v'(t_1) = v'(t_2) = \delta$.

Let $C_1 = (M'.N', v_1)$ be a configuration reached when $0 \leq \delta < 3$. The value of clock x_{t_1} is still smaller than the upper bound $\beta(t_1) = 3$. Then, from C_1, one can still wait before firing t_1, i.e., t_1 is not urgent and can fire immediately or within a duration $3 - \delta$, and transition t_2 is not firable, and cannot become firable before the urgent firing of t_1. Now, let $C_2 = (M'.N', v_2)$ be a configuration reached after elapsing δ with $3 \leq \delta < 5$. In C_2, we have $v_2(t_2) < 5$ and $v_2(t_1) = 3$, as the value of clock x_{t_1} was frozen when reaching the upper bound $\beta(t_3) = 3$. Hence, in C_2, transition t_1 is urgent and must fire, and transition t_2 still has to wait before firing. Hence, choosing $3 \leq \delta < 5$ forces to fire t_1 immediately after t_0. As this firing consumes the token in control place p_2, it will prevent t_2 from firing. In a similar way, let us call $C_3 = (M'.N', v_3)$ a configuration reached after elapsing $\delta \geq 5$ time units and then firing t_0. In this configuration, we have $v_3(t_1) = 3$ and $v_3(t_2) \in [5, 6]$, forcing t_1 or t_2 to fire immediately without elapsing time. This example shows that the time elapsed in a configuration has to be considered when computing successors of a state class. We have to consider whether the upper bound of a waiting transition has been reached or not, and hence to differentiate several cases when firing a single transition t. Fortunately, these cases are finite, and depend only on upper bounds attached to waiting transitions in domain D.

Definition 4.4. (Upper Bounds Ordering)
Let $M.N$ be a marking, D be a firing domain with constraints of the form $a_i \leq \theta_i \leq b_i$. Let $B_{M.N,D} = \{b_i \mid t_i \in \mathsf{enabled}(M)\}$. We can order bounds in $B_{M.N,D}$, and define bnd_i as the i^{th} bound in $B_{M.N,D}$. We also define $bnd_0 = 0$ and $bnd_{|B_{M.N,D}|+1} = \infty$.

In the example of Figure 5, the initial domain D_0 is

$$D_0 = \begin{cases} 0 \leq \theta_0 \leq \infty \\ 0 \leq \theta_1 \leq 3 \\ 5 \leq \theta_2 \leq 6 \end{cases}$$

The bounds appearing in D_0 are hence $0, 3, 6, \infty$. Consider a transition t_f firable from $C = (M.N, D)$. This means that there is a way to choose a delay θ_f that does not violate urgency of all other transitions. We use $B_{M.N,D}$ to partition the set of possible values for delay θ_f in a finite set of intervals, and find which transitions reach their upper bound when θ_f belongs to an interval. Recall that t_f can fire only if adding constraint $\theta_f \leq \theta_j$ for every fully enabled transition t_j to domain D still yields a satisfiable set of constraints. This means that when considering that t_f fires after a delay θ_f such that $bnd_i \leq \theta_f \leq bnd_{i+1}$, as D also contains a constraint of the form $a_f \leq \theta_f \leq b_f$, considering an interval such that bnd_i is greater than $\min\{b_j \in B_{M.N,D} \mid t_j \in \mathsf{FullyEnabled}(M.N)\}$ or smaller than a_f immediately leads to inconsistency of constraint $D_{|full} \cup \bigwedge_{t_j \in \mathsf{FullEnabled}(M.N)} \theta_f \leq \theta_j \wedge bnd_i \leq \theta_f \leq bnd_{i+1}$. We denote by $B^{t_f}_{M.N,D}$ the set of bounds $B_{M.N,D}$ pruned out from these inconsistent bound values. Now, choosing a particular interval $[bnd_i, bnd_{i+1}]$ for the possible values in θ_f indicates for which waiting transitions $t_1, \dots t_k$ the clocks $x_{t_1}, \dots x_{t_k}$ measuring time elapsed since enabling has reached upper bounds $\beta(t_1), \dots \beta(t_k)$. The values of these clocks become irrelevant, and hence the corresponding θ_i's have to be eliminated from the domains.

Definition 4.5. (Time progress (to the next bound))
Let $M.N$ be a marking, D be a firing domain, and $b = \min B_{M.N,D}$ be the smallest upper bound for enabled transitions. The domain reached after progressing time to bound b is the domain D' obtained by:

- replacing every variable θ_i by expression $\theta'_i - b$
- eliminating every θ'_k whose upper bound is b,
- computing the normal form for the result and renaming all θ'_i to θ_i

Progressing time to the next upper bound allows to remove variables related to waiting transitions whose clocks have reached their upper bounds from a firing domain. We call these transitions *timed-out transitions*. For a transition $t_k \in waiting(M.N)$ if $v(x_{t_k}) = \beta(t_k)$, variable θ_k, that represents the time needed to reach the upper bound of the interval is not meaningful any more: either t_k gets disabled in the future, or is fired urgently as soon as it becomes fully enabled, because $\theta_k = 0$. So the only information to remember is that t_k will be urgent as soon as it becomes fully enabled.

Definition 4.6. (Successors)
A *successor* of a class $C = (M.N, D)$ after firing of a transition t_f is a class $C' = (M'.N', D')$ such that $M'.N'$ is the marking obtained after firing t_f from $M.N$, and D' is a firing domain reached after firing t_f in some interval $[b_r, b_{r+1}]$ where b_r, b_{r+1} are consecutive values in $B^{t_f}_{M.N,D}$.

Given C and a firable transition t_f, we can compute the set $\mathsf{Post}(C, t_f)$ of successors of C, i.e. $\mathsf{Post}(C, t_f) := \{(M'.N', \mathsf{next}_r(D, t_f)) \mid b_r \in B^{t_f}_{M.N,D} \cup \{0\}\}$. The next marking is the same for every successor and is $M'.N' = M.N - {}^\bullet t_f + t_f^\bullet$. We then compute $\mathsf{next}_r(D, t_f)$ as follows:

1) Time progress: We successively progress time from D to bounds $b_1 < b_2 < \cdots < b_r$ to eliminate variables of all enabled transitions reaching their upper bounds, up to bound b_r. We call D^r the domain obtained this way. Every transition t_k in $Enabled(M.N)$ that has no variable θ_k in D^r is hence a waiting transition whose upper bound has been reached.

2) Firing condition: We add to D^r the following constraints: we add the inequality $(b_r \leq \theta_f \leq b_{r+1})$, and for every transition $t_j \in \mathsf{FullyEnabled}(M) \setminus \{t_f\}$, we add to D^r the inequality $\theta_f \leq \theta_j$. This means that no other transition was urgent when t_f has been fired. Let D^u be the new firing domain obtained this way. If any fully enabled transition t_j has to fire before t_f, then we have a constraint of the form $a_j \leq \theta_j \leq b_j$ with $b_j < a_f$, and D^u is not satisfiable. As we know that t_f is firable, this cannot be the case, and D^u has a solution, but yet, we have to include in the computation of the next firing domains reached after firing of t_f the constraints on θ_f due to urgency of other transitions.

3) Substitution of variables: As t_f fires after elapsing θ_f time units, the time to fire of other transitions whose clocks did not yet exceed their upper bounds decreases by the same amount of time. Variables of timed-out transitions have already been eliminated in D^u. So for every $t_j \neq t_f$ that has an associated constraint $a_j \leq \theta_j \leq b_j$ we do a variable substitution reflecting the fact that the new time to fire θ'_j decreases w.r.t. the former time to fire θ_j. We set $\theta_j := \theta_f + \theta'_j$. When this is done, we obtain a domain D'^{u,b_r} over a set of variables $\theta'_{i_1}, \ldots \theta'_{i_k}$, reflecting constraints on the possible remaining times to upper bounds of all enabled transitions that did not timeout yet.

4) Variable Elimination: As t_f fired at time θ_f, it introduced new relationships between remaining firing times of other transitions, i.e other $\theta'_i \neq \theta_f$, that must be preserved in the next state class. However, as t_f is fired and changed the marking of the waiting net, in the next class, either t_f is a newly enabled transition, or it is not enabled. In the first case, the new domain will contain a fresh variable attached to the new occurrence of t_f enabled by the firing. In the second case, variable θ_f

should not appear in the new domain. We hence need to remove the "old" variable θ_f from inequalities, while preserving an equivalent set of constraints, before introduction of constraints associated with newly enabled transitions. This is achieved by elimination of variable θ_f from D'^{u,b_r}, for instance with the well known Fourier-Motzkin technique (see Appendix A for details). We proceed similarly with variable θ'_i for every transition t_i that is enabled in marking $M.N$ but not in $M.N - {}^\bullet(t_f)$. After elimination, we obtain a domain D'^{E,b_r} over remaining variables.

5) Addition of new constraints : The last step to compute the next state classes is to introduce fresh constraints for firing times of newly enabled transitions. For every $t_i \in \uparrow \mathsf{enabled}(M.N, t_f)$ we add to D'^{E,b_r} the constraint $\alpha(t_i) \leq \theta'_i \leq \beta(t_i)$. For every timed-out transition t_k that becomes fully enabled, we add to D'^{E,b_r} the constraint $\theta_k = 0$. Timed-out transitions that become fully enabled are hence urgent in the next class. After adding all constraints associated to newly enabled transitions, we obtain a domain, in which we can rename every θ'_i to θ_i to get a domain D'^{F,b_r}. Notice that this domain needs not be minimal, so we do a last normalization step (see Definition 4.2) to obtain a final canonical domain $\mathsf{next}_r(D, t_f) = D'^{F,b_r\,*}$.

Let us compare the state class construction in TPNs and in waiting nets. In TPNs (see [6, 7]), for a domain D and a transition t_f, there is a single successor domain in $\mathsf{Post}(D, t_f)$, and hence a state class can have up to $|T|$ successors. In waiting nets, for a given firable transition t_f, $\mathsf{Post}(D, t_f)$ contains one domain per bound in $B^{t_f}_{M.N,D}$, i.e. $\mathsf{Post}(D, t_f)$ is not a singleton anymore. Now, if a waiting net has no control place, there is no bound on waiting transitions to consider, as enabled transitions are fully enabled in every configuration. Step 1 of successor construction leaves the starting domain D unchanged, and consequently the state class construction shown above is exactly the standard construction for TPNs. The number of successors for a state class in TPNs is exactly the number of enabled transitions. Let $\mathsf{Post}(C)$ be the set of successors of a class C. Then $|\mathsf{Post}(C)| \leq |\mathsf{enabled}(M.N)|^2$, as the number of bounds to consider for a transition cannot exceed the number of enabled transitions. Computing successors can be repeated from each class in $Post(C)$. For a given net $\mathcal{W}$, and a given marking $M_0.N_0$, we denote by $\mathcal{C}(W)$ the set of classes that can be built inductively. This set need not be finite, but we show next that this is due to markings, and that the set of domains appearing in state classes is finite.

Definition 4.7. (State Class Graph) The *State Class Graph* of a waiting net $\mathcal{W}$ is a graph $SCG(\mathcal{W}) = (\mathcal{C}(W), C_0, \longrightarrow)$ where $C_0 = (M_0.N_0, D_0)$, and $C \xrightarrow{t} C'$ iff $C' \in Post(C, t)$.

Let $\rho = (M_0.N_0, v_0) \xrightarrow{d_1} (M_0.N_0, v_0 \oplus d_1) \xrightarrow{t_1} (M_1.N_1, v_1) \dots (M_k.N_k, v_k)$ be a run of $\mathcal{W}$ and $\pi = (M'_0.N'_0, D_0).(M'_1.N'_1, D_1) \dots (M'_k.N'_k, D_k)$ be a path in $SCG(\mathcal{W})$. We will say that ρ and π *coincide* iff $\forall i \in 1..k, M_i.N_i = M'_i.N'_i$, and for every step $(M_i.N_i, v_i) \xrightarrow{d_i} (M_i.N_i, v_i \oplus d_i) \xrightarrow{t_i} (M_{i+1}.N_{i+1}, v_{i+1})$, there exists an interval $[b_r, b_{r+1}]$ such that $d_i \in [b_r, b_{r+1}]$ and $D_{i+1} = \mathsf{next}_r(D_i, t_i)$.

Let us illustrate the construction of state classes with the example of Figure 5. We start from an inital state class $Cl_0 = (M_0.N_0, D_0)$ with $M_0.N_0 = \{p_0, p_1, p_3\}\{\}$ and D_0 is given by the set of constraints $0 \leq \theta_0, 0 \leq \theta_1 \leq 3, 5 \leq \theta_2 \leq 6$. From the initial configuration $(M_0.N_0, v_0)$ one can only

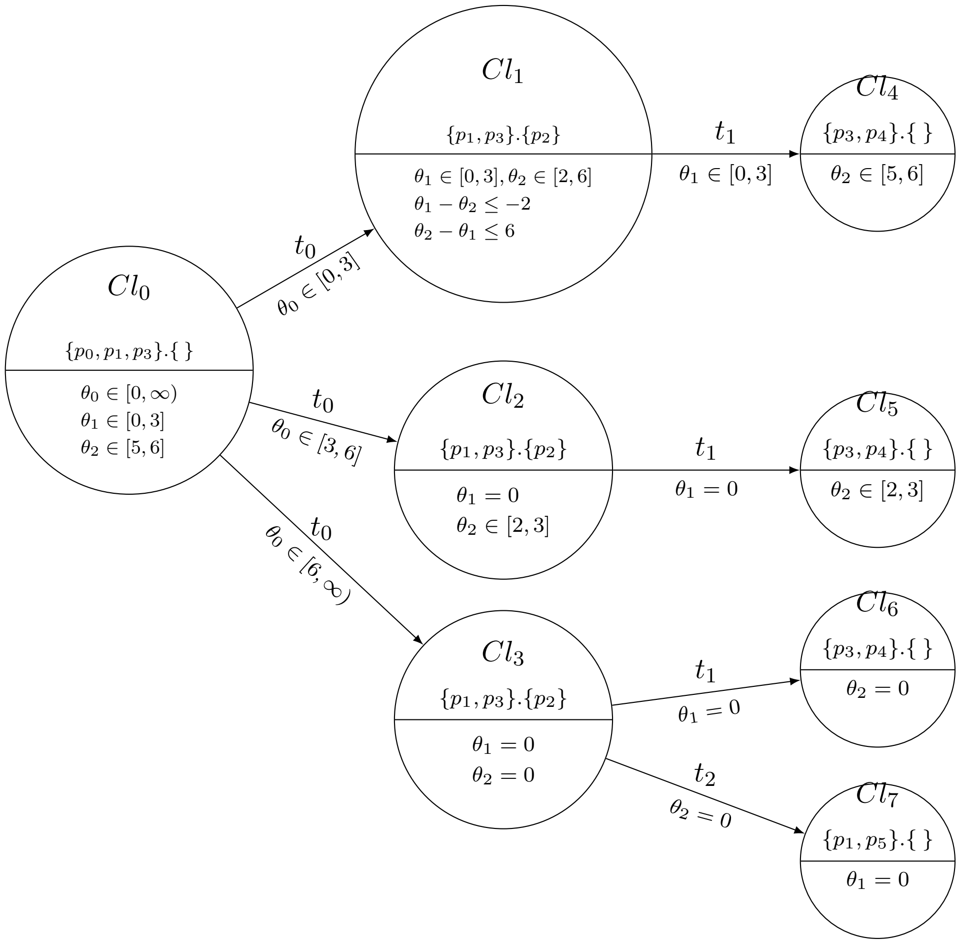


Figure 6. A non-deterministic state class graph for the waiting net of Figure 5.

fire transition t_0, that is, wait for some duration θ_0 and then perform a discrete move with transition t_0. So t_0 is also the only firable transition from state class Cl_0. Here, we have two important bounds, 3 and 6 which are the upper bounds attached to t_1 and t_2 respectively. Let us assume that t_0 fires after a duration in $[0,3]$. We hence fire from a domain that satisfies the following constraints:

$$D'_0 = \begin{cases} 0 \leq \theta_0 \leq 3 \\ 0 \leq \theta_1 \leq 3 \\ 5 \leq \theta_2 \leq 6 \end{cases}$$

One can easily find values for $\theta_0, \theta_1, \theta_2$ satisfying D'_0 and such that $\theta_0 \leq \theta_1$ and $\theta_0 \leq \theta_2$. So t_0 is firable from Cl_0. Let θ'_1, θ'_2 be variables describing the remaining times to fire after firing t_0. As t_0 is the fired transition, all remaining (positive) times are decreased by θ_0, i.e. $\theta'_1 = \theta_1 - \theta_0$ and

$\theta'_2 = \theta_2 - \theta_0$. This gives a variable change allowing to obtain the following domain:

$$\begin{cases} 0 \leq \theta_0 \leq 3 \\ 0 \leq \theta'_1, 0 \leq \theta'_2 \\ 0 \leq \theta'_1 + \theta_0 \leq 3 \\ 5 \leq \theta'_2 + \theta_0 \leq 6 \end{cases} \text{or equivalently} \begin{cases} 0 \leq \theta_0 \leq 3 \\ 0 \leq \theta'_1, 0 \leq \theta'_2 \\ 0 - \theta'_1 \leq \theta_0 \leq 3 - \theta'_1 \\ 5 - \theta'_2 \leq \theta_0 \leq 6 - \theta'_2 \end{cases}$$

The variable to eliminate is now θ_0, we can hence reorganize lines containing θ_0 in this domain as

$$\max(0, 0 - \theta'_1, 5 - \theta'_2) \leq \theta_0 \leq \min(3, 3 - \theta'_1, 6 - \theta'_2)$$

We hence obtain, after elimination of θ_0

$$\begin{cases} 0 \leq 3 & (tautology) & \\ 0 \leq \theta'_1, 0 \leq \theta'_2 & & \\ 0 \leq 3 - \theta'_1 & \iff & \theta'_1 \leq 3 \\ 0 \leq 6 - \theta'_2 & \iff & \theta'_2 \leq 6 \\ 0 - \theta'_1 \leq 3 & \iff & -3 \leq \theta'_1 \\ 0 - \theta'_1 \leq 3 - \theta'_1 & (tautology) & \\ 0 - \theta'_1 \leq 6 - \theta'_2 & & \\ 5 - \theta'_2 \leq 3 & \iff & 2 \leq \theta'_2 \\ 5 - \theta'_2 \leq 3 - \theta'_1 & & \\ 5 - \theta'_2 \leq 6 - \theta'_2 & (tautology) & \end{cases}$$

After elimination of redundancies and tautologies, and renaming θ'_1, θ'_2 into θ_1, θ_2 we obtain a domain D_1 with the following constraints:

$$D_1 = \begin{cases} 0 \leq \theta_1 \leq 3 \\ 2 \leq \theta_2 \leq 6 \\ \theta_2 - \theta_1 \leq 6 \\ \theta_1 - \theta_2 \leq -2 \end{cases}$$

The state class reached by firing t_0 after elapsing 0 to 3 time units is $Cl_1 = (M_1.N_1, D_1)$ with $M_1 = \{p_1, p_3\}$ and $N_1 = \{p_2\}$. One can notice that, as transitions t_1, t_2 remain enabled after firing of t_0, variable elimination adds diagonal constraints of the form $\theta_2 - \theta_1 \leq 6$ and $\theta_1 - \theta_2 \leq -2$. This is the case in general for all pairs of transitions that remain enabled. Domains are hence not limited to interval contraints of the form $a_i \leq \theta_i \leq b_i$. Similarly, starting from Cl_0 one can let between 3 and 6 time units elapse and reach ctate class $Cl_2 = (M_2.N_2, D_2)$. D_2 is easily obtained by progressing D_0 of 3 time units (this is a variable change replacing θ_i by $\theta'_i + 3$), and projecting away constraints containing θ'_1. Then, firing t_0 gives a pair of constraints $0 \leq \theta'_0 \leq 3$ and $2 \leq \theta'_2 + \theta'_0 \leq 3$. We let the reader check that this reduces to $2 \leq \theta_2 \leq 3$ after elimination of θ_0. Then by adding constraints on θ_1 arising from the fact that after firing of t_0, t_1 becomes fully enabled, we obtain :

$$D_2 = \begin{cases} \theta_1 = 0 \\ 2 \leq \theta_2 \leq 3 \end{cases}$$

One can immediately notice that from any configuration in class Cl_2, transition t_1 is urgent, and is the the only transition that can fire. We let the readers build the remaining classes, and give the complete state class graph for the waiting net of figure 5 in Figure 6. One can observe that this state class graph *is not* deterministic.

Proposition 4.8. (Completeness)
For every run $\rho = (M_0.N_0, v_0)\dots(M_k.N_k, v_k)$ of $\mathcal{W}$ there exists a path π of $SCG(\mathcal{W})$ such that ρ and π coincide.

Proof:
We only give a proof sketch, a complete proof of this proposition is provided in Appendix D. We proceed by induction on the length of runs. For the base case, we can easily prove that any transition firing from the initial configuration after some delay d gives a possible solution for D_0 and a successor class, as D_0 does not contain constraints of the form $\theta_i - \theta_j \leq c_{ij}$. The induction step is similar, and slightly more involved, because domains contain constraints involving pairs of variables. However, we can show (Lemma D.2 in Appendix D) that along run ρ for every pair of steps composed of a time elapsing of duration d_i followed by the firing of a transition t_f, we have $d_i \in [a_{i,f}, b_{i,f}]$, where $a_{i,f}$ is the lower and $b_{i,f}$ the upper bound on variable θ_f at step i of the run. Hence, for every run of $\mathcal{W}$ there is a path that visits the same markings and maintains consistent constraints. □

Proposition 4.9. (Soundness)
Let π be a path of $SCG(\mathcal{W})$. Then there exists a run ρ of $\mathcal{W}$ such that ρ and π coincide.

Proof:
As for Proposition 4.8, we can prove this property by induction on the length of paths in $SCG(\mathcal{W})$. A complete proof of this proposition is provided in Appendix D. □

Proposition 4.8 shows by induction on runs length that every marking reached by a run of a waiting net appears in its state class graph. The proof of Proposition 4.9 uses a similar induction on paths length, and shows that we do not introduce new markings. These propositions show that the state class graph is a sound and complete abstraction, even for unbounded nets. We can show a stronger property, which is that the set of domains appearing in a state class graph is finite.

Proposition 4.10. The set of firing domains in $SCG(\mathcal{W})$ is finite.

Proof:
We only give a proof sketch, and provide a complete proof in Appendix D. Domains are of the form $\{a_i \leq \theta_i \leq b_i\}_{t_i \subseteq T} \cup \{\theta_i - \theta_j \leq c_{i,j}\}_{t_i,t_j \subseteq T}$. We can easily adapt proofs of [6] (lemma 3 page 9)

to show that every domain generated during the construction of the SCG has inequalities of the form $a_i \leq \theta_i \leq b_i$ and $\theta_i - \theta_j \leq c_{ij}$, where $0 \leq a_i \leq \alpha(t_i)$, $0 \leq b_i \leq \beta(t_i)$ and $-\alpha(t_i) \leq c_{ij} \leq \beta(t_i)$ and $\alpha_i, \beta(t_i)$ are the constant bounds of interval attached to t_i. This does not yet prove that the set of domains is finite. We then show that domains are *bounded*, i.e. that θ_i's and their differences have rational upper and lower bounds (see Lemma D.3 in Appendix). We then show that domains are also *linear*, i.e. that constants appearing in inequalities are linear combinations of a finite set of constant values. Domain D_0 is bounded and linear, and a series of technical lemmas (given in Appendix D) show that variable elimination, reduction to a canonical form, etc. preserve bounds and linearity (a similar result was shown in [6] for domains of TPNs). The set of bounded linear domains between fixed bounds is finite, so the set of domains of a waiting net is finite. □

Finiteness of the set of domains of waiting nets is an important property, that is not a straightforward consequence of the syntactic proximity with TPNs nor of the effectiveness of successors construction. Indeed, waiting nets allow to stop time measurement for waiting transitions reaching their upper bounds. Petri nets with stopwatches, where time measurement can also be stopped, do not have a finite state class representation, even for bounded nets. The reason is that clock differences in domains of stopwatch Petri nets can take any value. Waiting nets avoid this issue because clocks are always stopped at a predetermined instant (when they reach the upper bound of an interval).

Corollary 4.11. If $\mathcal{W}$ is a bounded waiting net then $SCG(\mathcal{W})$ is finite.

Proof:
States of $SCG(\mathcal{W})$ are of the form $(M.N, D)$ where $M.N$ is a marking and D a domain for times to fire of enabled transitions. By definition of boundedness, the number of markings in $SCG(\mathcal{W})$ is finite. By Prop. 4.10, the set of domains appearing in $SCG(\mathcal{W})$ is finite, so $SCG(\mathcal{W})$ is finite. □

More precisely, if a net is k_M-bounded, there are at most $k_M^{|P|}$ possible markings, and the number of possible domains is bounded by $(2 \cdot K_{\mathcal{W}} + 1)^{|T+1|^2}$, where $K_{\mathcal{W}} = \max_{i,j} \lfloor \frac{\beta_i}{\alpha_j} \rfloor$ is an upper bound on the number of linear combinations of bounds appearing in domains. Hence the size of $SCG(\mathcal{W})$ is in $O(k_M^{|P|} \cdot (2 \cdot K_{\mathcal{W}} + 1)^{|T+1|^2})$. A direct consequence of Proposition 4.8, Proposition 4.9, and Corollary 4.11 is that many properties of bounded waiting nets are decidable.

Corollary 4.12. (Reachability and Coverability)
The reachability and coverability problems for bounded waiting nets are decidable and PSPACE-complete.

Proof:
For PSPACE membership, given a target marking $M_t.N_t$ it suffices to explore non-deterministically all runs starting from $(M_0.N_0, D_0)$ of length at most $|SCG(\mathcal{W})|$ to find marking $M_t.N_t$ and prove reachability, or a marking that covers $M_t.N_t$ witnessing coverability. Random walks of bounded length are feasible in NLOGSPACE w.r.t. the size of the explored graph, whence the NPSPACE=PSPACE complexity. For hardness, we already know that reachability for 1-safe Petri nets is PSPACE-Complete [29], and a (bounded) Petri net is a (bounded) waiting net without control places and with $[0, \infty)$ constraints.

Similarly, given 1-safe Petri net and a place p, deciding if a marking with $M(p) = 1$ (which is a coverability question) is reachable is PSPACE-complete [2]. This question can be recast as a coverability question for waiting nets, thus proving the hardness of coverability. $\square$

Let us illustrate the advantages and expressive power of waiting nets with the example of Figure 7 containing a cyclic behavior. This net contains a control part (in Gray), composed of standard places p_2, p_3, of control places c_3, c_4, and of two transitions tc_3 and tc_4 with respective time intervals $[2, 2]$ and $[1, 1]$. The main idea in this control part is to move a token from place c_3 to place c_4 after 2 time units, and then move the token from c_4 to c_3 after 1 time unit. Control places c_3 and c_4 are used to allow/forbid firing of transitions t_1 and t_2 in the right part of the net. Transitions t_1 and t_2 are independent transitions with respective time intervals $[1, 2]$ and $[2, 3]$ and which firings are respectively allowed when place c_3 and c_4 are filled. Note that allowing or forbidding firings of transitions does not affect their enabledness: while transition t_2 is waiting for a token in c_4 to fire, one measures the time elapsed since it was last enabled (and similarly when transition t_1 is waiting for a token in c_3).

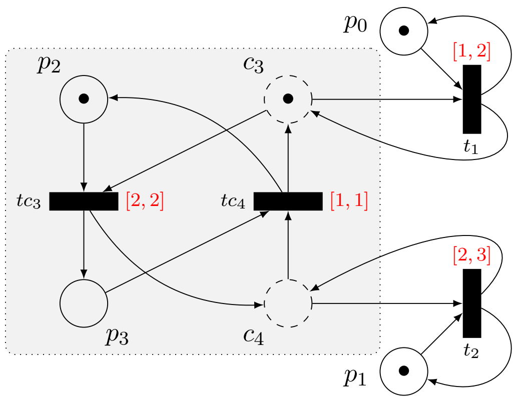


Figure 7. A cyclic Waiting net

The main idea of this example is to give control alternatively to transition t_1 and t_2, but measure enabled time for t_2 when control allows firing of t_1 and enabled time for t_1 when control allows firing of t_2. The timed word $(t_1, 1)(t_1, 2)(tc_3, 3)(t_2, 3.2)$ is a possible behavior of this net. Notice that transition t_2 becomes fully enabled after firing of t_3, but was enabled from the beginning of the run. In a symmetric way, the timed word $(t_1, 1)(t_1, 2)(tc_3, 3)(tc_4, 4)(t_1, 4)(t_1, 5)(t_1, 6)$ is a legal behavior of the model. Indeed, after firing of tc_3 time still elapses for t_1, which can fire immediately after firing of tc_4. These two example behaviors show that one can design controllers that decouple control and time measurement with waiting nets. The state class computed for this example contains 44 state classes, and show that one can fire up to 5 occurrences of t_1 before an occurrence of t_2 fires urgently. Notice that if one replaces control places by standard places in this example to obtain a TPN, then transition t_2 never fires, as it is disabled at most 1 time unit after its enabling by firing of t_4. This example shows that waiting nets can be a tool to design easily fair controllers.

5. Expressiveness

The next natural question for waiting nets is their expressiveness w.r.t. other timed models. There are several ways to compare expressiveness of timed models: One can build on relations between models such as isomorphism of their underlying timed transition systems, timed similarity, or bisimilarity. In the rest of this section, we compare models w.r.t. the timed languages they generate. For two particular types of model $\mathcal{M}_1$ and $\mathcal{M}_2$, we will write $\mathcal{M}_1 \leq_{\mathcal{L}} \mathcal{M}_2$ when, for every model $X_1 \in \mathcal{M}_1$, there exists a model X_2 in $\mathcal{M}_2$ such that $\mathcal{L}(X_1) = \mathcal{L}(X_2)$. Similarly, we will write $\mathcal{M}_1 <_{\mathcal{L}} \mathcal{M}_2$ if $\mathcal{M}_1 \leq_{\mathcal{L}} \mathcal{M}_2$ and there exists a model $X_2 \in \mathcal{M}_2$ such that for every model $X_1 \in \mathcal{M}_1$, $\mathcal{L}(X_2) \neq \mathcal{L}(X_1)$. Lastly, we will say that $\mathcal{M}_1$ and $\mathcal{M}_2$ are equally expressive and write $\mathcal{M}_1 =_{\mathcal{L}} \mathcal{M}_2$ if $\mathcal{M}_1 \leq_{\mathcal{L}} \mathcal{M}_2$ and $\mathcal{M}_1 \leq_{\mathcal{L}} \mathcal{M}_2$. In the rest of this section, we compare bounded and unbounded waiting nets with injective/non-injective labelling, with or without silent transitions labelled by ϵ to timed automata, TPNs, Stopwatch automata, and TPNs with stopwatches.

We first have obvious results. It is worth nothing that every model with non-injective labeling is more expressive than its injective counterpart. Similarly, every unbounded model is strictly more expressive than its bounded subclass. Waiting nets can express any behavior specified with TPNs. Indeed, a WTPN without control place is a TPN. One can also remark that (unbounded) TPNs, and hence WTPNs are not regular. It is also well known that the timed language of a bounded TPN can be encoded by a time bisimilar timed automaton [24, 7, 25]. We show next that one can extend the results of [7] to waiting nets, i.e. reuse the state class construction of section 4 to build a finite timed automaton $\mathcal{A}_{\mathcal{W}}$ that recognizes the same language as a waiting net $\mathcal{W}$. As shown by Proposition 4.8 and Proposition 4.9, the state class graph $SCG(\mathcal{W})$ is sound and complete. State class graphs abstract away the exact values of clocks and only remember constraints on remaining times to fire of fully enabled transitions, and times to upper bounds for enabled transitions. If we label moves of $SCG(\mathcal{W})$ by the label of the transition used to move from a state class to the next one, the state class graph becomes an automaton that recognizes the untimed language of $\mathcal{W}$. Further, one can add guards on clock values to moves and clock invariants to states of a state class graph to recover the timing information lost during abstraction.

Definition 5.1. (Extended State Class)
An *extended state class* is a tuple $C_{ex} = (M.N, D, \chi, trans, XP)$, where $M.N$ is a marking, D a domain, χ is a set of real-valued clocks, $trans \in (2^T)^\chi$ maps clocks to sets of transitions and $XP \subseteq T$ is a set of transitions which upper bound have already been reached.

Extended state classes were already proposed in [7] as a building step for state class timed automata recognizing languages of bounded TPNs. Here, we add information on transitions that have been enabled for a duration that is at least their upper bound. This is needed to enforce urgency when such transitions become firable. In extended state classes, every clock $x \in \chi$ represents the time since enabling of several transitions in $trans(x)$, that were enabled at the same instant. So, for a given transition t, the clock representing the valuation $v(x_t)$ is $trans^{-1}(t)$[1]. Let $\mathbb{C}^{ex}$denote the set of all

[1]An alternative is to associate a clock to every enabled transition, but using a map $trans$ as proposed in [7] allows to build automata with a reduced number of clocks.

state classes. We can now define the state class timed automaton $SCTA(\mathcal{W})$ by adding guards and resets to the transitions of the state class graph, and invariants to state classes.

Definition 5.2. (State Class Timed Automaton)
The state class timed automaton of $\mathcal{W}$ is a tuple $SCTA(\mathcal{W}) = (L, l_0, X, \Sigma, Inv, E, F)$ where:

- $L \subseteq \mathbb{C}^{ex}$ is a set of extended state classes. $l_0 = (M_0.N_0, D_0, \{x_0\}, trans_0, XP_0)$, where $trans_0(x_0) = \mathsf{Enabled}(M_0.N_0)$ and $XP_0 = \emptyset$.
- $F = L$, i.e. every run is accepting,
- $\Sigma = \lambda(T)$, and $X = \bigcup\limits_{(M.N,D,\chi,trans,XP)\in L} \chi \subseteq \{x_1, \dots x_{|T|}\}$ is a set of clocks
- E is a set of transitions of the form $(C_{ex}, \lambda(t), g, R, C'_{ex})$. In each transition, the locations $C_{ex} = (M.N, D, \chi, trans, XP)$ and $C'_{ex} = (M'.N', D', \chi', trans', XP')$ are two extended state classes such that $(M'.N', D') \longrightarrow (M.N, D)$ is a move of the STG with $D' = next_r(D, t)$ for some index r chosen among the upper bounds ordering computed from domain D.

  We can compute the set of transitions disabled by the firing of t from $M.N$, denoted $\mathsf{Disabled}(M.N, t)$ and from there, compute a new set of clocks χ'. We have $\chi' = \chi \setminus \{x \in \chi \mid trans(x) \subseteq \mathsf{Disabled}(M.N, t)\}$ if firing t does not enable new transitions. If new transitions are enabled or become fully enabled after reaching their upper bound, we have $\chi' = \chi \setminus \{x \in \chi \mid trans(x) \subseteq \mathsf{Disabled}(M.N, t)\} \cup \{x_i\}$, where i is the smallest index for a clock in X that is not used. Intuitively, x_i will memorize the time elapsed since the firing of t, and hence the time elapsed since new enabling of all transitions enabled by this firing instead of memorizing the same value for every transition.

  Similarly, we can set

$$trans'(x_k) = \begin{cases} \uparrow \mathsf{enabled}(M.N, t) \cup \{t \in XP \cap \mathsf{FullyEnabled}(M'.N')\} \text{ if } x_k = x_i \\ trans(x_k) \setminus \mathsf{Disabled}(M.N, t) \text{ if } x_k \neq x_i \text{ and } trans(x_k) \nsubseteq \mathsf{Disabled}(M.N, t) \\ \text{Undefined otherwise} \end{cases}$$

  The set XP lists the waiting transitions that have reached their upper bounds and were not fully enabled.

$$\begin{aligned} XP' = XP \quad &\cap \ \mathsf{Enabled}(M - {}^{\bullet}(t)) \setminus \mathsf{FullyEnabled}(M'.N') \\ &\cup \ \{t_k \in \mathsf{Enabled}(M'.N') \mid \theta_k \notin D'\} \end{aligned}$$

  In words, XP' contains the transitions already in XP that are still enabled, and did not receive a token in the control places of their preset, plus enabled waiting transitions that reached their upper bound, and for which no constraint is kept in domain D'.

  The set of clocks reset in a transition is $R = \{x_i\}$, where x_i is the first unused clock, if some transition is newly enabled, or was in XP and becomes fully enabled. $R = \emptyset$ otherwise.

  The guard g is set to $\alpha(t) \leq trans^{-1}(t)$ if there is no constraint of the form $trans^{-1}(t) \leq 0$ in the invariant of C_{ex}, that is if t is not a transition that became urgent after timing out, and g is set to true otherwise.

- Let us now consider invariants attached to states. Let $\mathsf{Urgent}(C_{ex})$ denote the set of fully enabled transitions of $M.N$ that are urgent. One can compute this set by looking at fully enabled transitions with a constraint of the form $\theta_k = 0$ in domain D. We distinguish two cases. The first one is when no transition is urgent, i.e., $\mathsf{Urgent}(C_{ex}) = \emptyset$. In this situation, we can let some time elapse in location C_{ex} until the upper bound for some clock attached to a transition is reached. The invariant $Inv(C_{ex})$ is hence:

$$Inv(C_{ex}) = \bigwedge_{\substack{x_j \in trans^{-1}(\mathsf{FullyEnabled}(M'.N')),\\ t_k \in trans(x_j) \cap \mathsf{FullyEnabled}(M'.N')}} x_j \leq \beta(t_k)$$

If $\mathsf{Urgent}(C'_{ex}) \neq \emptyset$ the invariant $Inv(C_{ex})$ is set to:

$$Inv(C_{ex}) = \bigwedge_{x_k \in trans^{-1}\left(\mathsf{Urgent}(C_{ex})\right)} x_k \leq 0$$

The invariant in location C_{ex} then guarantees that if t is urgent in C_{ex}, then it is fired immediately without elapsing time, or another urgent transition is fired immediately.

Proposition 5.3. Let $\mathcal{W}$ be a waiting net. Then $\mathcal{L}(SCTA(\mathcal{W})) = \mathcal{L}(\mathcal{W})$.

Proof:
Obviously, every sequence of transitions in $\mathcal{L}(SCTA(\mathcal{W}))$ is a sequence of transitions of the STG, and hence there exists a timed word that corresponds to this sequence of transitions. Furthermore, in this sequence, every urgent transition is fired in priority before elapsing time, and the delay between enabling and firing of a transition t lays between the upper and lower bound of the time interval $[\alpha_t, \beta_t]$ if some time elapses in a state before the firing of t, and at least β_t time units if t fires immediately after reaching some state in the sequence (it is an urgent transition, so the upper bound of its interval has been reached, possibly some time before full enabling). Hence, every timed word of $SCTA(\mathcal{W})$ is also a timed word of $\mathcal{W}$. We can reuse the technique of Prop. 4.8 and prove by induction on the length of runs of $\mathcal{W}$ that for every run of $\mathcal{W}$, there exists a run of $SCTA(\mathcal{W})$ with the same sequence of delays and transitions. Let us first consider a run of size 1, that is of the form $(M_0.N_0, v_0) \xrightarrow{d_0,t_0} (M_1.N_1, v_1)$. The domain D_0 is a conjunction of constraints of the form $\alpha_i \leq \theta_i \leq b_i$ for every enabled transition, and firability of t_0 after a delay d_0 is checked on the projection of D_0 on transitions that are not timed-out in the bound interval $[b_r, b_{r+1}]$ that contains d_0, i.e. on a subset of the original set of constraint D_0. We have associated to location l_0 the invariant $Inv(l_0) = x_0 \leq \min_{t_k \in \mathsf{FullyEnabled}(M_0.N_0)}(\beta_k)$ (there is a single clock x_0 as all enabled transitions are enabled at date 0). All transition labeled by t_0 leaving l_0 are attached a guard of the form $g = \alpha(t_0) \leq x_0$. The move of $\mathcal{W}$ via transition t_0 after d_0 time units is allowed, so we necessarily have $v(x_{t_0}) = d_0 \geq b_r \geq \alpha(t_0)$. As we have $b_r \leq \beta(t_0)$, and as t_0 is firable from D_0, we can also claim that $v_0 + d_0 \models Inv(l_0)$ and $v_0 + d_0 \models g$. Hence (d_0, t_0) is a timed word of $\mathcal{L}(SCTA(\mathcal{W}))$.

Let us now consider a sequence of moves of the form $(M_0.N_0, v_0) \xrightarrow{d_0,t_0} \dots \xrightarrow{d_{k-1},t_{k-1}} (M_k.N_k, v_k)$. As the state class graph is complete, there exists a sequence of abstract moves $(M_0.N_0, D_0) \xrightarrow{t_0}$

$\ldots \xrightarrow{t_{k-1}} (M_k.N_k, D_k)$. By induction hypothesis, there exists a sequence of delays and transitions in $SCTA(\mathcal{W})$ that leads to a configuration (l_k, μ_k) where $l_k = (M_k.N_k, D_k, \xi_k, trans_k)$ is an extended state class with invariant inv_k and μ_k is a valuation of clocks. Assume that a fully enabled transition t_k is firable from $(M_k.N_k, v_k)$, and let h be the index of move at which t_k was enabled. Still using the completeness argument, this means that t_k is firable from $(M_k.N_k, D_k)$. Hence, there exists a transition from $l_k = (M_k.N_k, D_k, \xi_k, trans_k)$ to some $l_{k+1} = (M_{k+1}.N_{k+1}, D_{k+1}, \xi_{k+1}, trans_{k+1})$ labeled by transition t_k. Assume that t_k is not a transition that has timeout. Then the move is attached with guard $g ::= \alpha(t_k) \leq trans^{-1}(t_k)$. We know that in configuration $(M_k.N_k, v_k)$ of $\mathcal{W}$, $v_k(x_{t_k}) = d_k + \sum\limits_{i \in h..k-1} d_i \geq \alpha(t_k)$. As t_k did not timeout, the clock $x_h = trans_k^{-1}(t_k)$ was never reset since the h^{th} transition. So, $\mu_k = v_k(x_{t_k})$, and t_k can fire from (l_k, μ_k) after d_k time units. Now assume that t_k is an urgent transition that became urgent after a timeout. Then, we have $d_k = 0$, and the invariant $Inv(l_k) = trans_k^{-1}(t_k) \leq 0$ attached to l_k does not let time elapse, and t_k can fire, and cannot let a duration of more than d_k time units elapse. □

We are now ready to compare expressiveness of waiting nets and their variants w.r.t. other types of time Petri nets, and with timed automata. For a given class $\mathcal{N}$ of net, we will denote by B-$\mathcal{N}$ the bounded subclass of $\mathcal{N}$, add the subscript ϵ if transitions with ϵ labels are allowed in the model, and a superscript $\overline{inj}$ if the labeling of transitions is non-injective. For instance $B\text{-}WTPN_{\epsilon}^{\overline{inj}}$ denotes the class of bounded waiting nets with non-injective labeling and ϵ transitions. It is well known that adding ϵ moves to timed automata increases the expressive power of the model [30]. Similarly, allowing non-injective labeling of transitions or adding ϵ transitions increases the expressive power of nets [25]. Lastly, adding stopwatches to timed automata or bounded time Petri nets makes them Turing powerful [16].

Theorem 5.4. $BWTPN <_{\mathcal{L}} TA(\leq, \geq)$.

Proof:
From Proposition 5.3, we can translate every bounded waiting net $\mathcal{W}$ to a finite timed automaton $SCTA(\mathcal{W})$. Notice that $SCTA(\mathcal{W})$ uses only constraints of the form $x_i \geq a$ in guards and of the form $x_i \leq b$ in invariants. Thus, $BWTPN \subseteq TA(\leq, \geq)$. This inclusion is strict. Consider the timed automaton $\mathcal{A}_1$ of Figure $8{-}a)$. Action a can occur between date 2 and 3 and b between date 4 and 5.

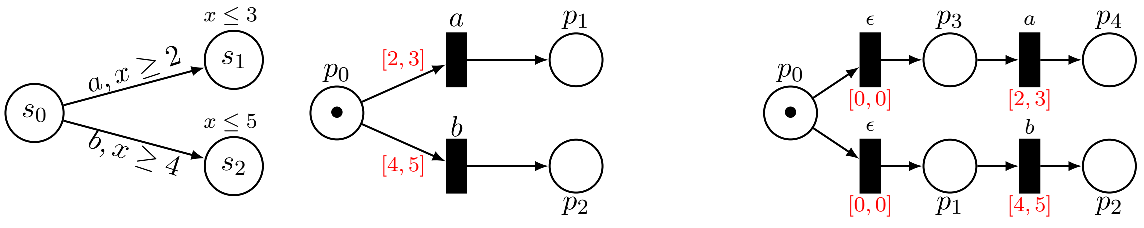


Figure 8. a) A timed automaton $\mathcal{A}_1$ b) An equivalent time<u>d</u> Petri net c) An equivalent TPN in TPN_{ϵ}

The timed language of $\mathcal{A}_1$ cannot be recognized by a BWTPN with only two transitions t_a and t_b, because t_a must be firable and then must fire between dates 2 (to satisfy the guard) and 3 (to satisfy the invariant in s_1). However, in TPNs and WTPNs, transitions that become urgent do not let time elapse, and cannot be disabled without making a discrete move. As t_b is the only other possible move, but is not yet allowed, no WTPN with injective labeling can encode the same behavior as $\mathcal{A}_1$. $\square$

Remark 5.5. It was proved in [25] that timed automata (with $\epsilon-$transitions) have the same expressive power as bounded TPNs with $\epsilon-$transitions. These epsilon transitions can be used to "steal tokens" of an enabled transition, and prevent it from firing urgently after a delay. This cannot be done with TPNs nor waiting nets without ϵ, because a transition that is fully enabled can be disabled only by a discrete move, and not by elapsing time. Hence, bounded TPN with $\epsilon-$transitions are strictly more expressive than waiting nets, and than waiting net with non-injective labeling.

Remark 5.6. Another easy result is that timed Petri nets and waiting nets are incomparable. Indeed, timed Petri nets cannot encode urgency of TPNs, and as a consequence some (W)TPNs have no timed Petri net counterpart, even in the bounded case. Similarly, one can design a timed Petri net in which a transition is firable only in a bounded time interval and is then disabled when time elapses. We have seen in Figure 8-a) that $\mathcal{L}(\mathcal{A}_1)$ cannot be recognized by a waiting net. However, it is easily recognized by the timed Petri net of figure 8-b).

Theorem 5.7. $TPN <_{\mathcal{L}} WTPN$ and $BTPN <_{\mathcal{L}} BWTPN$.

Proof:
Clearly, from Definition 3.1, we have $TPN \leq_{\mathcal{L}} WTPN$ and $BTPN \leq_{\mathcal{L}} BWTPN$, as TPNs are WTPNs without control places. It remains to show that inclusions are strict. Consider the waiting net $\mathcal{W}$ in Figure 9-a. We have $T = \{t_0, t_1\}$, $P = \{p_0, p_1\}, C = \{c_0\}$ and $M_0.N_0 = \{p_0\}\{\}$. Hence, from the starting configuration, t_0 is fully enabled and firable (because $M_0 \geq {}^\bullet(t_0)$), but t_1 is enabled and not firable ($M_0 \geq {}^\bullet(t_1)$ and $N_0(c_0) = 0$). Every valid run of $\mathcal{W}$ is of the form $\{p_0\}\{\} \xrightarrow{(d_0,t_0)} \{\}\{c_0\} \xrightarrow{(d_1,t_1)} \{p_1\}\{\}$ where $0 \leq d_0 \leq 20$ and $d_0 + d_1 = 20$.

Thus the timed language of $\mathcal{W}$ is $\mathcal{L}(\mathcal{W}) = \{(t_0, d_0)(t_1, 20) \mid 0 \leq d_0 \leq 20\}$. Let us show that there exists no TPN that recognizes the language $\mathcal{L}(\mathcal{W})$. In TPN, one cannot memorize the time already elapsed using the clocks of newly enabled transitions. A TPN $\mathcal{N}$ recognizing $\mathcal{L}(\mathcal{W})$ should satisfy the following properties:

1. The TPN should contain at least two different transitions namely t_0 and t_1
2. t_0 and t_1 are the only transitions which fire in any run of $\mathcal{N}$.
3. t_0 and t_1 are fired only once.
4. t_0 must fire first and should be able to fire at any date in $[0, 20]$ units.
5. t_1 must fire second at time 20 units, regardless of the firing date of t_0

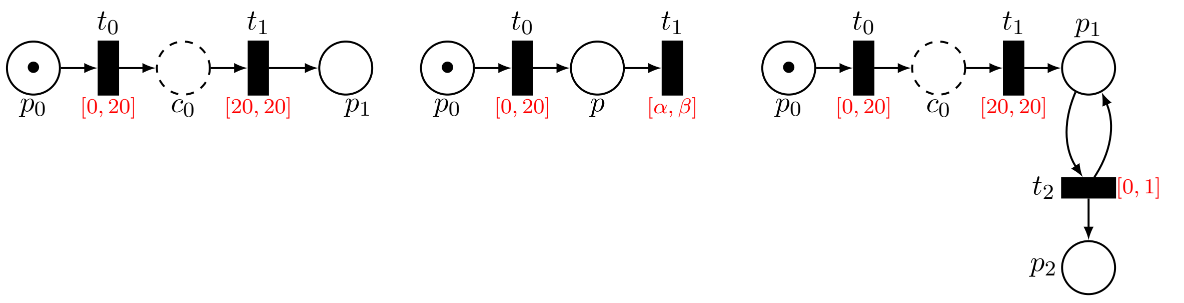


Figure 9. $a)$ A waiting net $\mathcal{W}$ $b)$ a part of TPN needed to encode $\mathcal{L}(\mathcal{W})$ $c)$ An unbounded waiting net

The above conditions are needed to ensure that $\mathcal{L}(\mathcal{N}) = \mathcal{L}(\mathcal{W})$. We can now show that it is impossible to build a net $\mathcal{N}$ satisfying all these constraints. Since t_1 must fire *after* t_0 , $\mathcal{N}$ must contain a subnet of the form shown in figure 9-b, where p is an empty place preventing firing t_1 before t_0. Notice however that p forbids enabling t_1 (and hence measuring time) from the beginning of a run. We can force t_0 to fire between 0 and 20 units with the appropriate time interval $[0, 20]$, but the TPN of Figure 9-b) can not remember the firing date of t_0, nor associate to t_1 a clock which value increases before t_0 fires. Let $[\alpha, \beta]$ be the time interval associated to t_1, and consider the two extreme but legal firing dates of t_0, namely 0 and 20 time units. Allowing these two dates amounts to requiring that $0 + \alpha(t_1) = 0 + \beta(t_1) = 20 + \alpha(t_1) = 20 + \beta(t_1) = 20$ which is impossible. Hence, there exists no TPN recognizing $\mathcal{L}(\mathcal{W})$. This shows that $BTPN <_{\mathcal{L}} BWTPN$. The proof easily extends to show that $TPN <_{\mathcal{L}} WTPN$, simply by adding an unbounded part of net that becomes active immediately after firing t_1, as depicted in the waiting net of Figure 9-c. □

As mentioned earlier in the paper, one can consider that in a run of waiting net, a transition with an empty control place in its preset can let its clock reach the upper bound of its interval, and then freeze it, waiting for the control place to be filled. A *stopwatch time Petri net* (SwTPN for short) is a TPN with an additional relation $sw : P \rightarrow T$, defining stopwatch arcs. An enabled transition t such that $M \geq sw(t)$ is said *active*, and its clock evolves during timed moves. An enabled transition such that $M < sw(t)$ is said *suspended* and it clock does not change during timed move until it becomes active again. A natural question is hence to compare waiting nets and SwTPNs. The first thing to notice is that bounded SwTPNs (BSwTPNs) are already Turing powerful [15] while bounded waiting nets are not. So we immediately obtain the following proposition.

Proposition 5.8. There exists bounded SwTPNs that cannot be simulated by bounded waiting nets.

One interesting point to remark is that expressiveness of waiting nets and SwTPNs have different origins. Waiting nets (and TPNs) can encode instructions of a Counter Machine only if they are unbounded. For stopwatch Petri nets, counter values in a Minsky Machine can be encoded by differences between upper and lower bounds on clock constraints in domains [15], and this encoding works even whith bounded SwTPNs. We can go further and prove that bounded waiting nets and BSwTPNS are incomparable. Consider again the example of Figure 9-a. We already know that the language of this bounded waiting net cannot be encoded by a bounded TPN with only two transitions and an injective labeling. Now, allowing stopwatch arcs still does not alllow for the encoding of a SwTPN recognizing the same timed language as the waiting net of Figure 9-a.

Proposition 5.9. Bounded waiting nets and bounded stopwatch Petri nets are uncomparable.

Proof:
We already know from Proposition 5.8 that some BSwTPNs cannot be simulated by waiting nets. It remains to show the converse direction. Assume that there exists a BSwTPN that can encode the same language as the waiting net of Figure 9-a. We know from the proof of Theorem 5.7 that, to encode this example with a TPN that has only two transitions t_0 and t_1, if $(t_0)^\bullet \cap {}^\bullet(t_1) \neq \emptyset$, then, one has to enforce occurrence of t_1 at date $20 - d_0$ for every possible date d_0 for the firing of t_0. We also proved that this is impossible for any time interval $[\alpha, \beta]$ attached to t_1. Let us assume that $(t_0)^\bullet \cap {}^\bullet(t_1) \neq \emptyset$, and that there exists $p_{sw} \in sw(t_1)$, i.e. that a stopwatch arc from p_{sw} to t_1 controls clock and firing of t_1. If $p_{sw} \in (t_0)^\bullet$ then t_1 is suspended until t_0 fires, and we are still in a situation where we cannot enforce firing of t_1 exactly $20 - d_0$ time units after firing of t_0. If $p_{sw} \in {}^\bullet(t_0)$, then after firing of t_0, t_1 cannot fire. Let us now assume that $(t_0)^\bullet \cap {}^\bullet(t_1) = \emptyset$. We already know from Theorem 5.7 that without a stopwatch arc restricting firing of t_1, one cannot prevent t_1 from firing at date 20 before t_0. As in the preceding case, if we attach interval $[\alpha, \beta]$ to transition t_1 and require $p_{sw} \in (t_0)^\bullet$, then t_1 is initially suspended, and we mush have $[\alpha, \beta] = [20 - d_0, 20 - d_0]$ for all values of d_0, which is impossible. Now assume that $p_{sw} \in {}^\bullet(t_0)$. Then t_1 is suspended after firing of t_0 and hence we cannot fire it. We hence have $p_{sw} \notin {}^\bullet(t_0) \cup (t_0)^\bullet$, and as the contents of p_{sw} is unchanged by firing of t_0, then t_1 is either never firable if $M_0(p_{sw}) = 0$, or can fire before t_0 if $M_0(p_{sw}) > 0$ and we set $[\alpha, \beta] = [20, 20]$, or at an unwanted dates if we attach another arbitrary interval $[\alpha, \beta]$ to t_1. □

Theorem 5.10. Injective TPNs and WTPNS are strictly less expressive than their non-injective counterparts, i.e.

$$BTPN <_{\mathcal{L}} BTPN^{\overline{inj}}, \qquad TPN <_{\mathcal{L}} TPN^{\overline{inj}},$$
$$BWTPN <_{\mathcal{L}} BWTPN^{\overline{inj}}, \quad WTPN <_{\mathcal{L}} WTPN^{\overline{inj}}$$

Proof:
With non-injective labeling, (W)TPNs can recognize unions of timed language, which is not the case for models with injective labeling. In every timed word of the language of a model with injective labeling, a letter represents an occurrence of a transition. That is, every occurrence of some letter σ labeling a transition t_σ is constrained by time in a similar way in a word: if t_σ is enabled at some date d, then t_σ must occur later at some date d' with $d + \alpha(t_\sigma) \leq d' \leq \beta(t_\sigma)$ *at every position* in the considered word. This remark also holds for distinct words with the same prefix: let $w.(\sigma, d)$ be a timed word, with $w = (\sigma_1, d_1).(\sigma_2, d_2) \dots (\sigma_k, d_k)$. The possible values for d lay in an interval that only depends on the unique sequence of transitions followed to recognize w. With non-injective labeling, one can recognize a word w via several sequences of transitions, and associate different constraints to the firing date of σ. The union of this set of constraints need not be a single interval. Consider for instance the TPN of Figure 10, that defines the language $\mathcal{L}(\mathcal{N}) = \{(a, d_1).(b, d_2) \mid d_1 \in [0, 1] \wedge d_2 \in [d_1 + 4, d_1 + 5]\} \cup \{(a, d_1).(b, d_2) \mid d_1 \in [0, 1] \wedge d_2 \in [d_1 + 7, d_1 + 8]\}$. Hence, TPNs and WTPNs variants are strictly less expressive than their non-injective counterparts. □

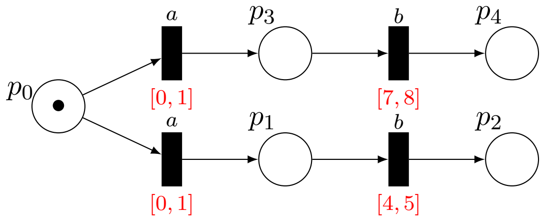


Figure 10. A TPN $\mathcal{N}_2$ with non-injective labeling.

Corollary 5.11. $BTPN^{\overline{inj}} <_{\mathcal{L}} BWTPN^{\overline{inj}}$

Proof:
Inclusion $BTPN^{\overline{inj}} \leq_{\mathcal{L}} BWTPN^{\overline{inj}}$ is straightforward from Definition 3.1. Now this inclusion is strict. Take the example of Figure 9-a), and attach distinct labels to t_0 and t_1. The language recognized cannot be encoded with a non-injective TPN, for the reasons detailed in the proof of Theorem. 5.7. □

To conclude on the effects of non-injective labeling, we can easily notice that $BWTPN^{\overline{inj}} \leq_{\mathcal{L}} TA(\leq,\geq)$ because the automaton construction of Definition 5.2 still works. Indeed, the automaton construction depends on transitions fired in extended state classes, and not on their labels. Hence, starting from a non-injective BWTPN, the construction of Definition 5.2 gives a timed automaton that recognizes the same language. As in the injective case, this relation is strict, because the example $\mathcal{A}_1$ of Figure 8-a cannot be implemented by a waiting net in $BWTPN^{\overline{inj}}$.

The last point to consider is whether allowing silent transitions increases the expressive power of the model. Consider again the automaton $\mathcal{A}_1$ of Figure 8-a. If transitions labeled by ϵ are allowed, then the timed language $\mathcal{L}(\mathcal{A}_1)$ can be encoded by the TPN of Figure 8-c. It was shown in [30] that timed automata with epsilon transitions are strictly more expressive than timed automata without epsilon. We hence have $TA(\leq,\geq) <_{\mathcal{L}} TA_{\epsilon}(\leq,\geq)$. We can also show that differences between WTPNs, TPN, and automata disappear when silent transitions are allowed.

Theorem 5.12. $TA_{\epsilon}(\leq,\geq) =_{\mathcal{L}} BTPN_{\epsilon} =_{\mathcal{L}} BWTPN_{\epsilon}$

Proof:
The equality $TA_{\epsilon}(\leq,\geq) = BTPN_{\epsilon}$ was already proved in [25]. From Definition 3.1, we have $BTPN_{\epsilon} \leq_{\mathcal{L}} WBTPN_{\epsilon}$. Further, for any waiting net $\mathcal{W} \in BWTPN_{\epsilon}$, one can apply the construction of Definition 5.2 to obtain a state class timed automaton $SCTA(\mathcal{W})$ (with ϵ transitions) recognizing the same language as $\mathcal{W}$. We hence have $BWTPN_{\epsilon} \leq_{\mathcal{L}} TA_{\epsilon}(\leq,\geq) =_{\mathcal{L}} BTPN_{\epsilon}$, which allows to conclude that $TA_{\epsilon}(\leq,\geq)$, $BTPN_{\epsilon}$, and $BWTPN_{\epsilon}$ have the same expressive power. □

Figure 11 summarizes the relations among different classes of nets and automata, including TPNs and automata with stopwatches. An arrow $\mathcal{M}_1 \longrightarrow \mathcal{M}_2$ means that $\mathcal{M}_1$ is strictly less expressive than $\mathcal{M}_2$, and this relation is transitively closed. Extensions of timed automata with stopwatches allow the obtained model to simulate two-counters machines (and then Turing machines) [16]. It is well known that unbounded TPNs can simulate Turing Machines too. More surprisingly, adding stopwatches to bounded TPNs already gives them the power of Turing Machines [15]. Obviously, all stopwatch models can simulate one another. Hence, these models are equally expressive in terms of timed languages as soon as they allow ϵ transitions. The thick dashed line in Figure 11 is the frontier for Turing powerful models, and hence also for decidability of reachability or coverability.

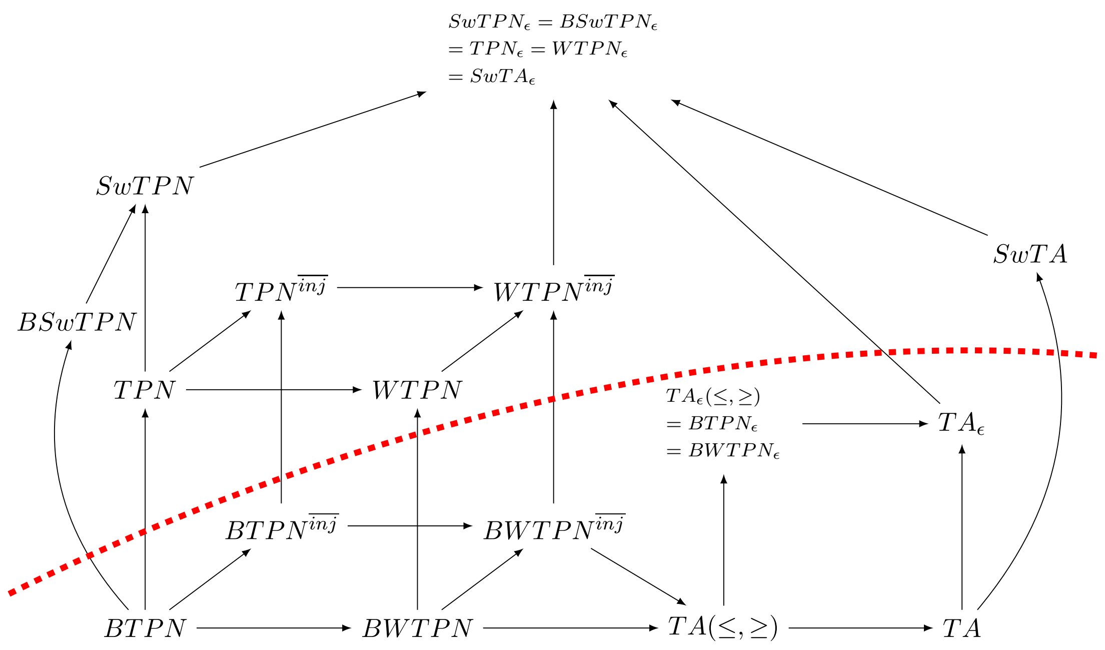


Figure 11. Relation among net and automata classes, and frontier of decidability.

6. Conclusion

We have proposed waiting nets, a new variant of time Petri nets, that measure time elapsed since enabling of a transition while waiting for additional control allowing its firing. This class obviously subsumes time Petri nets. More interestingly, expressiveness of bounded waiting nets lays between that of bounded TPNs and timed automata. Waiting nets allow for a finite abstraction of the firing domains of transitions. A consequence is that one can compute a finite state class graph for bounded WTPNs, and decide reachability and coverability.

As future work, we will investigate properties of classes of WTPN outside the bounded cases. In particular, we should investigate if being free-choice allows for the decidability of more properties in unbounded WTPNs [31]. A second interesting topic is control. Waiting nets are tailored to be guided by a timed controller, filling control places in due time to allow transitions firing. A challenge is to study in which conditions one can synthesize a controller to guide a waiting net in order to meet a given objective.

References

[1] Merlin PM. A Study of the Recoverability of Computing Systems. Ph.D. thesis, University of California, Irvine, CA, USA, 1974.

[2] Esparza J. Decidability and Complexity of Petri Net Problems - An Introduction. In: Proc. of Petri Nets, volume 1491 of *LNCS*. 1998 pp. 374–428. doi:10.1007/3-540-65306-6_20.

[3] Rackoff C. The Covering and Boundedness Problems for Vector Addition Systems. *Theor. Comput. Sci.*, 1978. **6**(2):223–231. doi:10.1016/0304-3975(78)90036-1.

[4] Mayr E. An Algorithm for the General Petri Net Reachability Problem. In: Proceedings of the 13th Annual ACM Symposium on Theory of Computing, May 11-13, 1981, Milwaukee, Wisconsin, USA. ACM, 1981 pp. 238–246. doi:10.1145/800076.802477.

[5] Popova-Zeugmann L. On Time Petri Nets. *J. Inf. Process. Cybern.*, 1991. **27**(4):227–244.

[6] Berthomieu B, Diaz M. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Trans. Software Eng.*, 1991. **17**(3):259–273. doi:10.1109/32.75415.

[7] Lime D, Roux O. Model Checking of Time Petri Nets Using the State Class Timed Automaton. *Discret. Event Dyn. Syst.*, 2006. **16**(2):179–205. doi:10.1007/s10626-006-8133-9.

[8] Ramchandani C. Analysis of asynchronous concurrent systems by timed Petri nets. Ph.D. thesis, Massachusetts Institute of Technology, 1974.

[9] Abdulla P, Nylén A. Timed Petri Nets and BQOs. In: ICATPN. LNCS 2075, p.53-70, 2001.

[10] Reynier P, Sangnier A. Weak Time Petri Nets Strike Back! In: Proc. of CONCUR 2009, volume 5710 of *LNCS*. 2009 pp. 557–571. doi:10.1007/978-3-642-04081-8_37.

[11] Ruiz VV, Gomez FC, Frutos-Escrig Dd. On Non-Decidability of Reachability for Timed-Arc Petri Nets. In: PNPM. IEEE Computer Society, 1999 pp. 188. doi:10.1109/PNPM.1999.796565.

[12] Akshay S, Genest B, Hélouët L. Decidable Classes of Unbounded Petri Nets with Time and Urgency. In: Kordon F, Moldt D (eds.), Application and Theory of Petri Nets and Concurrency - 37th International Conference, Petri Nets 2016, Toruń, Poland, June 19-24, 2016. Proceedings, volume 9698 of *Lecture Notes in Computer Science*. Springer, 2016 pp. 301–322. doi:10.1007/978-3-319-39086-4_18.

[13] Jacobsen L, Jacobsen M, Møller MH, Srba J. Verification of Timed-Arc Petri Nets. In: SOFSEM'11. LNCS 6543, p.46-72, 2011 . doi:10.1007/978-3-642-18381-2_4.

[14] Bowden F. Modelling time in Petri nets. In: Proc. Second Australia-Japan Workshop on Stochastic Models. 1996.

[15] Berthomieu B, Lime D, Roux O, Vernadat F. Reachability Problems and Abstract State Spaces for Time Petri Nets with Stopwatches. *Discret. Event Dyn. Syst.*, 2007. **17**(2):133–158. doi:10.1007/s10626-006-0011-y.

[16] Cassez F, Larsen K. The Impressive Power of Stopwatches. In: Palamidessi C (ed.), CONCUR 2000 - Concurrency Theory, 11th International Conference, University Park, PA, USA, August 22-25, 2000, Proceedings, volume 1877 of *Lecture Notes in Computer Science*. Springer, 2000 pp. 138–152. doi:10.1007/3-540-44618-4_12.

[17] Bucci G, Fedeli A, Sassoli L, Vicario E. Timed State Space Analysis of Real-Time Preemptive Systems. *IEEE Trans. Software Eng.*, 2004. **30**(2):97–111. doi:10.1109/TSE.2004.1265815.

[18] Parrot R, Briday M, Roux O. Timed Petri Nets with Reset for Pipelined Synchronous Circuit Design. In: Buchs D, Carmona J (eds.), Application and Theory of Petri Nets and Concurrency - 42nd International Conference, Petri Nets 2021, Virtual Event, June 23-25, 2021, Proceedings, volume 12734 of *Lecture Notes in Computer Science*. Springer, 2021 pp. 55–75. doi:10.1007/978-3-030-76983-3_4.

[19] Jones N, Landweber LH, Lien YE. Complexity of Some Problems in Petri Nets. *Theor. Comput. Sci.*, 1977. **4**(3):277–299. doi:10.1016/0304-3975(77)90014-7.

[20] Frutos-Escrig Dd, Ruiz V, Marroquín Alonso O. Decidability of Properties of Timed-Arc Petri Nets. In: Nielsen M, Simpson D (eds.), Application and Theory of Petri Nets 2000, 21st International Conference, ICATPN 2000, Aarhus, Denmark, June 26-30, 2000, Proceeding, volume 1825 of *Lecture Notes in Computer Science*. Springer, 2000 pp. 187–206. doi:10.1007/3-540-44988-4_12.

[21] Hélouët L, Agrawal P. Waiting Nets. In: Bernardinello L, Petrucci L (eds.), Proc. of PETRI NETS 2022, volume 13288 of *LNCS*. 2022 pp. 67–89. doi:10.1007/978-3-031-06653-5_4.

[22] Dill D. Timing Assumptions and Verification of Finite-State Concurrent Systems. In: Sifakis J (ed.), Automatic Verification Methods for Finite State Systems, International Workshop, Grenoble, France, June 12-14, 1989, Proceedings, volume 407 of *Lecture Notes in Computer Science*. Springer, 1989 pp. 197–212. doi:10.1007/3-540-52148-8_17.

[23] Alur R, Dill D. A Theory of Timed Automata. *Theor. Comput. Sci.*, 1994. **126**(2):183–235.

[24] Cassez F, Roux O. Structural translation from Time Petri Nets to Timed Automata. *Journal of Systems and Software*, 2006. **79**(10):1456–1468. doi:10.1016/j.jss.2005.12.021.

[25] Bérard B, Cassez F, Haddad S, Lime D, Roux OH. The expressive power of Time Petri nets. *TCS*, 2013. **474**:1–20. doi:10.1016/j.tcs.2012.12.005.

[26] Menasche M, Berthomieu B. Time Petri Nets for Analyzing and Verifying Time Dependent Communication Protocols. In: Rudin H, West CH (eds.), Protocol Specification, Testing, and Verification, III, Proceedings of the IFIP WG 6.1 Third International Workshop on Protocol Specification, Testing and Verification, organized by IBM Research, Rüschlikon, Switzerland, 31 May - 2 June, 1983. North-Holland, 1983 pp. 161–172.

[27] Menasche M. Analyse des réseaux de Petri temporisés et Application aux systèmes distribués. Ph.D. thesis, Univ. Paul Sabatier, Toulouse, 1982.

[28] Bengtsson J, Yi W. Timed Automata: Semantics, Algorithms and Tools. In: Desel J, Reisig W, Rozenberg G (eds.), Lectures on Concurrency and Petri Nets, Advances in Petri Nets, volume 3098 of *Lecture Notes in Computer Science*. Springer, 2003 pp. 87–124. doi:10.1007/978-3-540-27755-2_3.

[29] Cheng A, Esparza J, Palsberg J. Complexity Results for 1-Safe Nets. *Theor. Comput. Sci.*, 1995. **147**(1-2):117–136. doi:10.1016/0304-3975(94)00231-7.

[30] Diekert V, Gastin P, Petit A. Removing epsilon-Transitions in Timed Automata. In: Reischuk R, Morvan M (eds.), STACS 97, 14th Annual Symposium on Theoretical Aspects of Computer Science, Lübeck, Germany, February 27 - March 1, 1997, Proceedings, volume 1200 of *Lecture Notes in Computer Science*. Springer, 1997 pp. 583–594. doi:10.1007/BFb0023491.

[31] Akshay S, Hélouët L, Phawade R. Combining free choice and time in Petri nets. *J. Log. Algebraic Methods Program.*, 2020. **110**. doi:10.1016/j.jlamp.2018.11.006.

[32] Walter B. Timed Petri-Nets for Modelling and Analysing Protocols with Real-Time Characteristics. In: Proc. of PSTV. 1983 pp. 149–159. `https://dblp.org/rec/conf/ifip/Walter83.bib`.

[33] Hack M. Decidability Questions for Petri Nets. Ph.D. thesis, M.I.T., MIT, CA, USA, 1976.

[34] David A, Jacobsen L, Jacobsen M, Srba J. A Forward Reachability Algorithm for Bounded Timed-Arc Petri Nets. In: SSV'12, volume 102 of *EPTCS*. 2012 pp. 125–140. doi:10.4204/EPTCS.102.12.

[35] Dantzig G, Eaves BC. Fourier-Motzkin Elimination and Its Dual. *J. Comb. Theory, Ser. A*, 1973. **14**(3):288–297. doi:10.1016/0097-3165(73)90004-6.

[36] Bérard B, Cassez F, Haddad S, Lime D, Roux O. Comparison of the Expressiveness of Timed Automata and Time Petri Nets. In: FORMATS, volume 3829 of *Lecture Notes in Computer Science*. 2005 pp. 211–225. doi:10.1007/11603009_17.

[37] Bouyer P, Haddad S, Reynier PA. Timed Petri nets and timed automata: On the discriminating power of Zeno sequences. *Inf. Comput.*, 2008. **206**(1):73–107. doi:10.1016/j.ic.2007.10.004.

[38] Bérard B, Cassez F, Haddad S, Lime D, Roux OH. When are timed automata weakly timed bisimilar to time Petri nets ? *Theoretical Computer Science*, 2008. **403**(2-3):202–220. doi:10.1016/j.tcs.2008.03.030.

[39] Berthomieu B, Peres F, Vernadat F. Bridging the gap between timed automata and bounded Time Petri Nets. In: FORMATS'06, volume 4202 of *LNCS*. 2006 pp. 82–97. doi:10.1007/11867340_7.

[40] Walter B. Timed Petri Nets for Modelling and Analyzing Protocols with Real-time Characteristics. In: Protocol Specification, Testing, and Verification, III. 1983 pp. 149–159. `https://dblp.org/rec/conf/ifip/Walter83.bib`.

A. Fourier-Motzkin elimination

Fourier-Motzkin Elimination [35] is a method to eliminate a set of variables $V \subseteq X$ from a system of linear inequalities over X. Elimination produces another system of linear inequalities over $X \setminus V$, such that both systems have the same solutions over the remaining variables. Elimination can be done by removing one variable from V after another.

Let $X = \{x_1, \dots x_r\}$ be a set of variables, and w.l.o.g., let us assume that x_r is the variable to eliminate in m inequalities. All inequalities are of the form

$$c_1.x_1 + c_2.x_2 + \cdots + c_r.x_r \leq d_i$$

where c_j's and d_i are rational values, or equivalently $c_r.x_r \leq d_i - (c_1.x_1 + c_2.x_2 + \cdots + c_{r-1}.x_{r-1})$

If c_r is a negative coefficient, the inequality can be rewritten as $x_r \geq b_i - (a_{i,1}.x_1 + a_{i,2}.x_2 + \dots a_{i,r-1}.x_{r-1})$, and if c_r is positive, the inequality rewrites as $x_r \leq b_i - (a_{i,1}.x_1 + a_{i,2}.x_2 + \dots a_{i,r-1}.x_{r-1})$, where $b_i = \frac{d_i}{c_r}$ and $a_i = \frac{c_i}{c_r}$.

We can partition our set of inequalities as follows.

- inequalities of the form $x_r \geq b_i - \sum_{k=1}^{r-1} a_{ik}x_k$; denote these by $x_r \geq A_j(x_1, \dots, x_{r-1})$ (or simply $x_r \geq A_j$ for short), for j ranging from 1 to n_A where n_A is the number of such inequalities;
- inequalities of the form $x_r \leq b_i - \sum_{k=1}^{r-1} a_{ik}x_k$; denote these by $x_r \leq B_j(x_1, \dots, x_{r-1})$ (or simply $x_r \leq B_j$ for short), for j ranging from 1 to n_B where n_B is the number of such inequalities;

- inequalities $\phi_1, \ldots \phi_{m-(n_A+n_B)}$ in which x_r plays no role.

The original system is thus equivalent to:
$\max(A_1, \ldots, A_{n_A}) \leq x_r \leq \min(B_1, \ldots, B_{n_B}) \wedge \bigwedge_{i \in 1..m-(n_A+n_B)} \phi_i$.

One can find a value for x_r in a system of the form $a \leq x \leq b$ iff $a \leq b$. Hence, the above formula is equivalent to :
$\max(A_1, \ldots, A_{n_A}) \leq \min(B_1, \ldots, B_{n_B}) \wedge \bigwedge_{i \in 1..m-(n_A+n_B)} \phi_i$

Now, this inequality can be rewritten as system of $n_A \times n_B + m - (n_A + n_B)$ inequalities $\{A_i \leq B_j \mid i \in 1..n_A, j \in 1..n_B\} \cup \{\phi_i \mid i \in 1..m - (n_A + n_B)\}$, that does not contain x_r and is satisfiable iff the original system is satisfiable.

Remark A.1. The Fourier-Motzkin elimination preserves finiteness and satisfiability of a system of constraints. In general, the number of inequalities can grow in a quadratic way at each variable elimination. However, when systems describe firing domains of transitions and are in canonical form, they always contain less than $2 \cdot |T|^2 + 2 \cdot |T|$ inequalities, and then elimination produces a system of at most $2 \cdot |T|^2 + 2 \cdot |T|$ inequalities once useless inequalities have been removed.

B. Canonical forms : the Floyd-Warshall algorithm

A way to compute the canonical form for a firing domain D (i.e. the minimal set of constraints defining $[\![D]\!]$, the possible delays before firing of transitions) is to consider this domain as constraints on distances from variable's value to value 0, or to the value of another variable. The minimal set of constraints can then be computed using a slightly modified version of the Floyd-Warshall algorithm. Given a marking $M.N$, a firing domain D is defined with inequalities of the form:

- for every transition $t_i \in \mathsf{enabled}(M)$ we have an inequality of the form $a_i \leq \theta_i \leq b_i \forall t_i \in \mathsf{enabled}(M)$, with $a_i, b_i \in \mathbb{Q}^+$,
- for every pair of transitions $t_j \neq t_k \in \mathsf{enabled}(M)^2$ we have an inequality of the form $\theta_j - \theta_k \leq c_{jk} \forall t_j \neq t_k \in \mathsf{enabled}(M)$ with $c_{jk} \in \mathbb{Q}^+$

We then apply the following algorithm:

The system of inequalities input to the algorithm is satisfiable iff at every step of the algorithm $r \geq 0$. One can easily see that computing a canonical form, or checking satisfiability of a firing domain can be done in cubic time w.r.t. the number of enabled transitions.

C. Difference Bound Matrices

An interesting property of firing domains is that they can can be represented with Difference Bound Matrices (DBMs). To have a unified form for constraints we introduce a reference value **0** with the constant value 0. Let $\mathcal{X}_0 = \mathcal{X} \cup \{0\}$. Then any domain on variables in $\mathcal{X}$ can be rewritten as a conjunction of constraints of the form $x - y \preceq n$ for $x, y \in \mathcal{X}_0$, $\preceq \in \{<, \leq\}$ and $n \in \mathbb{Q}$.

Algorithm 1: Floyd-Warshall

```
Input: $D, E = \mathsf{enabled}(M)$;
for $t_k \in E$ do
    for $t_j \in E$ do
        for $t_i \in E$ do
            $r := \min(r, b_k - a_k)$
            $a_j := \max(a_j, a_k - c_{kj})$
            $b_i := \min(b_i, b_k + c_{ik})$
            $c_{ij} := \min(c_{ij}, c_{ik} + c_{kj})$
        end
    end
end
Output : $D^*$
```

For instance, $1 \leq \theta_1 \leq 2$ can be rewritten as $\mathbf{0} - \theta_1 \leq -1 \wedge \theta_1 - \mathbf{0} \leq 2$. Naturally, if the encoded domain has two constraints on the same pair of variables, we use their intersection: for a pair of constraints $\mathbf{0} - \theta_1 \leq -1 \wedge \mathbf{0} - \theta_1 \leq 0$ we only keep $\mathbf{0} - \theta_1 \leq -1$

A firing domain D can then be represented by a matrix M_D with entries indexed by $\mathcal{X}_0$:

- For each constraint $\theta_i - \theta_j \preceq n$ of D, $M_D[i,j] = (n, \preceq)$
- For each constraint $a_i \leq \theta_i \leq b_i$, that is equivalent to $\mathbf{0} - \theta_i \leq -a_i$ and $\theta_i - \mathbf{0} \leq b_i$, we set $M_D[0,i] = (-a_i, \preceq)$ and $M_D[i,0] = (b_i, \preceq)$
- For each clock difference $\theta_i - \theta_j$ that is unbounded in D, let $D_{ij} = \infty$
- Implicit constraints $\mathbf{0} - \theta_i \leq 0$ and $\theta_i - \theta_i \leq 0$ are added for all clocks.

Canonical DBM There can be infinite number of DBMs to represent a single set of solutions for a domain D, but each domain has a unique canonical representation, that can be computed as a closure of M_D or equivalently as a closure on a constraint graph. From matrix M_D, we create a directed graph $\mathcal{G}(D)$: Nodes of the graph are variables $\mathbf{0}$ and variables θ_1, θ_n. For each entry of the form $M_D[ij] = (n, \preceq)$ we have an edge from node θ_j to θ_i labeled with n. The tightest constraint between two variables θ_i and θ_j, is the value of the shortest path between the respective nodes in above graph. This can be done using the Floyd-Warshall algorithm shown above.

D. Proofs for Section 4

The following lemma shows that if a transition is firable, then one can always find an appropriate timing allowing for the computation of a successor.

Lemma D.1. Let $(M.N, D)$ be a state class, let t_f be a transition firable from that class. Then there exists a bound $r \in B^{t_f}_{M.N,D}$ such that $\mathsf{next}_r(D, t_f)$ is satisfiable.

Proof:
Let t_f be such a firable transition from the state class $(M \cdot N, D)$. This means, θ_f satisfies $D \wedge \bigwedge_{t_j \in \mathsf{FullyEnabled}(M \cdot N)} \theta_f \leq \theta_j$. Now, since t_f is firable, this implies $t_f \in E$ and $\forall t_j \in \mathsf{FullyEnabled}(M \cdot N)$ $a_f^* \leq b_j$ (i.e. t_f can be fired before any other transition becomes urgent) ... (1).

Let $[\, bnd_k, bnd_{k+1}]$ be the smallest interval between two consecutive bounds of $B_{M.N,D}$ containing a_f^*. Such an interval exists as $bnd_0 \leq a_f^* \leq b_f^* < \infty$. Now, we know that elimination preserves satisfiability. So the successor domain $\mathsf{next}_k(D, t_f)$ is satisfiable, because it is obtained by elimination from $D \wedge \bigwedge_{t_j \in \mathsf{FullyEnabled}(M \cdot N)} \theta_f \leq \theta_j \wedge bnd_k \leq \theta_f \leq b_{k+1}$ which contains at least one solution with $\theta_f = a_f^*$, and by adding new individually satisfiable constraints associated with newly enabled transitions. These constraints do not change satisfiability of domains obtained after elimination, because each one constrains only a single variable that was not used in the domain D'^{E,b_k}. Hence, choosing $r = k$ witnesses truth of the lemma. □

Proposition 4.8 For every run $\rho = (M_0.N_0, v_0) \dots (M_k.N_k, v_k)$ of $\mathcal{W}$ there exists a path π of $SCG(\mathcal{W})$ such that ρ and π coincide.

Proof:
Markings in a run ρ and a path π that coincide with ρ must be the same. To guarantee existence of π, it hence remains to show that there exists a consistent sequence of domains $D_1 \dots D_k$ such that path $\pi = (M_0.N_0, D_0).(M_1.N_1, D_1) \dots (M_k.N_k, D_k)$ is a path of $SCG(\mathcal{W})$ and coincides with ρ. We show it by induction on the size of runs.

- **Base Case :**
  Let $\rho = (M_0.N_0, v_0) \xrightarrow{e_1=(d_1,t_1)} (M_1.N_1, v_1)$. The corresponding path in SCG is of the form $\pi = (M_0.N_0, D_0) \xrightarrow{t_1} (M_1.N_1, D_1)$, where D_0 is already known. By definition we have $v_0(t) = 0$ for every enabled transition in $M_0.N_0$ and $v_0(t) = \perp$ for other transitions. After letting d_1 time units elapse, the net reaches configuration $(M_0.N_0, v_0')$ where $v_0'(t) = v_0(t) + d_1$ if $t \in \mathsf{FullyEnabled}(\mathsf{M_0.N_0})$, $v_0'(t) = \min(v(t) + d_1, \beta(t))$ if $t \in \mathsf{enabled}(\mathsf{M_0}) \setminus \mathsf{FullyEnabled}(\mathsf{M_0})$ and $v_0'(t) = \perp$ otherwise.

  The valuation v_1 is computed according to timed moves rules, which means that

$$v_1(t) = \begin{cases} 0 \text{ if } t \in\uparrow \mathsf{enabled}(M_1, t_1) \\ , v_0'(t) \text{ if } t \in Enabled(M_0) \cap Enabled(M_1) \\ \perp \text{ otherwise} \end{cases}$$

Now, since $\rho = (M_0.N_0, v_0) \xrightarrow{(d_1,t_1)} (M_1.N_1, v_1)$ is a valid move in $\mathcal{W}$, t_1 as a valid discrete move after the timed move d_1, which gives us the following condition :

$$\begin{cases} M_0.N_0 \geq {}^\bullet t_1 \\ M_1.N_1 = M_0.N_0 - {}^\bullet t_1 + t_1^\bullet \\ \alpha(t_1) \leq v_0'(t_1) \leq \beta(t_1) \\ \forall t, v_1(t) = 0 \textbf{ if } \uparrow \mathsf{enabled}(t, M_1, t_1) \textbf{ else } v_0'(t) \end{cases}$$

Further, as t_1 can be the first transition to fire, we have $v'_0(t_i) \leq \beta_i$ for every t_i in $\mathsf{FullyEnabled}(\mathsf{M}_0)$. As all clocks attached to fully enabled transitions have the same value d_1, it means that for every t_i we have $d_1 \leq \beta_i$. Let us now show that t_1 is firable from class C_0. Transition t_1 is firable from $M_0.N_0, D_0$ iff there exists a value θ_1 such that $\alpha(t_1) \leq v'_0(t_1) \leq \beta(t_1)$, and iff adding to D_0 the constraint that for every transition fully enabled $\theta_1 \leq \theta_j$ yields a satisfiable constraints. We can show that setting $\theta_1 = d_1$ allows to find a witness for satisfiability. Clearly, from the semantics, we have $\alpha(t_1) \leq \theta_1 = d_1 \leq \beta(t_1)$, and as no t_i fully enabled is more urgent than t_1, we can find θ_i, $d_1 \leq \theta_i \leq \beta_i$, and hence θ_i satisfies $\theta_1 \leq \theta_i$. As firability from D_0 holds, and $M_0.N_0 \geq {}^\bullet(t_1)$, there exists D_1 that is a successor of D_0 such that $(M_0.N_0, D_0).(M_1.N_1, D_1)$ is a path of $SCG(\mathcal{W})$.

- **Induction step:**
  Suppose that for every run ρ of size at most n of the waiting net, there exists a path π of size n in $SCG(\mathcal{W})$ that coincides with ρ. We have to prove that it implies that a similar property holds for every run of size $n+1$. Consider a run ρ from $(M_0.N_0, v_0)$ to $(M_n.N_n, v_n)$ of size n and the coinciding run π, of size n too, from $(M_0.N_0, D_0)$ to $(M_n.N_n, D_n)$. Assume that some transition t_f is firable after a delay d, i.e., $(M_n.N_n, v_n) \xrightarrow{(d,t_f)} (M_{n+1}.N_{n+1}, v_{n+1})$. We just need to show that t_f is firable from $C_n = (M_n.N_n, D_n)$.

  As t_f is firable from $(M_n.N_n, v_n)$ after d time units, we necessarily have $\alpha(t_f) \leq v_n(t_f) + d \leq \beta(t_f)$, and for every t_j fully enabled $v_n(t_f) + d \leq \beta(t_j)$ (otherwise t_j would become urgent). For every transition t_k enabled in $(M_n.N_n, v_n)$, let us denote by r_k the index in the run where t_k is last enabled in ρ. We hence have that $v_n(t_k) = min(\beta_i, \sum_{i=r_k+1..n} d_i)$. In the abstract run π we have $b_i = \beta_i - \sum\limits_{k\in r_i..n} a_{k,f})$ where $a_{k,f}$ is the lower bound of domain D_k for the time to fire of the transition used at step k and symmetrically $a_i = \max(0, \alpha_i - \sum\limits_{k\in r_i..n} b_{k,f})$.

  At every step of path π, at step k of the path, domain D_k sets constraints on the possible time to fore of enabled transitions. In every D_k, for every enabled transition t_i, we have $a_{k,i} \leq \theta_i \leq b_{k,i}$ for some values $a_{k,i} \leq b_{k,i} \leq \beta_i$, and additional constraints on the difference between waiting times. In particular, the set of transitions enabled in $M_n.N_n$ is not empty, and domain D_n imposes that $a_{n,f} \leq \theta_f \leq b_{n,f}$. Transition t_f is firable from D_n iff adding one constraint of the form $\theta_f \leq \theta_i$ per fully enabled transitions still yields a satisfiable domain. Now, we can show that the time spent in every configuration of $\mathcal{W}$ satisfies the constraints on value for the time to fire allowed for the fired transition at every step.

□

Lemma D.2. Let ρ be a run of size n an π be an abstract run (of size n too) that coincide. Then, for every transition $(M_i.N_i, v_i) \xrightarrow{d_i, t_f^i} (M_{i+1}.N_{i+1}, v_{i+1})$ we have $d_i \in [a_{i,f}, b_{i,f}]$.

Proof:
A transition t_f^i can fire at date $\sum\limits_{j\in 1..i} d_j$ iff $v_i(t_f) \in [\alpha_i, \beta_i]$. At every step i of a run, we have $a_{i,j} = \max(0, \alpha_j - \sum\limits_{q\in Rj+1..i-1} b_f^q)$ and $b_{i,j} = \max(0, \beta_j - \sum\limits_{q\in Rj+1..i-1} a_f^q)$ where b_f^q (resp. a_f^q) is the upper

bound (resp. the lower bound) of the interval constraining the value of firing time for the transition fired at step q. At step i if transition t_j is newly enabled, then $a_{i+1,j} = \alpha_j$ and $b_{i+1,j} = \beta_j$. Otherwise, $a_{i+1,j} = max(0, a_{i,j} - b^i_f)$ and $b_{i+1,j} = max(0, b_{i,j} - a^i_f)$. At step i, if t^i_f was newly enabled at step $i-1$ then we necessarily have $a_{i+1,j} = \alpha_j$, $b_{i+1,j} = \beta_j$ $v_i(t_f) = 0$, and hence $d_{i+1} \in [a_{i+1,j}, b_{i+1,j}]$. Now, let us assume that the property is met up to step i. if transition t_f fires at step $i+1$, we necessarily have $\alpha_f \leq v_i(t_f) + d_{i+1} \leq \beta_f$. This can be rewritten as $\alpha_f \leq \sum\limits_{q \in R_f+1..i} d_q + d_{i+1} \leq \beta_f$. Considering step i, we have $a_{i,f^i} \leq d_i \leq b_{i,f^i}$,and hence $\alpha_f - b_{i,f^i} \leq \sum\limits_{q \in R_f+1..i-1} d_q + d_{i+1} \leq \beta_f - a_{i,f^i}$. we can continue until we get $\alpha_f - \sum q \in R_f + 1..ib_{q,f^q} \leq d_{i+1} \leq \beta_f - \sum q \in R_f + 1..ia_{q,f^q}$, that is $d_{i+1} \in [a_{i+1,f}, b_{i+1,f}]$. $\square$

As one can wait d time units in configuration $(M_n.N_n, v_n)$, it means that for every fully enabled transition t_j, $v_n(t_j) + d \leq \beta_j$. It now remains to show that setting $\theta_f = d$ still allows for values for remaining variables in D_n. Setting $\theta_f = d$ and $\theta_f \leq \theta_i$ for every fully enabled transition amount to adding constraint $d \leq \theta_i$ to D_n. Further, we have $a_{n,f} \leq d \leq b_{n,f}$. We can design a constraint graph for D_n, where nodes are of the form $\{\theta_i \mid t_i \in FullyEnabled(M_n.N_n)\} \cup \{x_0\}$ where x_0 represents value 0, and an edge from θ_i to θ_j has weight $w > 0$ iff $\theta_i - \theta_j \leq w$. Conversely, a weight $w \leq 0$ represents the fact that $w \leq \theta_i - \theta_j$. Similarly, and edge of positive weight w from θ_i to x_0 represents constraint $\theta_i \leq w$ and an edge of negative weight $-w$ from x_0 to θ_i represents the fact that $w \leq \theta_i$. It is well known that a system of inequalities such as the constraints defining our firing domains are satisfiable iff there exists no negative cycle in its constraint graph. Let us assume that D_n is satisfiable, but $D'_n = D_n \uplus \theta_f = 0 \wedge \bigwedge\limits_{t_i FullyEnabled} d \leq \theta_i$ is not. It means that $CG(D_n)$ has no cycle of negative weight, but D'_n has one. Now, the major difference between D'_n is that there exists an edge $\theta_f \xrightarrow{d} x_0$, another one $x_0 \xrightarrow{-d} \theta_f$, and an edge $x_0 \xrightarrow{-d} \theta_i$ for every t_i that is fully enabled. Hence, new edges are only edges from/to x_0. If a negative cycle exists in $CG(D'_n)$, as D_n is in normal form, this cycle is of size two or three. If it is of size two, it involves a pair of edges $\theta_j \xrightarrow{b_{n,j}} x_0$ and $x_0 \xrightarrow{-d} \theta_j$. However, following lemma D.2, $d \leq b_{n,i}$ for every fully enabled transition t_i, so the weight of the cycle cannot be negative. Let us now assume that we have a negative cycle of size three, i.e. a cycle involving θ_i, θ_f and x_0, with edges $\theta_i \xrightarrow{c} \theta_f \xrightarrow{d} x_0 \xrightarrow{-d} \theta_i$. This cycle has a negative weight iff $c < 0$. However, we know that $\theta_i \geq \theta_f$, this is hence a contradiction. Considering a cycle with a value θ_k instead of θ_f leads to a similar contradiction, and we need not consider cycles of size more than 3 because D_n is in normal form, and hence the constraint graph labels each edge with the weight of the minimal path from a variable to the next one.

Last, using lemma D.1, as t_f is firable from $(M_n.N_n, D_n)$ there exists $D_{n+1} \in Post(C_n, t_f)$, and hence $\pi.(M_{n+1}.N_{n+1}, D_{n+1})$ is a path of the state class graph of $\mathcal{W}$ that coincides with $\rho.(M_n.N_n, v_n) \xrightarrow{(d,t_f)} (M_{n+1}.N_{n+1}, v_{n+1})$ $\square$

Proposition 4.9 Let π be a path of $SCG(\mathcal{W})$. Then there exists a run ρ of $\mathcal{W}$ such that ρ and π coincide.

Proof:
Since ρ and π must coincide, if $\pi = (M_0.N_0, D_0) \xrightarrow{t_1} (M_1.N_1, D_1) \dots (M_k.N_k, D_k)$, then $\rho = (M_0.N_0, v_0) \overset{(d_1,t_1)}{\longrightarrow} (M_1.N_1, v_1) \dots (M_k.N_k, v_k)$. Since successive markings in both π and ρ are computed in the same way from presets and postsets of fired transitions (i.e. $M_i.N_i = M_{i-1}.N_{i-1} -^\bullet (t_i) + (t_i)^\bullet$), we just have to show that for every abstract run π of $SCG(\mathcal{W})$, one can find a sequence of valuations $v_0, v_1, \dots v_k$ such that $\rho = (M_0.N_0, v_0) \overset{(d_1,t_1)}{\longrightarrow} (M_1.N_1, v_1) \dots (M_k.N_k, v_k)$ is a run of $\mathcal{W}$ and such that firing t_i after waiting d_i time units is compatible with constraint D_i. We proceed by induction on the length of runs.

Base Case : Let $\pi = (M_0.N_0, D_0) \xrightarrow{t_1} (M_1.N_1, D_1)$, where D_0 represents the firing domain of transitions from $M_0.N_0$. We have $D_0 = \{\alpha_i \leq \theta_i \leq \beta_i \mid t_i \in \mathsf{enabled}(M_0)\}$. Now, for t_1 to be a valid discrete move, there must be a timed move d_1 s.t. $(M_0.N_0, v_0) \xrightarrow{(d_1,t_1)} (M_1.N_1, v_1)$, which follows the condition :

$$\begin{cases} M_0.N_0 \geq {}^\bullet t_1 \\ M_1.N_1 = M_0.N_0 - {}^\bullet t_1 + t_1^\bullet \\ \alpha(t_1) \leq v_0(t_1) \oplus d_1 \leq \beta(t_1) \\ \forall t, v_1(t) = 0 \textbf{ if } \uparrow \mathsf{enabled}(t, M_1, t_1) \textbf{ else } v_0(t) \oplus d_1 \end{cases}$$

The conditions on markings are met, since t_1 is a transition from $M_0.N_0$ to $M_1.N_1$ in the SCG. Existence of a duration d_1 is also guaranteed, since, adding $\theta_1 \leq \theta_i$ for every fully enabled transition t_i to domain D_0 still allows firing t_1, i.e. finding a firing delay θ_1. Hence choosing as value d_1 any witness for the existence of a value θ_1 guarantees that no urgency is violated (valuation $v_0(t_i) + d_1$ is still smaller than β_i for every fully enabled transition t_i). Let $\rho = (M_0.N_0, D_0) \xrightarrow{(d_1,t_1)} (M_1.N_1, v_1)$, where v_1 is obtained from v_0 by elapsing d_1 time units and then resetting clocks of transitions newly enabled by t_1. Then ρ is compatible with π.

General Case : Assume that for every path π_n of the state class graph $SCG(\mathcal{W})$ of size up to $n \in \mathbb{N}$, there exists a run ρ_n of $\mathcal{W}$ such that ρ_n and π_n coincide. We can now show that, given a path $\pi_n = (M_0.N_0, D_0) \xrightarrow{t_1} (M_1.N_1, D_1) \dots (M_n.N_n, D_n)$ and a run $\rho_n = (M_0.N_0, v_0) \overset{(d_1,t_1)}{\longrightarrow} (M_1.N_1, v_1) \dots (M_n.N_n, v_n)$ such that ρ_n and π_n coincide, and an additional move $(M_n.N_n, D_n) \xrightarrow{t_f} (M_{n+1}.N_{n+1}, D_{n+1})$ via transition t_f, we can build a run ρ_{n+1} such that $\pi_{n+1} = \pi_n.(M_n.N_n, D_n) \xrightarrow{t_f} (M_{n+1}.N_{n+1}, D_{n+1})$ and ρ_{n+1} coincide. We have $D_{n+1} = \mathsf{next}_r(D_n, t_f) \in \mathsf{Post}(D_n)$ for some bound r. As we want ρ_{n+1} and π_{n+1} to coincide, we necessarily have $\rho_{n+1} = \rho_n.(M_n.N_n, v_n) \overset{(d,t_f)}{\longrightarrow} (M_{n+1}.N_{n+1}, v_{n+1})$, i.e. $M_{n+1}.N_{n+1}, t_f$ are fixed for both runs, v_{n+1} is unique once d and t_f are set, so we just need to show that there exists a value for d such that ρ_{n+1} and π_{n+1} coincide.

From $(M_n.N_n, v_n)$, d time units can elapse iff $v_n(t_i) + d \leq \beta_i$, for every transition t_i fully enabled in $M_n.N_n$, and t_f can fire from $(M_n.N_n, v_n)$ iff $v_n(t_f) + d \in [\alpha_f, \beta_f]$. Let us denote by r_i the index in π_n, ρ_n where transition t_i is last newly enabled, and by a_{i,t_j} (resp b_{i,t_j}) the lower bound (resp. upper bound) on value θ_j in domain D_i. Last let t_f^i be the transition fired at step i

We have that $v_n(t_i) = \min(\beta_i, \sum_{j\in r_i+1..n} d_j)$, $a_{n,j} = \alpha_j - \sum_{q\in r_j+1..n} b_{q,t_f^q}$, and $b_{n,j} = \beta_j - \sum_{q\in r_j+1..n} a_{q,t_f^q}$

If $(M_n.N_n, D_n) \xrightarrow{t_f} (M_{n+1}.N_{n+1}, D_{n+1})$ is enabled, then we necessarily have that $D_n \cup \theta_f \leq \bigvee \theta_i$ is satisfiable. Let us choose $d = \theta_f = a_{n,j}$ and show that this fulfills the constraints to fire t_f in ρ_{n+1}

$v_n(t_i) + d \leq \beta_i$ iff $v_n(t_i) + \alpha_j - \sum_{q\in r_j+1..n} b_{q,t_f^q} \leq \beta_i$, for every fully enabled transition t_i. iff $\sum_{q\in r_j+1..n} d_q + \alpha_j - \sum_{q\in r_j+1..n} b_{q,t_f^q} \leq \beta_i$ As we are looking for a run that coincides with π_{n+1}, we can assume wlog that we always choose the smallest value for d_q, i.e. we choose $d_q = a_{q,t_f^q}$. Hence, the inequality rewrites as $\sum_{q\in r_j+1..n} a_{q,t_f^q} + \alpha_j - \sum_{q\in r_j+1..n} b_{q,t_f^q} \leq \beta_i$, or equivalently $\alpha_j - \sum_{q\in r_j+1..n} b_{q,t_f^q} \leq \beta_i - \sum_{q\in r_j+1..n} a_{q,t_f^q}$. This amount to proving $a_{n,j} \leq b_{n,i}$. We can do a similar transformation to transform $v_n(t_f)+d \in [\alpha_f, \beta_f]$ into an inequality $v_n(t_f)+\alpha_j - \sum_{q\in r_j+1..n} b_{q,t_f^q} \leq \beta_f$, then transformed into $a_{n,f} \leq b_{n,f}$, and $\alpha_f \leq v_n(t_f) + d$ into $\alpha_f \leq v_n(t_f) + \alpha_j - \sum_{q\in r_j+1..n} b_{q,t_f^q}$, and then $\alpha_f \leq \sum_{q\in r_j+1..n} a_{q,t_f^q} + \alpha_f - \sum_{q\in r_j+1..n} b_{q,t_f^q}$, which can be rewritten as $a_{q,t_f^q} \leq b_{q,t_f^q}$. As $D_n \cup \theta_f \leq \bigvee \theta_i$ is satisfiable, the conjunction of these inequalities holds too. □

Lemma D.3. (Boundedness) For all i, j, k the constants a_i, b_i and c_{jk}, of a domain of any state class graph have the following bounds :

$$\begin{aligned} 0 &\leq a_i \leq \alpha(t_i) \\ 0 &\leq b_i \leq \beta(t_i) \\ -\alpha(t_k) &\leq c_{jk} \leq \beta(t_j) \end{aligned}$$

Proof:
First of all, every variable θ_i represents minimal and maximal times to upper bounds of interval, so by definition it can only be a positive value. We hence have $0 \leq a_i$, $0 \leq b_i$. Now to prove $a_i \leq \alpha(t_i)$ and $b_i \leq \beta(t_i)$ always hold, we will study the effect of every step to compute $\mathsf{next}_r(D, t_f)$.

Let us recall how $\mathsf{next}_r(D, t_f)$ is built. We first add $\theta_f \leq \theta_i$ to D for every fully enabled transition t_i, and the inequality $b_r \leq \theta_f \leq b_{r+1}$. We then do a variable substitution as follows. We write:

$$\theta_j := \begin{cases} b_j + \theta'_j \textbf{ if } b_j \leq b_r \textbf{ and } \mathsf{enabled}(M) \setminus \mathsf{FullyEnabled}(M.N) \\ \theta_f + \theta'_j \textbf{ if } b_j > b_r \textbf{ and } \mathsf{FullyEnabled}(M) \\ 0 \textbf{ otherwise} \end{cases}$$

After variable substitution we have inequalities of the form $a_i \leq \theta'_i + b_i \leq b_i$, $a_i \leq \theta'_i + \theta_f \leq b_i$, $b_r \leq \theta_f \leq b_{r+1}$, $\theta_f \leq \theta'_i$, $\theta'_i - \theta'_j \leq c_{ij}$ if t_i, t_j are both fully enabled, $a_i \leq \theta'_i + \theta_f \leq b_i$, and $\theta_i = 0$ for every enabled transition t_i reaching its upper bound b_i

We use Fourier-Motzkin elimination to remove variable θ_f. This elimination makes new positive values of the form $a'_j = \max(0, a_j - b_i)$ or $b'_j = \max(0, b_j - a_i)$ appear (See also Lemma D.5). Yet, we still have $a'_j \leq \alpha_j$ and $b'_j \leq \beta_j$.

Then addition new constraints for newly enabled transitions do not change existing constraints, and for every newly enabled transition t_i, we have $\alpha_i \leq \theta_i \leq \beta_i$. The last step consist in computing a canonical form. Remember that canonical forms consist in computing a shortest path in a graph. Hence D^* also preserves boundedness. (See also Lemma D.5) Now, in the canonical form, we can consider bounds for $\theta_j - \theta_k$, knowing that both values are positive. $-a'_k \leq \theta_j - \theta_k \leq b'_j$, and hence $-\alpha(t_k) \leq c_{jk} \leq \beta(t_j)$. □

Definition D.4. (linearity) Let $\mathcal{W}$n be a waiting net, and let $K_{\mathcal{W}} = \max_{i,j} \lfloor \frac{\beta_i}{\alpha_j} \rfloor$ A domain D is *linear* (w.r.t. waiting net $\mathcal{W}$) if, for every constraint in D, lower and upper bounds a_i, b_i of constraints of the form $a_i \leq \theta_i \leq b_i$ and upper bounds $c_{i,j}$ of difference constraints of the form $\theta_i - \theta_j \leq c_{i,j}$ are linear combination of α_i's and β_i's with integral coefficients in $[-K_{\mathcal{W}}, K_{\mathcal{W}}]$.

Obviously, the starting domain D_0 of a waiting net $\mathcal{W}$ is bounded and linear. We can now show that the successor domains reached when firing a particular transition from any bounded and linear domain are also bounded and linear.

Lemma D.5. Elimination of a variable θ_i from a firing domain of a waiting net preserves boundedness and linearity.

Proof:
Fourier-Motzkin elimination proceeds by reorganization of a domain D, followed by an elimination, and then pairwise combination of expressions (see complete definition of Fourier Motzkin elimination in appendix A). We can prove that each of these steps produces inequalities that are both linear and bounded.

Let D be a firing domain, and D' the domain obtained after choosing the fired transition t_f and the corresponding variable substitution. An expression in D' of the form $\theta_f - \theta_i \leq c_{f,i}$ can be rewritten as $\theta_f \leq c_{f,i} + \theta_i$. An expression of the form $\theta_i - \theta_f \leq c_{i,f}$ can be rewritten as $\theta_i - c_{i,f} \leq \theta_f$. We can rewrite all inequalities containing θ_f in such a way that they are always of the form $exp \leq \theta_f$ or $\theta_f \leq exp$. Then, we can separate inequalities in three sets :

- D^+, that contains inequalities of the form $exp^- \leq \theta_f$, where exp^- is either constant a_f or an expression of the form $\theta_i - c_{i,f}$. Let E^- denote expression appearing in inequalities of this form.
- D^-, that contains inequalities of the form $\theta_f \leq exp^+$, where exp^+ is either constant b_f or an expression of the form $\theta_i + c_{f,i}$. Let E^+ denote expression appearing in inequalities of this form.
- $D^{\bar{\theta_f}}$ that contains all other inequalities.

The next step is to rewrite D into an equivalent system of the form $D^{\bar{\theta_f}} \cup \max(E^-) \leq \theta_f \leq \min(E^+)$,and then eliminate θ_f to obtain a system of the form $D^{\bar{\theta_f}} \cup \max(E^-) \leq \min(E^+)$. This system can then be rewritten as $D^{\bar{\theta_f}} \cup \{exp^- \leq exp^+ \mid exp^- \in E^- \wedge exp^+ \in E^+\}$. One can easily see that in this new system, new constants appearing are obtained by addition or substraction of constants in D, and hence the obtained domain is still linear.

At this point, nothing guarantees that the obtained domain is bounded by larger α's and $\beta's$. Let us assume that in D, we have $0 \leq a_i \leq \alpha_i$, $0 \leq b_i \leq \beta_i$ and $-\alpha_k \leq c_{j,k} \leq \beta_j$. Then the last step of FME can double the maximal constants appearing in D (for instance when obtaining $\theta_j - c_{j,f} \leq \theta_i + c_{f,i}$ or its equivalent $\theta_j - \theta_i \leq +c_{f,i} + c_{j,f}$. However, values of a_i's and $b_i's$ can only decrease, which, after normalization, guarantees boundedness of $\theta_j - \theta_i$. □

Lemma D.6. Reduction to canonical form preserves linearity.

Proof:
It is well known that computing a canonical form from a domain D represented by a DBM Z_D amounts to computing the shortest path in a graph representing the constraints. Indeed, a DBM is in canonical form iff, for every pair of indexes $0 \leq i, j \leq |T|$, and for every index $0 \leq k \leq |T|$ we have $Z(i,j) \leq Z(i,k) + Z(k,j)$. The Floyd Warshall algorithm computes iteratively updates of shortest distances by executing instructions of the form $Z(i,j) := \min(Z(i,j), Z(i,k) + Z(k,j))$. Hence, after each update, if $Z(i,j)$ is a linear combination of $\alpha's$ and $\beta's$, it remains a linear combination. □

Lemma D.7. Fourier Motzkin elimination followed by reduction to canonical form preserves boundedness and linearity.

Proof:
From Lemma D.5 and Lemma D.6, we know that domains generated by FME + canonical reduction are linear. However, after FME, the domain can contain inequalities of the form $a_i' \leq \theta_i \leq b_i'$ with $a_i' \leq a_i \leq \alpha_i$ and $b_i' \leq b_i \leq \beta_i$. However, it may also contain inequalities of the form $x \leq \theta_i - \theta_j \leq y$ where $-2 \cdot \max(\alpha_i) \leq x$ and $y \leq \cdot \max(\beta_i)$. Now, using the bounds on values of $\theta_i's$, the canonical form calculus will infer $a_i' - b_j \leq \theta_i - \theta_j \leq b_i' - a_j'$, and we will have $-\alpha_i \leq \theta_i - \theta_j \leq \beta_j$. □

Lemma D.8. (Bounded Linearity) For all i, j, k the constants a_i, b_i and c_{jk}, of a domain of any state class graph are linear in α's and β's

Proof:
Clearly, the constraints in D_0 are linear in α's and β's see Definition (4.7), it remains to prove that, if D is bounded and linear, then for every fired transition t and chosen time bound r, $\mathsf{next}_k(D_r, t)$ is still bounded and linear. We already know that Fourier Motzkin Elimination, followed by canonical form reduction preserves boundedness and linearity (Lemma D.7). Addition of new constraints do not change constants of existing constraints, and the constants of new constraints (of the form $\alpha_i \leq \theta_i \leq \beta_i$ are already linear. Further, these constraints are completely disjoint from the rest of the domain (there is no constraint of the form $\theta_k - \theta_i \leq c_{k,i}$ for a newly enabled transition t_i). Hence computing a canonical form before or after inserting these variables does not change the canonical domain. So, computing D^* after new constraints insertion preserves linearity, and thus, the constants appearing in the constraints in domain $\mathsf{next}_k(D_r, t)$ are bounded and linear w.r.t. α's and β's. □

Proposition 4.10 The set of firing domains in $SCG(\mathcal{W})$ is finite.

Proof:
We know that a domain in a SCG is of form :

$$\begin{cases} a_i^* \leq \theta_i \leq b_i^* \\ \theta_j - \theta_k \leq c_{jk}^*. \end{cases}$$

By boundedness (Lemma D.3) we have proved that the constants $a^{*'}s, b^{*'}s$ and $c^{*'}s$ are bounded above and below by $\alpha's$ and $\beta's$ up to sign, we have also proved that they are linear combinations of $\alpha's$ and $\beta's$ (Lemma D.8). Now, it remains to show that there can only be finitely many such linear combinations, which was shown in [6]. Hence, the set of domains appearing in $SCG(\mathcal{W})$ is finite. □