Waiting Nets

Loïc Hélouët, Pranay Agrawal

Waiting Nets

Loïc Hélouët[1], Pranay Agrawal[2]

[1] University Rennes, Inria, CNRS, IRISA, France, `loic.helouet@inria.fr`
[2] 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. For many systems, one wants to 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* decoupling time measurement and control. Their semantics ignores clocks 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 and show that they are strictly more expressive than TPNs.

1 Introduction

Time Petri nets (TPNs) are an interesting model to specify cyber-physical systems introduced in [22]. 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 [15] (coverability [25], reachability [21], boundedness [25]...) are undecidable for TPNs. Yet, for the class of bounded TPNs, reachability [24] 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 [7,20].

There are many variants of Petri nets with time. An example is *timed Petri nets* (TaPN), 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 [1,26]. However, reachability remains undecidable [27]. We refer readers to [18] for a survey on TaPN and their verification. Without any notion of

	Reachability	coverability	Boundedness
Time Petri Nets	Undecidable [19]	Undecidable [19]	Undecidable [19]
(bounded)	Decidable	Decidable	–
Timed Petri nets	Undecidable [27]	Decidable [16,1]	Decidable [16]
(bounded)	Decidable	Decidable	–
Restricted Urgency	Undecidable [2]	Decidable [2]	Decidable [2]
(bounded)	Decidable	Decidable	–
Stopwatch Petri nets	Undecidable [8]	Undecidable [8]	Undecidable [8]
(bounded)	Undecidable [8]	Undecidable [8]	–
TPNR	Undecidable [23]	Undecidable [23]	Undecidable [23]
(bounded)	Decidable [23]	Decidable [23]	–
Waiting Nets	Undecidable (Rmk. 1)	Undecidable (Rmk. 1)	Undecidable (Rmk. 1)
(bounded)	PSPACE-Complete (Thm. 2)	PSPACE-Complete (Thm. 2)	–

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

urgency, TaPN cannot model delay expiration. In [2], a model mixing TaPN and urgency is proposed, with decidable coverability, even for unbounded 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 resources. To solve this problem, [8] 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 [10]. For similar reasons, time Petri nets with preemptable resources [9], 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". This model is strictly more expressive than TPN, as TPN are a simple syntactic restriction of waiting nets. Waiting nets allow clocks of enabled transitions to reach their upper bounds, and wait for missing control to fire. A former attempt called Timed Petri nets with Resets (TPNR) distinguishes some delayable transitions that can fire later than their upper bounds [23]. 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, 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. Our last contribution is a study of the expressiveness of waiting nets w.r.t timed language equivalence. Interestingly, the expressiveness of bounded waiting nets lays between that of bounded TPNs and timed automata. Due to space limitation, proofs in this paper are only sketched, but can be found in an extended version [17].

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 encoded with Difference bound Matrices (DBMs) and their satisfiability checked in $O(n^3)$ [14]. 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 $w = (\sigma_1, d_1)(\sigma_2, d_2)\ldots$ such that $d_i \leq d_{i+1}$. A timed language is a set of timed words. Timed automata [4] are frequently used to recognize timed languages.

Definition 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), 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. For completeness, we recall the semantics of timed automata in appendix. The timed language recognized by $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A})$.

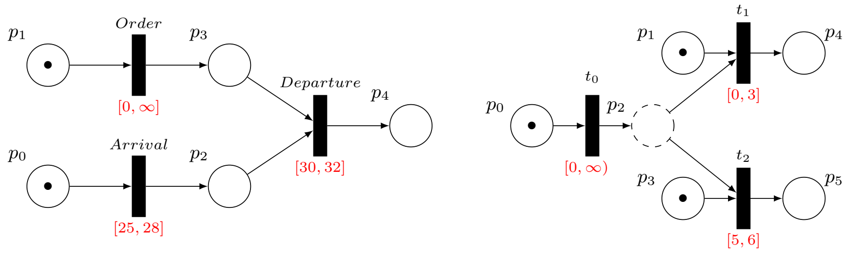

Fig. 1. A simple TPN $a)$ and a simple waiting net $b)$

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, and in particular, the solution of [20] 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

TPN are a powerful model: they can be used to encode a two-counter machine, and can hence simulate the semantics of many other formal models. A counterpart to this expressiveness is that most problems (reachability, coverability, 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, in which timing information is symbolically represented by firing domains. For many applications, working with bounded resources is sufficient. However, TPN 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.

Consider the example of Figure 1, that represents 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 beginning of a run of the net. 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*. The time constraint attached to *Departure* is an interval of the form $[30,32]$. 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". The TPN of Figure 1-a) does not implement this 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 more that 32 minutes in station, if the order is not released first. Similarly, Timed Petri nets, that do not have a notion of urgency, cannot encode this scenario where a transition has to fire after 32 time units.

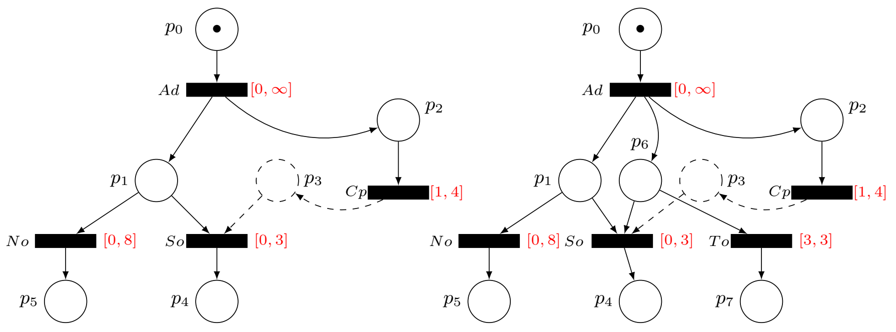

Fig. 2. a) Decoupled time and control in a waiting net. b) ... with a timeout transition.

We propose an extension of TPNs called *Waiting nets* (WTPN 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.

Definition 2. *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 $\lambda(t)$,*
- *${}^\bullet() \in (\mathbb{N}^{P\cup C})^T$ 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 ($\alpha(t) \leq \beta(t)$).*

Labeling map λ can be injective or not. 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 that 3 days after the add cannot be used.

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. *(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 4 (Transition Firing). *Firing of a transition t from marking $M.N$ is done in two steps. It first 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 it 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 $M.N$.*

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

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 $t \notin \mathit{enabled}(M)$. To be consistent, for every value $r \in \mathbb{R}$, we set $\perp + r := \perp$.

Definition 5 (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 net is a pair $(M_0.N_0, v_0)$, where $v_0(x_t) = 0$ if $t \in$* $\mathsf{enabled}(\mathsf{M_0})$ *and $v_0(x_t) = \perp$ otherwise. A transition t is* firable *from configuration $(M.N, v)$ iff it is fully enabled, and $v(x_t) \in [\alpha(t), \beta(t)]$.*

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.

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

Timed moves let $d \in \mathbb{R}^{\geq 0}$ time units elapse, but leave markings unchanged. We adopt an *urgent semantics* that considers differently *fully enabled* transitions and *waiting* transitions. If t is a fully enabled transitions then, t *allows* elapsing of d time units from $(M.N, v)$ iff $v(t) + d \leq \beta(t)$. The new valuation reached after elapsing d time units is $v(t) + d$. If we already have $v(t) = \beta(t)$, then t does not allow time elapsing. We say that firing of t is *urgent*, that is t *has to* be fired or disabled by the firing of another transition before elapsing time. 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. Urgency does not apply to waiting transitions, which can let an arbitrary amount of time elapse when at least one control places 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 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 $d' < d$ $(M.N, v) \xrightarrow{d'} (M.N, v'')$.

Discrete moves fire transitions that meet their time constraints, and reset clocks attached to transitions newly enabled by token moves. A discrete move relation from configuration $(M.N, v)$ to $(M'.N', v')$ via transition $t_i \in T$ 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 6. *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 $\mathcal{W}$. 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 1. 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 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 [6], we do not define accepting conditions for runs of 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 $\mathcal{L}^U(\mathcal{W}) = \{w \in T^* \mid \exists w_\rho \in \mathcal{L}(\mathcal{W}), w \text{ is the untiming of } w_\rho\}$. 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 definitions 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). The net starts in an initial configuration $(M_0.N_0, v_0)$ where $M_0(p_0) = 1$ and $M_0(p_i) = 0$ for all other places in P, $N_0(p_3) = 0$, $v_0(Ad) = 0$ and $v_0(t) = \perp$ for all other transitions in T. From this configuration, one can let an arbitrary duration d_0 elapse before firing transition Ad, leading to a configuration $M_1.N_0$ with $M_1(p_1) = M_1(p_2) = 1$, and $v_1(Cp) = v_1(No) = v_1(So) = 0$. Then, one can let a duration smaller than 4 elapse and fire No, or let a duration between 1 and 4 time units elapse and fire Cp. Notice that the net cannot let more than 4 time units elapse before taking a discrete move, as firing of Cp becomes urgent 4 time units after enabling of the transition. Let us assume that Cp is fired after elapsing 2.3 time units. This leads to a new configuration $(M_2.N_2, v_2)$ where $M_2(p_1) = M_2(p_2) = 1, N_2(p_3) = 1$, $v_2(No) = v_2(S_o) = 2.3$. In this net, firing of So can only occur after firing of Cp, but yet time measurement starts for So as soon as ${}^{\circ}(So)$ 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 $M_0.N_0$, $M_1.N_0$, $M_2.N_2$ described above, plus two additional markings $M_3.N_0$ where $M_3(p_5) = 1$ and $M_4.N_0$ where $M_4(p4) = 1$. 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. We give a more complex example in [17].

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. For TPNs, the set of reachable valuations can be abstracted to get a finite set of domains, to build a *state class graph* [7]. 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.

Let t be a transition with $\alpha(t) = 3$ and $\beta(t) = 12$, and assume that t has been enabled for 1.6 time units. According to the semantics of WPNs, $v(x_t) = 1.6$, and t cannot fire yet, as $x_t < \alpha(t)$. Transition t can fire only after a certain duration θ_t such that $1.4 \leq \theta_t \leq 10.4$. Similar constraints hold for all enabled transitions. We will show later that these constraint are not only upper and lower bounds on $\theta_t's$, but also constraints of the form $\theta_i - \theta_j \leq c_{ij}$.

Definition 7 (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:*

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

A variable θ_i in a firing domain D over variables $\theta_1, \ldots, \theta_m$ 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. Now, maintaining an interval for values of $\theta_i's$ is not sufficient. Allowing a transition t_i to fire means that no other transition t_j becomes urgent before firing of t_i, i.e. that adding constraint $\theta_i \leq \theta_j$ for every fully enabled transition t_j still allows to find a possible value for θ_i. Then, assuming that t_i fires, the new firing domain D' over variables $\theta'_1, \dots, \theta'_q$ will constrain the possible values of $\theta'_j s$ for all transitions t_j that remain enabled after firing of t_i. As time progresses, we have $\theta'_j = \theta_j - \theta_i$, which gives rise to diagonal constraint of the form $\theta'_j - \theta'_k \leq c_{jk}$ after elimination of variables appearing in D.

A firing domain D defines a set of possible values for $\theta'_i s$. We denote by $[\![D]\!]$ the set of solutions for a firing domain D. Now, the way to define a set of solutions 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.

Definition 8 (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 $[\![D]\!]$. If two sets of constraints are equivalent then they have the same canonical form. The constraints we consider are of the form $K_1 \leq x \leq K_2$ and $K_1 \leq x - y \leq K_2$, where K_1, K_2 are rational values. This type of constraints can be easily encoded by *Difference Bound Matrices* [14]. Checking satisfiability of a domain D, or computing a canonical form D^* can be done in $O(n^3)$, where n is the number of variables (see [17] for details and [5] for a survey on DBMs). Syntactically, state classes and canonical forms of waiting nets have the same definition as those of TPNs: the fact that a transition is waiting or fully enabled does not affect the representation of constraints. Now, there is a major difference between state graphs of TPNs and those of waiting nets: for waiting nets, the maximal duration that can elapse in a state class in contrained by fully enabled transitions only. However, at the same time, when elapsing time, one has to adapt contraints attached to waiting transitions which clocks have reached their upper bound. In some sense, for waiting transitions, variable θ_t represents a *time to upper bound of intervals* rather than a *time to fire*. When computing the effect of firing a fully enabled transition, one has to consider which waiting transitions have reached their upper bounds. A consequence is that state class graphs of waiting nets are not deterministic, as a class has several successors via the same transition.

Following the semantics of section 3, a transition t_i can fire from a domain D if one can find a value for θ_i that does not violate urgency of other fully enabled transitions. However, the upper bound of waiting transitions should not prevent t_i from firing. To get rid of this upper bound, we can use the notion of *projection*.

Definition 9 (Projection). *Let D be a firing domain with variables a_i, b_i, c_{jk} set as in def. 7. The* projection *of D on its fully enabled transitions is a domain*
$D_{|full} = \{a_i \leq \theta_i \leq b_i \mid t_i \in \mathsf{FullyEnabled}(M.N)\}$
$\quad \cup \{a_i \leq \theta_i \leq \infty, \mid t_i \in \mathsf{Waiting}(M.N)\}$
$\quad \cup \{\theta_j - \theta_k \leq c_{jk} \in D \mid t_j, t_k \in \mathsf{FullyEnabled}(M.N)\}$.

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. We say that 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. So, t_i can be the next transition fired iff there exists a value θ_j greater than or equal to θ_i that does not exceed b_j for every fully enabled transition t_j.

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 $N > {}^{\circ}(t)$. 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.

Consider the waiting net of Figure 1-b. This net starts in a configuration $C_0 = (M_0.N_0, v_0)$ with $M_0(p_0) = M_0(p_1) = M_0(p_3) = 1$ $M_0(p) = 0$ for every other place, and $N_0(p_2) = 0$. From this configuration, one can let an arbitrary amount of time $\delta \in \mathbb{R}^{\geq 0}$ elapse. If $0 \leq \delta < 3$, then the value of clock x_1 is still smaller than the upper bound $\beta(t_1) = 3$. Then, if t_0 fires from $C_0' = (M_0.N_0, v_0 + \delta)$, the net reaches a new configuration $C_1 = (M_1.N_1, v_1)$ where $M_1(p_1) = M_1(p_3) = 1$, $M_1(p) = 0$ for every other place, and $N_0(p_2) = 1$. We have $v_1(x_0) = 0, v_1(x_1) = v_1(x_2) = \delta$. One can still wait before firing t_1 in configuration, i.e., t_1 is not urgent and can fire immediately of within a duration $3 - \delta$. Now, if $3 \leq \delta < 5$, then $v_1(x_1) = 3, v_1(x_2) < 5$ so 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. Conversely, if $\delta \geq 5$ then after firing t_0, the net is in configuration $C_2 = (M_1.N_1, v_2)$ where $v_2(x_1) = 3$ and $v_2(x_2) \in [5, 6]$, 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 by domain D.

Definition 10 (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 $bnd_{|B_{M.N,D}|+1} = \infty$.*

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 $\theta_f \leq \theta_j$ for every fully enabled transition t_j. 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 gives 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 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 11 (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 with $\theta_k = 0$. So the only information to remember is that t_k will be urgent as soon as it becomes fully enabled.

Definition 12 (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 $[b_r, b_{r+1}]$ with b_r, b_{r+1} consecutive 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 r. We call D^r the domain obtained this way. Every transition t_k in $\mathit{Enabled}(M.N)$ 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, in the next class, it is either newly enabled, or not enabled. We hence need to remove θ_f from inequalities, while preserving an equivalent set of constraints. This is achieved by elimination of variable θ_f from D'^{u,b_r}, for instance with the well known Fourier-Motzkin technique (see [17] 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 8) to obtain a final canonical domain $\mathsf{next}_r(D, t_f) = D'^{F,b_r\,*}$.

More than one transition can fire from $(M.N, D)$, and for a given firable transition t_f, $Post(D, t_f)$ contains one domain per bound in $B^{t_f}_{M.N,D}$. It is hence clear that a state class can have more than one successor, with different markings and domains. Now, if a waiting net has no control place, transitions are either enabled or fully enabled in every configuration. Step 1 of successor construction leaves the starting domain D unchanged, and consequently the state class built is exactly the standard construction for TPNs (see [7,20]). Let $\mathsf{Post}(C)$ be the set of successors of a class C. Then $|\mathsf{Post}(C)| \leq |\mathsf{enabled}(M.N)|^2$. 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 comes from markings, and that the set of domains appearing in state classes is finite.

Definition 13. *(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 $D_{i+1} = \mathsf{next}_r(D_i, t_i)$.

Proposition 1 (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 (sketch). 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 2 in [17]) 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. $\square$

Proposition 2 (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.*

Proposition 1 shows that every marking reached by a run of a waiting net appears in its state class graph. The proof of Proposition 2 uses a similar induction on runs 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 3. *The set of firing domains in $SCG(\mathcal{W})$ is finite.*

Proof (sketch). 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 [7] (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 $-\alpha(t_i) \leq c_{ij} \leq \beta(t_i)$. This does not yet prove that the set of domains is finite. We define domains that are *bounded and linear*, i.e. upper and lower bounded by some constants, and where 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 [17]) show that variable elimination, reduction to a canonical form, etc. preserve bounds and linearity (a similar result was shown in [7] 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. $\square$

This property of waiting nets is essential, as waiting nets allow to stop clocks. Bounded Petri nets with stopwatches do not have a finite state class representation, because clock differences in domains can take any value. WPNs do not have this kind of problem because clocks are stopped at a predetermined instant (when they reach the upper bound of an interval).

Corollary 1. *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 time to fire of enabled transitions. By definition of boundedness, there is a finite number of markings appearing in $SCG(\mathcal{W})$. By Prop. 3, the set of domains appearing in $SCG(\mathcal{W})$ is finite, so $SCG(\mathcal{W})$ is finite.$\square$

More precisely, if a net is k_P-bounded, there are at most k_P^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 $O(k_P^P \cdot (2 \cdot K_{\mathcal{W}} + 1)^{|T+1|^2})$. A direct consequence of Proposition 1, Proposition 2, and Corollary 1 is that many properties of bounded waiting nets are decidable.

Corollary 2 (Reachability and Coverability). *The reachability and coverability problems for bounded waiting nets are decidable and PSPACE-complete.*

Proof. For membership, given a target marking $M_t.N_t$ it suffices to explore non-deterministically runs starting from $(M_0.N_0, D_0)$ of length at most $|SCG(\mathcal{W})|$ to find marking $M_t.N_t$, or to find a marking that covers $M_t.N_t$. Such reachability questions are known to be 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 [12], 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 [15]. This question can be recast as a coverability question for waiting nets, thus establishing the hardness of coverability. $\square$

5 Expressiveness

A natural question is the expressiveness of waiting nets w.r.t other models with time. 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 says 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 [11,20]. We show next that one can extend the results of [20], 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 1 and Proposition 2, 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 time to fire. If we label moves by the name of the transition used to move from a state class to the next one, we obtain an automaton that recognizes the untimed language of $\mathcal{W}$. Further, one can decorate a state class graph with clocks and invariants to recover the timing information lost during abstraction.

Definition 14 (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 [20] 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 $trans^{-1}(t_i)$. Let $\mathbb{C}^{ex}$denote the set of all 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 15 (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 $XP_0 = \emptyset$.*
- *$\Sigma = \lambda(T)$, and $X = \bigcup_{(M.N,D,\chi,trans)} \chi \subseteq \{x_1, \dots x_{|T|}\}$ 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, $C_{ex} = (M.N, D, \chi, trans, XP)$ 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 $D' = next_r(D, t)$.*
  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, 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. Similarly, we can set
  $$trans'(x_k) = \begin{cases} trans(x_k) \setminus \mathsf{Disabled}(M.N, t) \text{ if } trans(x_k) \not\subseteq \mathsf{Disabled}(M.N, t) \\ \uparrow \mathsf{enabled}(M.N, t) \text{ if } x_k = x_i \\ \text{Undefined otherwise} \end{cases}$$
  $$XP' = XP \cap \mathsf{Enabled}(M - {}^{\bullet}(t)) \setminus \mathsf{FullyEnabled}(M'.N') \\ \cup \{t_k \in \mathsf{Enabled}(M'.N') \mid \theta_k \notin D'\}$$
  The guard g is set to $\alpha(t) \leq trans^{-1}(t)$. Let $\mathsf{Urgent}(C_{ex}, t, C'_{ex}) = XP \cap \mathsf{Enabled}(M - {}^{\bullet}(t)) \cap \mathsf{FullyEnabled}(M'.N')$. The set of clocks reset is $R = \{x_i\}$ if some clock is newly enabled, and $R = \emptyset$ otherwise. For the invariant, we have two cases. If $\mathsf{Urgent}(C_{ex}, t, C'_{ex}) = \emptyset$ i.e. if there is no transition of XP that becomes fully enabled (and hence urgent) after firing t, the invariant Inv' is set to $\bigwedge_{\substack{x_j \in trans^{-1}(\mathsf{FullyEnabled}(M'.N')), \\ t_k \in trans(x_j) \cap \mathsf{FullyEnabled}(M'.N')}} x_j \leq \beta(t_k)$. Conversely, if $\mathsf{Urgent}(C_{ex}, t, C'_{ex}) \neq \emptyset$ the invariant is set to $\bigwedge_{t_k \in \mathsf{Urgent}(C_{ex}, t, C'_{ex})} trans^{-1}(t_k) \leq 0$

Proposition 4. *Let $\mathcal{W}$ be a waiting net. Then $\mathcal{L}(SCTA(\mathcal{W})) = \mathcal{L}(\mathcal{W})$.*

Proof (sketch). 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. 1 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.□

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 $B-WTPN_{\epsilon}^{\overline{inj}}$ denotes the class of bounded waiting nets with non-injective labeling and ϵ transitions. It is well known that adding ϵ moves to automata increases the expressive power of the model [13]. Similarly, allowing non-injective labeling of transitions increases the expressive power of nets. Lastly, adding stopwatches to timed automata or bounded time Petri nets make them Turing powerful [10].

Theorem 1. $BWTPN <_{\mathcal{L}} TA(\leq, \geq)$.

Proof. From Proposition 4, 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 3. Action a can occur between date 2 and 3 and b between date 4 and 5.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 $\mathcal{A}_1$.

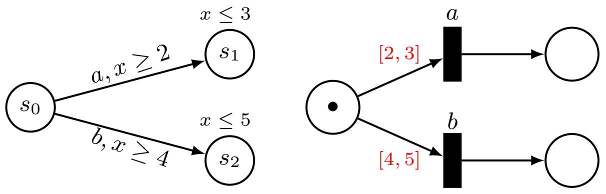

Fig. 3. a) A timed automaton $\mathcal{A}_1$ b) an equivalent time$\underline{d}$ Petri net

Remark 2. It was proved in [6] 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 a waiting transition, and prevent it from firing after a delay. This cannot be done with waiting nets without ϵ. Hence, bounded TPN with $\epsilon-$transitions are strictly more expressive than waiting nets, and than waiting net with non-injective labeling.

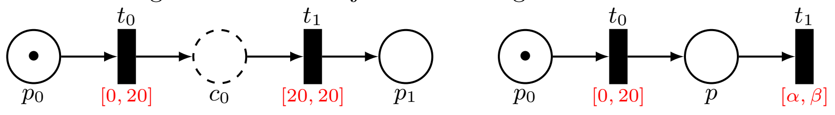

Fig. 4. *a*) A waiting net $\mathcal{W}$ *b*) a part of TPN needed to encode $\mathcal{L}(\mathcal{W})$.

Remark 3. 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 3-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 3-b).

Theorem 2. *$TPN <_{\mathcal{L}} WTPN$ and $BTPN <_{\mathcal{L}} BWTPN$.*

Proof (sketch). TPNs are WTPNs without control places so $TPN \leq_{\mathcal{L}} WTPN$ and $BTPN \leq_{\mathcal{L}} BWTPN$. We can show that inclusions are strict with the net $\mathcal{W}$ of Figure 4, that recognizes language $\mathcal{L}(W) = \{(t_0, d_0)(t_1, 20) \mid 0 \leq d_0 \leq 20\}$. Assuming that a TPN recognizes this language, it must contain the subnet of figure 4-b), for some values α, β. However, there is no assignment for α, β allowing to consider all values for d_0 in $\mathcal{L}(W)$ (see Appendix G in [17] for details). □

Theorem 3. *All injective classes are strictly less expressive than their non-injective counterparts, i.e. $BTPN <_{\mathcal{L}} BTPN^{\overline{inj}}$, $TPN <_{\mathcal{L}} TPN^{\overline{inj}}$, $BWTPN <_{\mathcal{L}} BWTPN^{\overline{inj}}$, and $WTPN <_{\mathcal{L}} WTPN^{\overline{inj}}$.*

Proof (sketch). With injective labeling, (W)TPNs can recognize unions of timed language, which is not the case for models with injective labeling. Let $\mathcal{N}_2$ be the TPN of Figure 5. We have $\mathcal{L}(\mathcal{N}_2) = \{(a, d_1).(b, d_2) \mid d_1 \in [0, 1] \wedge d_2 \in [d_1 + 4, d_1 + 5]\} \cup \{(a, d_1).(b, d_2) \mid d_1 \in [0, 1] \wedge d_2 \in [d_1 + 7, d_1 + 8]\}$. $\mathcal{L}(\mathcal{N}_2)$ is not recognized by any (waiting) net with injective labeling. □

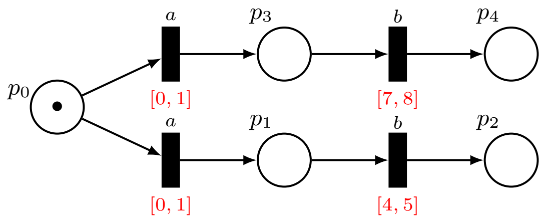

Fig. 5. A TPN $\mathcal{N}_2$ with non-injective labeling.

Corollary 3. *$BTPN^{\overline{inj}} <_{\mathcal{L}} BWTPN^{\overline{inj}}$*

Proof. Inclusion $BTPN^{\overline{inj}} \leq_{\mathcal{L}} BWTPN^{\overline{inj}}$ is straightforward from definition 2. Take the example of Figure 4-a). The language recognized cannot be encoded with a non-injective TPN, for the reasons detailed in the proof of Thm. 2. □

To conclude on the effects of non-injective labeling, we can easily notice that $BWTPN^{\overline{inj}} <_{\mathcal{L}} TA(\leq, \geq)$ because the automaton construction of Definition 15 still works (one labels transitions of the automaton with labels attached to transitions and keep the same construction). The last point to consider is whether allowing silent transitions increases the expressive power of the model. It was shown in [13] that timed automata with epsilon transitions are strictly more expressive than 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 4. *$TA_\epsilon(\leq,\geq) =_\mathcal{L} BTPN_\epsilon =_\mathcal{L} BWTPN_\epsilon$*

Proof. The equality $TA_\epsilon(\leq,\geq) = BTPN_\epsilon$ was already proved in [6]. Given $BWTPN_\epsilon$, one can apply the construction of Definition 15 to obtain a state class timed automaton (with ϵ transitions) recognizing the same language. □

Figure 6 shows 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. All extensions with clocks and stopwatches allow the considered model to simulate runs of Turing Machines. Actually, it has been shown that these models can encode two-counters machines (and then Turing machines). 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 red dashed line in Figure 6 is the frontier for Turing powerful models, and hence also for decidability of reachability or coverability.

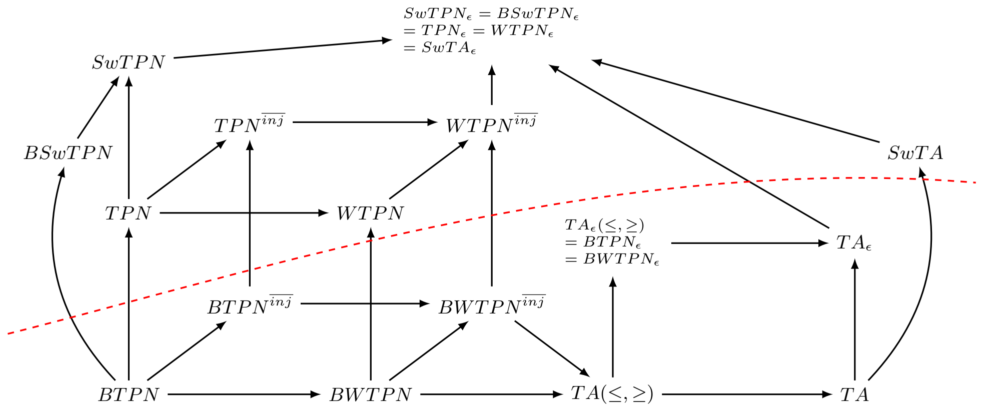

Fig. 6. Relation among net and automata classes, and frontier of decidability.

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 diagram 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 [3]. 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. P.A. Abdulla and A. Nylén. Timed Petri nets and BQOs. In *ICATPN*, LNCS 2075, p.53-70, 2001.
2. S. Akshay, B. Genest, and L. Hélouët. Decidable classes of unbounded Petri nets with time and urgency. In F. Kordon and D. Moldt, editors, *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*, pages 301–322. Springer, 2016.
3. S. Akshay, L. Hélouët, and R. Phawade. Combining free choice and time in Petri nets. *J. Log. Algebraic Methods Program.*, 110, 2020.
4. R. Alur and D.L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
5. J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003.
6. B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. H. Roux. The expressive power of time Petri nets. *TCS*, 474:1–20, 2013.
7. B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. Software Eng.*, 17(3):259–273, 1991.
8. B. Berthomieu, D. Lime, O.H. Roux, and F. Vernadat. Reachability problems and abstract state spaces for time Petri nets with stopwatches. *Discret. Event Dyn. Syst.*, 17(2):133–158, 2007.
9. G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario. Timed state space analysis of real-time preemptive systems. *IEEE Trans. Software Eng.*, 30(2):97–111, 2004.
10. F. Cassez and K.G. Larsen. The impressive power of stopwatches. In C. Palamidessi, editor, *CONCUR 2000 - Concurrency Theory, 11th International Conference, University Park, PA, USA, August 22-25, 2000, Proceedings*, volume 1877 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2000.
11. F. Cassez and O.H. Roux. Structural translation from time Petri nets to timed automata. *Journal of Systems and Software*, 79(10):1456–1468, 2006.
12. A. Cheng, J. Esparza, and J. Palsberg. Complexity results for 1-safe nets. *Theor. Comput. Sci.*, 147(1&2):117–136, 1995.
13. V. Diekert, P. Gastin, and A. Petit. Removing epsilon-transitions in timed automata. In R. Reischuk and M. Morvan, editors, *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*, pages 583–594. Springer, 1997.
14. D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems, International Workshop, Grenoble, France, June 12-14, 1989, Proceedings*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.
15. J. Esparza. Decidability and complexity of Petri net problems - an introduction. In *Proc. of Petri Nets*, volume 1491 of *LNCS*, pages 374–428, 1998.
16. D. de Frutos-Escrig, V.V. Ruiz, and O. Marroquín Alonso. Decidability of properties of timed-arc Petri nets. In M. Nielsen and D. Simpson, editors, *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*, pages 187–206. Springer, 2000.

17. L. Hélouët and P. Agrawal. Waiting nets (extended version). Technical report, INRIA, CMI & ENS Paris Saclay, 03 2022. `https://hal.inria.fr/hal-03613598`.
18. L. Jacobsen, M. Jacobsen, M. H. Møller, and J. Srba. Verification of timed-arc Petri nets. In *SOFSEM'11*. LNCS 6543, p.46-72, 2011.
19. N.D. Jones, L. H. Landweber, and Y. E. Lien. Complexity of some problems in Petri nets. *Theor. Comput. Sci.*, 4(3):277–299, 1977.
20. D. Lime and O.H. Roux. Model checking of time Petri nets using the state class timed automaton. *Discret. Event Dyn. Syst.*, 16(2):179–205, 2006.
21. E.W. Mayr. 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*, pages 238–246. ACM, 1981.
22. P. M. Merlin. *A Study of the Recoverability of Computing Systems.* PhD thesis, University of California, Irvine, CA, USA, 1974.
23. R. Parrot, M. Briday, and O.H. Roux. Timed Petri nets with reset for pipelined synchronous circuit design. In D. Buchs and J. Carmona, editors, *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*, pages 55–75. Springer, 2021.
24. L. Popova-Zeugmann. On time Petri nets. *J. Inf. Process. Cybern.*, 27(4):227–244, 1991.
25. C. Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6:223–231, 1978.
26. P.-A. Reynier and A. Sangnier. Weak time Petri nets strike back! In *Proc. of CONCUR 2009*, volume 5710 of *LNCS*, pages 557–571, 2009.
27. V. V. Ruiz, F. C. Gomez, and D. de Frutos-Escrig. On non-decidability of reachability for timed-arc Petri nets. In *PNPM*, pages 188–. IEEE Computer Society, 1999.