DOI 10.1007/s00165-007-0050-7
BCS © 2007
Formal Aspects of Computing (2008) 20: 225–238

Formal Aspects of Computing

Interval timed coloured Petri net: efficient construction of its state class space preserving linear properties

Hanifa Boucheneb

Department of Computer Engineering, École Polytechnique de Montréal,
Campus de l'Université de Montréal, 2500 chemin de Polytechnique, Montréal, QC H3T 1J4, Canada.
E-mail: hanifa.boucheneb@polymtl.ca

Abstract. We consider here the interval timed coloured Petri net model (ITCPN). This model associates with each created token a time interval specifying when the token will become available and forces enabled transitions to occur as soon as possible. This model can simulate other timed Petri nets and allows to describe large and complex real-time systems. We propose a much more efficient contraction for its generally infinite state space than those developed in the literature. Our contraction approach captures all linear properties of the model and produces finite graphs for all bounded models.

Keywords: Interval timed coloured Petri nets; State space; State class graph; Linear properties

1. Introduction

Coloured Petri Nets (*CPNs*) are widely used for modelling and analysis of large and complex systems [Jen82]. In these models, a colour (a value) is associated with each token allowing to make much compact and manageable descriptions. Several large systems have been successfully modelled and analyzed using these models. To describe systems whose behaviours are time dependent, the classical approach consists in introducing time in terms of intervals or durations labelling places, transitions or edges of the *CPN*. The resulting models are called *timed coloured Petri nets*. Another approach based on untimed coloured Petri nets and the concept of causal time have been proposed in [TKP02], where the progression of time is modelled by a special transition called *tick* which increments time by one unit whenever it occurs. However, this approach leads to larger models and reachability graphs than those we obtain for timed coloured Petri net. As an example, Fig. 1 right shows an untimed coloured Petri net corresponding to the timed model shown in Fig. 1 left. These models execute repeatedly every 3 time units the transition t_1. The reachability graphs of Fig. 1 left and Fig. 1 right consist respectively of one node and four nodes. Moreover, replacing the time parameter 3 by bigger values increases the graph size of Fig. 1 right but does not affect the graph size of Fig. 1 left. Therefore, timed coloured Petri nets are, in our opinion, more suitable for enumerative analysis than untimed coloured Petri nets.

We can find in the literature several timed coloured Petri nets such as *Van der Aalst*'s model [Van93], *Christensen*'s model [CKM01] and *Pao-Ann Hsiung*'s model [HsG02]. In *Pao-Ann Hsiung*'s model, a time interval is associated with each transition specifying its minimal and maximal firing delays. Time intervals of this model

Correspondence and offprint requests to: H. Boucheneb, E-mail: hanifa.boucheneb@polymtl.ca

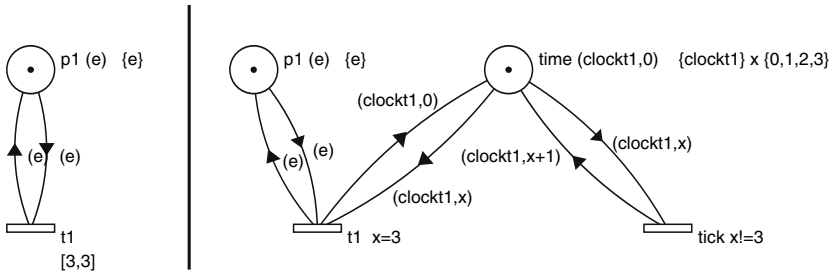

Fig. 1. Using untimed/timed coloured Petri net to model time

have the same semantics as those of Merlin's model (time Petri nets) [BeD91, MeF76]. In *Christensen*'s model, a date is associated with each created token. The token will become available at its date and still available until it is consumed. An enabled transition will occur as soon as possible (when all its required tokens become available). The *Van der Aalst*'s model called interval timed coloured Petri net (ITCPN) associates with each created token a time interval specifying when the token will become available (the earliest and latest times). As in *Christensen*'s model, an enabled transition will occur as soon as possible. Among these extensions, the model proposed by *Van der Aalst* seems to be more appropriate for the *CPN* model, since time intervals are associated with tokens instead of transitions. But, unlike *Pao-Ann Hsiung*'s model, the *Van der Aalst*'s model does not allow unbounded intervals and then expressing that a created token could be never available (be lost) is not possible. To overcome this limitation, we extend this model by allowing unbounded intervals.

To analyze this model, we propose to use an enumerative method (model checking) to verify linear properties of the model. As for other timed models [BeD91, CKM01, DOT96, DiS94, HHW97, HsG02, Vic01], this method needs an extra step for contracting its generally infinite state space into a finite graph which preserves properties of interest (linear properties). We say that a contraction preserves linear properties of some model, if it has the same firing sequences as the state space of the model.

Van der Aalst proposed in [Van93], a contraction method for the *ITCPN* model which is "sound" (i.e.: any firing sequence in the state space of the model is also possible in the contracted state space) but not "complete" (i.e.: some firing sequence in the contracted state space does not reflect any firing sequence in the model state space). Moreover, for models allowing infinite occurrence sequences, the *Van der Aalst*'s method would lead to infinite graphs. We propose here another contraction approach which has not these drawbacks. Our contraction approach generates finite graphs for all bounded[1] *ITCPNs* and preserves all linear properties of the model. Moreover, the characterization and computation of graph nodes are performed efficiently.

Firstly, we give, in Sect. 2, some definitions related to the *ITCPN* model and its behaviour. Section 3 deals with the *Van der Aalst*'s contraction approach of the *ITCPN* state space. Section 4 is devoted to our contraction approach. We develop here, a practical and efficient method for computing reachable state classes and then we show that our contraction generates finite graphs for all bounded *ITCPNs*. We also show, in this section, by means of an example how to use the resulting graphs to verify some timed linear properties. Finally, Sect. 5 concludes this paper.

2. Interval timed coloured Petri nets

We introduce here only necessary definitions and notations. For further details, we refer to [Jen82] for *CPNs* and to [Van93] for *ITCPNs*.

2.1. Definition of the ITCPN model

Definition 2.1 time domain and multi-sets

- The time domain is the set of all non-negative real numbers, i.e.: $\Re^+$.

[1] An *ITCPN* is bounded if and only if it has a finite number of reachable markings.

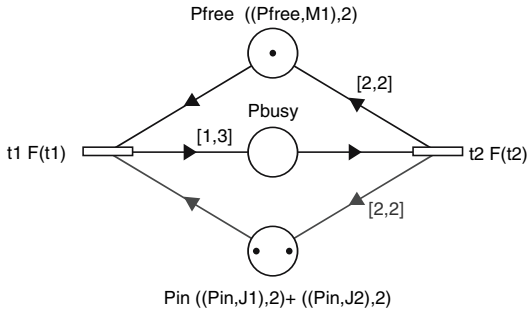

Fig. 2. An *ITCPN* model

- Let A be a set. A multi-set over the set A is a function N which associates with each element of set A, a non-negative integer number. It is represented by the following formal sum : $\sum_{a\in A} N(a)\cdot a$, where $N(a)$ is the occurrence number of a in N.
- Let A be a set, N_1 and N_2 two multi-sets over A. Operators $+, -, \leqslant, =$ on multi-sets are defined as usual:
  - $N_1 + N_2 = \sum_{a\in A}(N_1(a) + N_2(a))\cdot a$.
  - $N_1 \leqslant N_2$ if and only if, $(\forall\, a \in A, (N_1(a) \leqslant N_2(a)))$.
  - $N_1 = N_2$ if and only if, $(\forall\, a \in A, (N_1(a) = N_2(a)))$.
  - if $N_2 \leqslant N_1$ then $N_1 - N_2 = \sum_{a\in A}(N_1(a) - N_2(a))\cdot a$.

  We denote by A_{MS} the set of all multi-sets over A, and by $\emptyset$ the empty multi-set.

An *ITCPN* is a *coloured Petri net* augmented with time intervals associated with tokens. From the semantic point of view, each created token has a time stamp which can be any value inside its associated interval. The time stamp of a token indicates the delay required for the token to become available.

Definition 2.2 an *ITCPN* model
An *ITCPN* is a tuple $(\triangle, P, T, C, F, \mathrm{TM}_0)$ where:

- $\triangle$ is a finite set of types, called colour sets. Each colour set is finite.
- P is a finite and non empty set of places.
- T is a finite set of transitions such that $(P \cap T = \emptyset)$.
- $C : P \rightarrow$ Powerset($\triangle$). $C(p)$ is a finite set which specifies the set of allowed values (or colours) for any token of place p.
- Let CT be the set of all possible coloured tokens, i.e.: $CT = \{(p, c) \mid p \in P \wedge c \in C(p)\}$
  and INT the set of all intervals such that their bounds are rational numbers, i.e.:
  $INT = \{[y, z] \in Q^+ \times (Q^+ \cup \{\infty\}) \mid y \leqslant z\}$.[2]
  F is the transition function over T: $F(t) : \mathrm{Dom}(F(t)) \longrightarrow (CT \times INT)_{\mathrm{MS}}$,
  where $\mathrm{Dom}(F(t))$ is the definition domain of $F(t)$, $\mathrm{Dom}(F(t)) \subseteq CT_{\mathrm{MS}}$.
  $F(t)$ specifies which tokens are consumed and produced by firing transition t and also the interval inside which the time stamps of the produced tokens must be chosen (domains of their time stamps).
  Each transition is supposed to produce a finite set of tokens.
- TM_0 is the initial timed marking, $\mathrm{TM}_0 \in (CT \times Q^+)_{\mathrm{MS}}$.

2.2. *ITCPN* behaviour

We first explain the behaviour of an *ITCPN*, using an example given in [Van93] and reported here in Fig. 2.

[2] Unlike Van der Aalst's model, we allow here unbounded intervals.

Figure. 2 is the graphic representation of an *ITCPN* model, which is composed of three places $p_{\text{in}}, p_{\text{busy}}, p_{\text{free}}$, two transitions t_1, t_2 and three colour sets: $M = \{M_1, M_2, ..., M_s\}$ associated with the place p_{free}, $J = \{J_1, J_2, .., J_r\}$ associated with the place p_{in}, and $M \times J$ associated with the place p_{busy}.
It represents a jobshop, where jobs of place p_{in} are executed repeatedly. The jobshop is composed of one or several machines. Each machine is represented by one token, which is either in place p_{free} or in place p_{busy}. Tokens consumed and produced by firing transitions t_1 and t_2 are specified by functions $F(t_1)$ and $F(t_2)$:
$\forall j \in J, \forall m \in M$,

- $F(t_1)((p_{\text{in}}, j) + (p_{\text{free}}, m)) = (p_{\text{busy}}, (m, j), [1, 3])$.
  Which means that transition t_1 consumes two tokens one from each place p_{in} and p_{free}, and produces one token in place p_{busy}. When transition t_1 occurs, the time stamp of the created token may be any value inside interval [1, 3].
- $F(t_2)(p_{\text{busy}}, (m, j)) = (p_{\text{free}}, m, [2, 2]) + (p_{\text{in}}, j, [2, 2])$.
  Which means that transition t_2 consumes one token from place p_{busy} and produces two tokens one in each place p_{in} and p_{free}. When transition t_2 occurs, the time stamp of both created tokens is 2.

2.2.1. States of an ITCPN

To characterize the model state, we associate with each token a delay (a continuous variable), which is initialized at its creation with its time stamp. Afterwards, the delay decreases synchronously with time until it reaches zero. The state can be defined as a multi-set of timed tokens (i.e.: tokens completed with values of their delays).

Definition 2.3 timed token and timed marking

- A timed token is a 3-uple (p, c, v) where p is its place, c is its colour and v is the value of its delay.
- A timed marking TM is a multi-set of timed tokens, i.e.: $\text{TM} \in (CT \times R^+)_{\text{MS}}$. A state of an *ITCPN* is a timed marking.

Consider the previous model (Fig. 2). Its initial timed marking TM_0 is:$(p_{\text{free}}, M_1, 2) + (p_{\text{in}}, J_1, 2) + (p_{\text{in}}, J_2, 2)$.
It consists of three tokens not yet available. Tokens will become available after two time units.

2.2.2. State evolution

Initially, the model is in its initial timed marking. Afterwards, its state evolves either by time progressions (delays decrease with time) or by firing transitions.

Definition 2.4 events of a timed marking
Let TM be a timed marking and $U(\text{TM})$ the marking obtained from TM by eliminating all delays (the underlying untimed marking).

- Let t be a transition of T. Transition t is enabled for TM if and only if, all tokens required for its firing are present in TM, i.e.: $\exists m \in \text{Dom}(F(t)), m \leqslant U(\text{TM})$.
- An event e of TM is a pair composed with a transition enabled for TM and all timed tokens participating in its enabling (timed tokens required for its occurrence).
  We denote by $\text{Jin}(e)$ the multi-set of timed tokens required for its occurrence and by $EE(\text{TM})$ the set of all events of TM.

In this model, an event shall occur as soon as possible (i.e.: when all required tokens become available). Its firing takes no time but may lead to a new marking: consumed tokens disappear and possibly new tokens are created. Note that an event will never occur in case a conflicting event is fired or the progression of time is blocked (because there is an infinite firing sequence which takes no time). In the first case, the event is disabled (no enough tokens), while in the second case, some of its tokens will never become available.

Definition 2.5 time progression and event occurrence
Let TM be a reachable state of an *ITCPN* model, e_f an event of TM and dv a non-negative real number.

- The occurrence delay (i.e.: the firing delay) of e_f, denoted by $\text{FD}(e_f)$, is the delay required for all tokens of $\text{Jin}(e_f)$ to become available, i.e.: $\text{FD}(e_f) = \max_{(p_f, c_f, v_f) \in \text{Jin}(e_f)}(v_f)$.

- A time progression of dv units can occur from the state TM (without any firing) if and only if, dv is less or equal to the occurrence delays of all events, i.e.: $dv \leqslant \min_{e_i \in EE(\mathrm{TM})}(\mathrm{FD}(e_i))$.
  After this time progression, the delay of each token (p, c, v) of TM decreases by $min(dv, v)$ time units (delays cannot be negative). Its value becomes $\max(v - dv, 0)$. We denote by $\lfloor \mathrm{TM} \rfloor_{-dv}$ the obtained timed marking, i.e.: $\lfloor \mathrm{TM} \rfloor_{-dv} = \sum_{(p,c,v) \in \mathrm{TM}} \mathrm{TM}((p, c, v)) \bullet (p, c, \max(v - dv, 0))$.
- Event e_f can occur from TM before other events if and only if, its occurrence delay is not greater than those of other events, i.e.: $(\mathrm{FD}(e_f) \leqslant \min_{e_i \in EE(\mathrm{TM})}(\mathrm{FD}(e_i)))$.
- If e_f can occur from TM, it occurs instantaneously exactly after $\mathrm{FD}(e_f)$ time units. Its occurrence leads to a state TM′ such that: $\lfloor \mathrm{TM} - \mathrm{Jin}(e_f) \rfloor_{-\mathrm{FD}(e_f)} + \mathrm{Jout}(e_f)$.
  Where $\mathrm{Jout}(e_f)$ is obtained from $F(t)(U(\mathrm{Jin}(e_f)))$ by replacing each time interval by any value chosen inside it. Note that the firing of event e_f will disable all events conflicting with it for TM. An event e_i of TM does not conflict with e_f iff $\mathrm{Jin}(e_f) + \mathrm{Jin}(e_i) \leqslant \mathrm{TM}$.
- An evolution of TM is a sequence of event occurrences and time progressions that can successively occur from TM. The evolutions of an *ITCPN* model are those of its initial timed marking.

Consider the model given in Fig. 2 and its initial state TM_0: $(p_{\mathrm{free}}, M_1, 2) + (p_{\mathrm{in}}, J_1, 2) + (p_{\mathrm{in}}, J_2, 2)$.
The initial state TM_0 has two conflicting events:
$e_1 = (t_1, (p_{\mathrm{free}}, M_1, 2) + (p_{\mathrm{in}}, J_1, 2))$ and $e_2 = (t_1, (p_{\mathrm{free}}, M_1, 2) + (p_{\mathrm{in}}, J_2, 2))$.
Their firing delays are: $\mathrm{FD}(e_1) = \max(2, 2)$ and $\mathrm{FD}(e_2) = \max(2, 2)$.
Both events can occur from the initial state after two time units but the firing of one of them will disable the other.
The state reached after two time units is:
$(p_{\mathrm{free}}, M_1, \max(0, 2 - 2)) + (p_{\mathrm{in}}, J_1, \max(0, 2 - 2)) + (p_{\mathrm{in}}, J_2, \max(0, 2 - 2))$.
The occurrence of event e_1 leads to a state TM_1 such that:
$\mathrm{TM}_1 = (p_{\mathrm{in}}, J_2, 0) + (p_{\mathrm{busy}}, (M_1, J_1), v)$ and $v \in [1, 3]$.
The occurrence of event e_2 leads to a state TM_2 such that:
$\mathrm{TM}_2 = (p_{\mathrm{in}}, J_1, 0) + (p_{\mathrm{busy}}, (M_1, J_2), v)$ and $v \in [1, 3]$.

Because of time density, the state space of the *ITCPN* model is generally infinite and then not useful for enumerative analysis methods. For these methods, we need an extra step for contracting its infinite state space into a finite graph preserving properties of interest.

3. Van der Aalst's contraction approach

Van der Aalst proposed in [Van93] a contraction method which is "sound" but not "complete". This is due to the fact that this method "forgets" the occurrence time to memorize only intervals, i.e.: a state class (a set of states) is defined as a multi-set of triplets of the form (place, color, interval). Consequently, for event producing several tokens, the dependencies (relations binding intervals) are lost and resulting classes may contain unreachable states (leading sometimes to unreachable markings). For instance, consider the model shown in Fig. 3. We suppose that the color domain of each place is $\{e\}$, the initial state is $(p_0, e, 0)$ and transition functions are defined as follows:
$F(t_0)((p_0, e)) = (p_1, e, [0, 2])$; $F(t_1)((p_1, e)) = (p_2, e, [1, 2]) + (p_3, e, [1, 3])$;
$F(t_2)((p_2, e)) = 0$ and $F(t_3)((p_3, e)) = 0$.

Using the Van der Aalst's method, the firing of transition t_0 leads to the state class $(p_1, e, [0, 2])$. The firing of t_1 from this class produces the class $(p_2, e, [1 + 0, 2 + 2]) + (p_3, e, [3 + 0, 4 + 2])$ where states $(p_2, e, 1) + (p_3, e, 6)$ and $(p_2, e, 4) + (p_3, e, 3)$ are represented but not reachable. From the former state, transition t_2 is fired before t_3 $(1 < 6)$ while from the second one, transition t_3 is fired before t_2 $(3 < 4)$. Hence, Van der Aalst's method states that both markings (p_2, e) and (p_3, e) are reachable which is in fact wrong. The reason is that after firing transition t_1, the created token (p_2, e) becomes available before token (p_3, e) ($[1, 2] < [3, 4]$). Therefore, transition t_3 cannot be fired before t_2. Marking (p_2, e) is then not reachable. Due to these represented but unreachable states (markings), the resulting graph has not necessarily the same properties as the initial model.

Moreover, for some bounded ITCPNs allowing infinite firing sequences, *Van der Aalst*'s method produces infinite graphs. As an example, consider the model shown in Fig. 4. We suppose that the color domain of each place is $\{e\}$, the initial state is (p_1, e), $F(t_1)(p_1, e) = (p_1, e, [1, 2]) + (p_2, e, [0, 1])$, and $F(t_2)(p_2, e) = 0$.
Using the Van der Aalst's approach, the initial state class is $\alpha_0 = (p_1, e, [1, 2])$. From α_0, event $(t_1, (p_1, e, [1, 2]))$ may occur at any date inside [1,2]. Its occurrence leads to the state class $\alpha_1 = (p_1, e, [1+1, 2+2]) + (p_2, e, [0+1, 1+2])$.
Both events $(t_1, (p_1, e, [2, 4]))$ and $(t_2, (p_2, e, [1, 3]))$ may occur respectively at dates inside intervals $[2, min(3, 4)]$

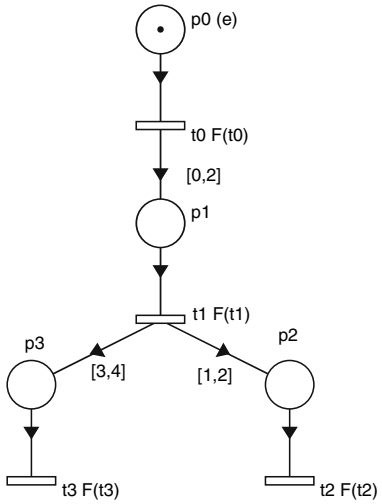

Fig. 3. *ITCPN* model used to explain Van der Aalst's approach

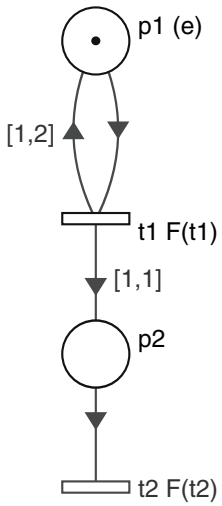

Fig. 4. Bounded *ITCPN* but unbounded underlying *CPN*

and $[1, min(3, 4)]$. The occurrence of event $(t_2, (p_2, mess, [1, 3]))$ from α_1 leads to the class $\alpha_2 = (p_1, e, [2, 4])$ which has the same marking as the initial state class but the interval of its token is different. The repetitive firing of the sequence $t_1 t_2$ will generate an infinite number of state classes. All these state classes share the same marking but their time intervals are different. Hence, the graph of state classes obtained using the Van der Aalst's approach is infinite.

We propose, in the following, another approach which is both "sound" and "complete". As we will show, our approach generates finite graphs for all bounded *ITCPNs*.

4. Our contraction approach

To contract the *ITCPN* state space, we first agglomerate, into one state class, all states reachable by firing the same sequence of events independently of their occurrence dates.

4.1. Characterization and computation of state classes

Let S be a sequence of events that can occur from the initial state of an *ITCPN* model (i.e.: S is a firable sequence).

- We denote by C_S the set of all states reachable by firing, from the initial state, the same sequence S, independently of instants at which events of the sequence have occurred in the past.
- By convention, if the sequence S is empty, the class C_S is the initial state class that contains only the initial state (the initial timed marking of the model).
- The event set of a state class C_S is the union of event sets of all its states.
  We denote by E_S the set all events of C_S.

Definition 4.1 characterization of a state class
The state class C_S can be characterized by a pair $(\mathrm{SM}_S, \mathrm{FT}_S)$ where:

- SM_S is a timed marking such that delay values are replaced by delays (variables, one distinct variable per token). SM_S is the set of timed tokens present in places after firing the sequence S from the initial state of the model.
- FT_S is a logical formula which characterizes the delay valuations of all states agglomerated in the class C_S. Each valuation of delays corresponds to a state within the class.

As an example, the initial state class of the model shown in Fig. 2 is $C_\emptyset = (\mathrm{SM}_\emptyset, \mathrm{FT}_\emptyset)$ where:

- $\mathrm{SM}_\emptyset = (p_{\mathrm{free}}, M_1, d_1) + (p_{\mathrm{in}}, J_1, d_2) + (p_{\mathrm{in}}, J_2, d_3)$ and
- $\mathrm{FT}_\emptyset = (d_1 = 2 \wedge d_2 = 2 \wedge d_3 = 2)$.

The class $C_\emptyset$ has two events:
$e_1 = (t_1, (p_{\mathrm{free}}, M_1, d_1) + (p_{\mathrm{in}}, J_1, d_2))$ and $e_2 = (t_1, (p_{\mathrm{free}}, M_1, d_1) + (p_{\mathrm{in}}, J_2, d_3))$.

Definition 4.2 Let $C_S = (\mathrm{SM}_S, \mathrm{FT}_S)$ be a state class and e_f one of its events.

- Event e_f can occur from C_S if and only if, there is a state TM of C_S such that e_f can occur from it.
- If e_f can occur from C_S, its occurrence leads to the state class $C_{S.e_f}$ which contains all states reachable by firing e_f from any state of C_S.
- The set of evolutions of C_S is the union of all evolutions of its states.
- Note that events and functions $\mathrm{FD}_{\min}$, $\mathrm{FD}_{\max}$, Jin and Jout are defined as in Sect. 2.2.2 except that delay values are replaced by delay names.

Proposition 4.1 • Event e_f can occur from C_S if and only if, there is at least one state in the class for which the firing delay of event e_f is less or equal to the firing delays of all other events, i.e.: the following formula is consistent: $\mathrm{FT}_S \wedge (\bigwedge_{e_i \in E_S} (\mathrm{FD}(e_f) \leqslant \mathrm{FD}(e_i)))$.

- Suppose that event e_f can occur from the state class C_S. Its occurrence leads to the state class $C_{S.e_f} = (\mathrm{SM}_{S.e_f}, \mathrm{FT}_{S.e_f})$, where $\mathrm{SM}_{S.e_f} = \mathrm{SM}_S - \mathrm{Jin}(e_f) + \mathrm{Jout}(e_f)$ and $\mathrm{FT}_{S.e_f}$ is computed in five steps as follows:

  1. Initialize $\mathrm{FT}_{S.e_f}$ with $\mathrm{FT}_S \wedge (\mathrm{FD}(e_f) \leqslant dh) \wedge (\bigwedge_{e_i \in E_S} dh \leqslant \mathrm{FD}(e_i))$.
  2. Eliminate by substitution delays of all tokens consumed by event e_f (i.e.: tokens of $\mathrm{Jin}(e_f)$).
  3. Add for each delay d, a new variable $\underline{d}$ and the constraint $\underline{d} = \max(0, d - dh)$.
  4. Eliminate by substitution dh and all old delays d.
  5. Rename each delay $\underline{d}$ in d and add for each token created by e_f the constraint $a_d \leqslant d \leqslant b_d$, where d is the delay associated with the token and $[a_d, b_d]$ is its time stamp interval.

Proof. The firing condition is immediate from the definition. The computation of the successor class is done in five steps. Starting from the firing condition, we first introduce the variable dh representing the time progression before firing e_f and then we eliminate by substitution delays of tokens consumed by e_f. As delays decrease with time progression until reaching zero, we add a new variable $\underline{d}$ for each delay d which is equal to the remaining delay after dh time units, i.e.: $\underline{d} = \max(0, d - dh)$. Afterwards, we eliminate by substitution dh and all old variables d. At the last step, we rename each $\underline{d}$ in d and add the time constraints of all newly created tokens. Their delays may be any value inside their time stamp intervals. □

As an example, consider the initial state class $C_\emptyset$ of the model in Fig. 2 and its event $e_1 = (t_1, (p_{\text{free}}, M_1, d_1) + (p_{\text{in}}, J_1, d_2))$. Event e_1 can occur from the state class $C_\emptyset$ because the following formula is consistent:
$d_1 = 2 \wedge d_2 = 2 \wedge d_3 = 2 \wedge (\max(d_1, d_2) \leqslant \max(d_1, d_3)) \wedge (\max(d_1, d_2) \leqslant \max(d_1, d_2))$.
Its occurrence leads to the state class $C_{e_1} = (\text{SM}_{e_1}, \text{FT}_{e_1})$ such that $\text{SM}_{e_1} = (p_{\text{in}}, J_2, d_3) + (p_{\text{busy}}, (M_1, J_1), d_4)$ and FT_{e_1} is computed as follows:

1. Initialize FT_{e_1} with:
   $d_1 = 2 \wedge d_2 = 2 \wedge d_3 = 2 \wedge \max(d_1, d_2) \leqslant dh \wedge dh \leqslant \max(d_1, d_2) \wedge dh \leqslant \max(d_1, d_3)$.
2. Eliminate by substitution delays of all tokens consumed by event e_1 (i.e.: delays d_1 and d_2):
   $d_3 = 2 \wedge dh = 2$.
3. Add for each remaining delay d, a new variable $\underline{d}$ and the constraint $\underline{d} = \max(0, d - dh)$:
   $d_3 = 2 \wedge dh = 2 \wedge \underline{d_3} = \max(0, d_3 - dh)$.
4. Eliminate by substitution dh and all old delays d: $\underline{d_3} = 0$.
5. Rename each delay $\underline{d}$ in d and add for each token (p, c, d) of Jout(e_1), the constraint $a_d \leqslant d \leqslant b_d$: $(d_3 = 0 \wedge (1 \leqslant d_4 \leqslant 3))$.

The reachable state class by firing event e_1 from the class $C_\emptyset$ is $C_{e_1} = (\text{SM}_{e_1}, \text{FT}_{e_1})$ where:

- $\text{SM}_{e_1} = (p_{\text{in}}, J_2, d_3) + (p_{\text{busy}}, (M_1, J_1), d_4)$ and
- $\text{FT}_{e_1} = (d_3 = 0 \wedge 1 \leqslant d_4 \leqslant 3)$.

The firing rule established in Proposition 4.1. is not simple to implement since it requires resolution of a system of inequations. From the firing rule, we can deduce that the formula of each state class can be rewritten as a combination of connectors $\wedge$, $\vee$, and atomic constraints. Each atomic constraint has one of the following forms: $x - y \leqslant c$, $x - o \leqslant c$ or $o - x \leqslant c$, where x and y are delays (variables), c is a rational constant $c \in (Q \cup \{\infty\})$, and o is the symbol representing the value zero (o is always zero). Moreover, the computation of reachable state classes needs to test the consistency (domain emptiness) and equivalency of formulas (domain equality), and other operations like substitution and elimination of some variables. There are two well-known data structures that make simple the implementation of these operations: *Difference Bound Matrices DBMs* [Ben02, DOT96] and *Clock Difference Diagrams CDDs* [Ben02]. *DBMs* are useful to represent convex domains defined as a conjunction of atomic constraints, by giving for each pair of variables (clocks or delays) the upper bound of their difference. Almost operations on *DBMs* are made simple using their canonical forms[3] computed in $O(m^3)$, (m being the number of variables in the *DBM*). The computation of the canonical form is based on the shortest path *Floyd-Warshall*'s algorithm and is considered as the most costly operation on *DBMs*. However, knowing that *DBMs* are not closed under set-union, we may need several *DBMs* to represent formulas containing connector $\vee$ which makes operations more expensive. For some operations like the union-set, intersection and complementation, *CDDs* are more appropriate as they allow representing, in a compact way, such formulas while performing, with less complexity, these operations. But, some other operations needed to compute reachable state classes are very complex when we use *CDDs* [BBD02].

In order to reduce both space and time complexities, we show, in the following, that all state classes sharing the same marking and some delay function have also the same firing sequence. Therefore, they can be agglomerated and represented by their common marking and delay function. Moreover, only one *DBM* is needed for representation and computation of delay functions of reachable state classes.

4.2. More efficient characterization and computation of state classes

Let $C_S = (\text{SM}_S, \text{FT}_S)$ be a state class, $\aleph_S$ the set of all variables (delays) in SM_S and o the symbol representing the value 0). We define the delay function D_S as follows: $D_S : (\aleph_S \cup \{o\})^2 \to Q \cup \{\infty\}$,
$D_S(x, y) = \max(x - y \mid \text{FT}_S)$.[4]

The function $D_\emptyset$ of the initial class $C_\emptyset$ is defined by: $\forall (x, y) \in (\aleph_\emptyset \cup \{o\})^2$, $D_\emptyset(x, y) = \begin{cases} 0 & \text{if } x = y; \\ b_x - a_y & \text{otherwise} \end{cases}$

Where $a_o = 0$, $b_o = 0$ and $[a_d, b_d]$, for $d \in \aleph_\emptyset$, is the time stamp interval of the token associated with d.

[3] A canonical form of a *DBM* is the representation with tightest bounds on all variable (clock or delay) differences.
[4] The biggest value of $x - y$ in the domain represented by FT_S.

As an example, consider the initial state class $C_\emptyset = (\mathrm{SM}_\emptyset, \mathrm{FT}_\emptyset)$ of the model in Fig. 2:

- $\mathrm{SM}_\emptyset = (p_{\mathrm{free}}, M_1, d_1) + (p_{\mathrm{in}}, J_1, d_2) + (p_{\mathrm{in}}, J_2, d_3)$ and
- $\mathrm{FT}_\emptyset = (d_1 = 2 \wedge d_2 = 2 \wedge d_3 = 2)$.

Its function delay $D_\emptyset$ is defined over the set $\{o, d_1, d_2, d_3\}$ as follows:

$D_\emptyset$	o	d_1	d_2	d_3
o	0	−2	−2	−2
d_1	2	0	0	0
d_2	2	0	0	0
d_3	2	0	0	0

Proposition 4.2 Let e_f be an event of C_S, i.e.: $e_f \in E_S$. Event e_f can occur from the state class C_S if and only if: $\min_{e_i \in E(S)} \max_{(p_i,c_i,d_i)\in \mathrm{Jin}(e_i)} \min_{(p_f,c_f,d_f)\in \mathrm{Jin}(e_f)} D_S(d_i, d_f) \geqslant 0$.

Proof. Proposition 4.1 states that event e_f can occur from the state class C_S if and only if the following formula is consistent: (1) $\mathrm{FT}_S \wedge (\bigwedge_{e_i \in E_S} (\mathrm{FD}(e_f) \leqslant \mathrm{FD}(e_i)))$.
After developing FD, we can rewrite it to obtain: $\mathrm{FT}_S \wedge \min_{e_i \in E_S} \max_{(p_i,c_i,d_i)\in \mathrm{Jin}(e_i)} \min_{(p_f,c_f,d_f)\in \mathrm{Jin}(e_f)} d_i - d_f \geqslant 0$.
By assumption, FT_S is consistent and $D_S(d_i, d_f)$ is the biggest value of $d_i - d_f$ in the domain of FT_S. Moreover, since delays decrease synchronously with time, the following relations hold: $\forall\, x, y, z \in \aleph$,
$(D_S(x, y) \leqslant D_S(z, y) \Longrightarrow \forall\, u \in \aleph, D_S(x, u) \leqslant D_S(z, u))$ and
$(D_S(x, y) \leqslant D_S(x, z) \Longrightarrow \forall\, u \in \aleph, D_S(u, y) \leqslant D_S(u, z))$.
It follows that there exists a token $(p_{f_f}, c_{f_f}, d_{f_f}) \in \mathrm{Jin}(e_f)$ such that:
$\forall\, d \in \aleph, D_S(d, d_{f_f}) = \min_{(p_f,c_f,d_f)\in \mathrm{Jin}(e_f)} (D_S(d, d_f))$
Therefore there exist an event $e_{i_i} \in E_S$ and a token $(p_{i_i}, c_{i_i}, d_{i_i}) \in \mathrm{Jin}(e_{i_i})$ such that $D_S(d_{i_i}, d_{f_f}) = \min_{e_i \in E_S} \max_{(p_i,c_i,d_i)\in \mathrm{Jin}(e_i)} \min_{(p_f,c_f,d_f)\in \mathrm{Jin}(e_f)} (D_S(d_i, d_f))$.
It follows that the firing condition is satisfied if and only if:
$\min_{e_i \in E_S} \max_{(p_i,c_i,d_i)\in \mathrm{Jin}(e_i)} \min_{(p_f,c_f,d_f)\in \mathrm{Jin}(e_f)} (D_S(d_i, d_f)) \geqslant 0$. □

We have shown that the firing condition from a state class C_S can be simplified to be expressed by means of the marking and the delay function of C_S. The following proposition shows how to compute using the delay function of C_S, the delay function of any successor class of C_S.

Proposition 4.3 Let S be a firable sequence, e_f an event which can occur from the state class C_S and S' the event sequence $(S.e_f)$. The function $D_{S'}$ can be computed using D_S as follows:

- $\forall\, d \in \aleph_{S'}$,
  - $D_{S'}(d, d) = 0$.
  - $D_{S'}(o, d) = \begin{cases} -a_d & \text{if } d \text{ is created by } e_f; \\ \min(0, \min_{e_i \in E_S} \max_{(p_i,c_i,d_i)\in \mathrm{Jin}(e)} D_S(d_i, d)) & \text{otherwise} \end{cases}$
  - $D_{S'}(d, o) = \begin{cases} b_d & \text{if } d \text{ is created by } e_f; \\ \max(0, \min_{(p_f,c_f,d_f)\in \mathrm{Jin}(e_f)} D_S(d, d_f)) & \text{otherwise} \end{cases}$
- $\forall (d, d') \in \aleph_{S'}^2, d \neq d'$,
  - $D_{S'}(d, d') = \begin{cases} \min(D_S(d, d'), D_{S'}(d, o) + D_{S'}(o, d')) & \text{if } d \text{ and } d' \text{ are not created by } e_f; \\ D_{S'}(d, o) + D_{S'}(o, d') & \text{otherwise} \end{cases}$

Proof. Recall the definition of $D_{S'}$: $\forall (x, y) \in (\aleph_{S'} \cup \{o\})^2$, $D_{S'}(x, y) = \max(x - y \mid \mathrm{FT}_{S'})$.
(1) Then: $\forall\, x \in \aleph_{S'} \cup \{o\}$, $D_{S'}(x, x) = 0$;
(2) Computing $D_{S'}(d, o)$ and $D_{S'}(o, d)$, for $d \in \aleph_{S'}$:

- In case d is associated with some token created by event e_f, its domain is the time interval $[a_d, b_d]$. Hence, $D_{S'}(d, o) = \max(d \mid \mathrm{FT}_{S'}) = b_d$ and $D_{S'}(o, d) = \max(-d \mid \mathrm{FT}_{S'}) = -a_d$.

- In case d is associated with some token not created by e_f, $D_{S'}(d, o)$ and $D_{S'}(o, d)$ are respectively the biggest values of $\max(0, d - dh)$ and $-\max(0, d - dh)$ (i.e.: the biggest value of $\min(0, dh - d)$), satisfying the firing condition of e_f, i.e.: $\mathrm{FT}_S \wedge \mathrm{FD}_{\min}(e_f) \leqslant dh \leqslant \min_{e_i \in E_S} \mathrm{FD}_{\max}(e_i)$.
  After developing FD, we can rewrite this firing condition to obtain:
  $(\mathrm{FT}_S \wedge dh \leqslant \min_{e_i \in E_S} \max_{(p_i,c_i,d_i)\in \mathrm{Jin}(e_i)} (d_i) \wedge -dh \leqslant \min_{(p_f,c_f,d_f)\in \mathrm{Jin}(e_f)} (-d_f))$.
  The biggest values of $d - dh$ and $dh - d$ satisfying the firing condition of e_f are respectively equal to the biggest values of $d - dh$ and $dh - d$ satisfying the following formula:
  $(\mathrm{FT}_S \wedge dh - d \leqslant \min_{e_i \in E_S} \max_{(p_i,c_i,d_i)\in \mathrm{Jin}(e_i)} (d_i - d) \wedge d - dh \leqslant \min_{(p_f,c_f,d_f)\in \mathrm{Jin}(e_f)} (d - d_f))$.
  We obtain these values by replacing each occurrence $d - d_f$ and $d_i - d$, in the second and the third terms of the formula, by their biggest values in the domain of FT_S (i.e.: $D_S(d, d_f)$ and $D_S(d_i, d)$). It comes that the biggest values of $\max(0, d - dh)$ and $min(0, dh - d)$), satisfying the firing condition of e_f are respectively:
  $D_{S'}(d, o) = \max(0, \min_{(p_f,c_f,d_f)\in \mathrm{Jin}(e_f)} D_S(d, d_f))$ and
  $D_{S'}(o, d) = min(0, \min_{e_i \in E_S} \max_{(p_i,c_i,d_i)\in \mathrm{Jin}(e_i)} D_S(d_i, d))$.

(3) Computing $D_{S'}(d, d')$, for $(d, d') \in \aleph^2_{S'}$. We have $D_{S'}(d, d') = \max(d - d' \mid \mathrm{FT}_{S'})$:

- In case d or d' is associated with some token created by e_f, there is no constraint in the firing condition of e_f on the difference $d - d'$ but the biggest values of d and $-d'$ in $\mathrm{FT}_{S'}$ are respectively $D_{S'}(d, o)$ and $D_{S'}(o, d')$. Therefore, $D_{S'}(d, d') = D_{S'}(d, o) + D_{S'}(o, d')$.
- In case d and d' are associated with tokens not created by e_f, $D_{S'}(d, d')$ is equal to $\max(d - d' \mid \mathrm{FT}_{S'})$. $D_{S'}(d, d')$ is also the biggest value of $\max(0, d - dh) - \max(0, d' - dh)$ satisfying the firing condition of e_f, i.e.: $(\mathrm{FT}_S \wedge \max(d - dh, 0) - \max(0, d' - dh) \leqslant$
  $\max(0, \min_{(p_f,c_f,d_f)\in \mathrm{Jin}(e_f)} (d - d_f)) + \min(0, \min_{e_i \in E_S} \max_{(p_i,c_i,d_i)\in \mathrm{Jin}(e_i)} (d_i - d')))$.

The biggest value of $\max(0, d - dh) - \max(0, d' - dh)$ satisfying the firing condition of e_f is then:
$\min(D_S(d, d'), D_{S'}(d, o) + D_{S'}(o, d'))$. □

From the previous propositions, it comes that state classes sharing the same marking and the same delay function have exactly the same firing sequences. Therefore, they can be agglomerated into one node without loosing linear properties of the model. The agglomerated state classes are then represented by their common marking and delay function. The graph of all reachable state classes is obtained by applying the firing rule established in Propositions 4.2. and 4.3. to the initial class represented by $(\mathrm{SM}_\emptyset, D_\emptyset)$ and to each new class. In this way, all state classes sharing the same marking and the same delay function are agglomerated in the same node. The following theorem establishes one necessary and sufficient condition to obtain a finite state class graph. The proof of this theorem is based on the following proposition shown in [BeD91].

Proposition 4.4 Let Y be a finite linear combination i.e.: $Y = n_1 \times y_1 + n_2 \times y_2 + \cdots + n_r \times y_r$ where $n_1, n_2, \ldots, n_r$ are integer coefficients and $y_1, y_2, \ldots, y_r$ are rational constants. If Y is bounded by finite rational constants (i.e.: $a \leqslant Y \leqslant b$) then the number of different linear combinations Y is finite. In other words, the value domain of Y is finite.

Theorem 4.1 An *ITCPN* has a finite state class graph if and only if, it is bounded (it has a finite number of reachable markings).

Proof. $\Rightarrow$) is obvious.
$\Leftarrow$) If the model is bounded, it has a finite set of reachable markings. Since the number of different markings is finite, it suffices to prove that for each marking, we have a finite number of different classes that share the same marking. Consider a marking and a firable sequence S which leads to the considered marking. We have to show that there is a finite number of different functions D_S. For each pair of delays (d, d') of $\aleph^2_S$, terms $D_S(d, d')$, $D_S(o, d')$ and $D_S(d, o)$ are finite combinations of rational constants with integer coefficients (finite combinations because the number of different time intervals in the model is finite). These terms are bounded or equal to ∞:

- $D_S(d, o)$ is initialized to b_d. If $b_d = \infty$ then the value of $D_S(d, o)$ is ∞ and does not change with time. Otherwise, the value of $D_S(d, o)$ decreases with time until reaching the value zero. Then:
  $0 \leqslant D_S(d, o) \leqslant b_d \neq \infty$ or $D_S(d, o) = \infty$.
- $D_S(o, d')$ is initialized to $-a_{d'}$. Its value increases until reaching the value zero. Then:
  $-a_{d'} \leqslant D_S(o, d') \leqslant 0$.
- $-a_{d'} \leqslant D_S(d, d') \leqslant b_d \neq \infty$ or $D_S(d, d') = \infty$.

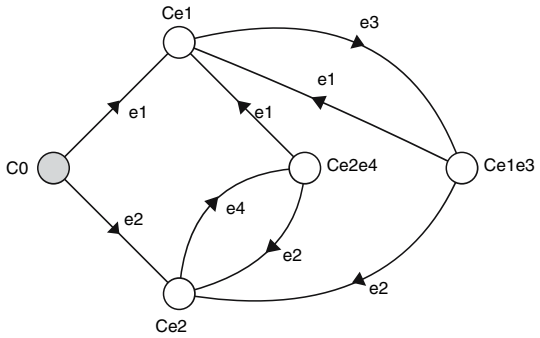

Fig. 5. The state class graph of the *ITCPN* model shown in Fig. 2

From Proposition 4.4, the value domains of $D_S(d, o)$, $D_S(o, d')$ and $D_S(d, d')$ are finite. Consequently, the number of different D_S is finite. □

This necessary and sufficient condition that ensures a finite state class graph may be difficult to use since we have not a general procedure to decide whether or not an *ITCPN* has a finite number of different markings. However, we have a straightforward sufficient condition using the underlying coloured Petri net (*CPN*), and we know several methods to decide this property on *CPN*, namely the invariant method: An *ITCPN* has a finite number of markings (i.e.: bounded), if its underlying *CPN* has a finite number of different markings (i.e.: bounded). The reverse is not true. Indeed, an *ITCPN* can have a finite set of reachable markings but its underlying *CPN* has an infinite number of reachable markings As an example the model shown in Fig. 4 has three reachable markings: M_0, $M_1 = (p_1, e) + (p_2, e)$ and $M_2 = (p_1, e) + 2(p_2, e)$, but its underlying coloured Petri net is unbounded (place p_2 is unbounded).

For example, applying our approach to the model given in Fig. 2 produces the state class graph shown in Fig. 5. It consists of 5 state classes, 8 edges and 4 events:

- $C_{\emptyset} = ((p_{\text{free}}, M_1, d_1) + (p_{\text{in}}, J_1, d_2) + (p_{\text{in}}, J_2, d_3), (d_1 = 2 \wedge d_2 = 2 \wedge d_3 = 2))$
- $C_{e_1} = ((p_{\text{busy}}, (M_1, J_1), d_4) + (p_{\text{in}}, J_2, d_3), (d_3 = 0 \wedge 1 \leqslant d_4 \leqslant 3))$
- $C_{e_1e_3} = ((p_{\text{free}}, M_1, d_1) + (p_{\text{in}}, J_1, d_2) + (p_{\text{in}}, J_2, d_3), (d_1 = 2 \wedge d_2 = 2 \wedge d_3 = 0))$
- $C_{e_2} = ((p_{\text{busy}}, (M_1, J_2), d_5) + (p_{\text{in}}, J_1, d_2), (d_2 = 0 \wedge 1 \leqslant d_5 \leqslant 3))$
- $C_{e_2e_4} = ((p_{\text{free}}, M_1, d_1) + (p_{\text{in}}, J_1, d_2) + (p_{\text{in}}, J_2, d_3), (d_1 = 2 \wedge d_3 = 2 \wedge d_2 = 0))$
- $e_1 = ((p_{\text{free}}, M_1, d_1) + (p_{\text{in}}, J_1, d_2), t_1)$
- $e_2 = ((p_{\text{free}}, M_1, d_1) + (p_{\text{in}}, J_2, d_3), t_1)$
- $e_3 = ((p_{\text{busy}}, (M_1, J_1), d_4), t_2)$
- $e_4 = ((p_{\text{busy}}, (M_1, J_2), d_5), t_2)$

4.3. Using state class graphs to verify the *ITCPN* properties

The state class graph of an *ITCPN*, obtained using our approach, preserves linear properties of the model. So, if it is finite, it can be used to verify linear properties of the model. Untimed linear properties can be checked on the graph using the classical linear model checking techniques [ACD93, HHW97].

For timed properties, a variety of real time extensions of linear time logic (LTL) have been proposed for expressing requirements of real time systems. Among these extensions, we consider the metric interval temporal logic (MITL) which extends LTL by associating a time interval with temporal operators (always G, eventually F, and until U). The verification of these properties can be performed using a technique similar to the one developed in [AlD90]. This technique consists in two steps. The first step constructs some timed *Buchi* automaton for the negation of the property to be checked. The second step computes the synchronous product of the state space of the model (or a contraction preserving its linear properties) with the timed *Buchi* automaton constructed in the first step. The property is satisfied if and only if the synchronous product is empty. To apply this technique, we suppose that the model is *zenon* free (there is no infinite firing sequence with execution time equal to 0)

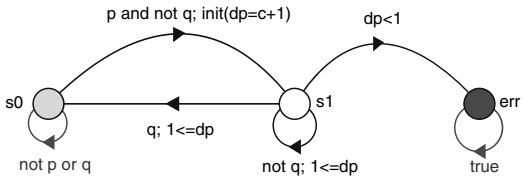

Fig. 6. The timed Buchi automaton of $\neg(G(p \Longrightarrow F_{[0,c]}q))$

[AlD90, DOT96]. In [AlD90], clock constraints and guards of the timed *Buchi* automaton are used to express time requirements and atomic propositions of the property to be checked. In our case, we use delays instead of clocks. Note that using delays instead of clocks simplifies the computation of reachable state classes as no over-approximation operation is needed [Ben02].

As an example, we consider the important bounded response requirement of real time systems. This requirement is expressed by the *MITL* formula as follows: $G(p => F_{[0,c]}q)$ which means that request p must be followed by a response q within c time units. The timed *Buchi* automaton of the negation of this property is shown in Fig. 6, where s_0 is the initial node and *err* is an acceptation node. The delay dp and the constraint $dp = c+1$ are added to the current state class if it satisfies the formula *p and not q* and the edge (s_0, s_1) has to be fired synchronously with an event of the model. The property q must hold before or when dp reaches the value 1. Otherwise, the node *err* is reachable. We suppose that the ITCPN model is not zeno. Zenoness is a pathological situation which suggests that infinity of actions may take place in a finite amount of time.

Consider the state class graph shown in Fig. 5 and the property $G((pin, J_1) \Longrightarrow F_{[0,4]}(pbusy, (M_1, J_1)))$. This property means whenever the job J_1 is waiting for execution, it must be in execution within 4 time units. To verify this property, we have to construct the synchronous product of the timed *buchi* automaton of the property and the state class graph shown in Fig. 5. Its construction is essentially based on propositions 4.2 and 4.3. The part of the resulting graph, shown in Fig. 7, exhibits that the synchronous product is not empty. The property is not satisfied since there exists a cycle which passes over the acceptation node (*err*).
Starting from the initial summit of the synchronous product $((SM_0, s_0), FT_0)$, successors summits can be computed iteratively using the following rule:
Let ((SM, s), FT) be a summit of the synchronous product, e_f an event of SM, and $(s, s', gp, init, gd)$ be an outgoing edge of s, where gp is a guard over SM, *init*[5] is the constraint to be added to FT in case SM satisfies the guard gp, and gd is the constraint to be added to the successor summit resulting from the synchronous firing of e_f with $(s, s', gp, init, gd)$. Event e_f may occur synchronously with edge $(s, s', gp, init, gd)$ from ((SM, s), FT) if and only if:

1 SM satisfies the guard gp,
2 e_f is firable from (SM, FT $\wedge$ *init*). Let (SM′, FT′) be the resulting successor state class.
3 The formula (FT′ $\wedge$ gd) is consistent.

In this case, e_f may occur synchronously with edge $(s, s', gp, init, gd)$ from ((SM, s), FT), its occurrence leads to the summit ((SM′, s'), FT′ $\wedge$ gd).

Note that the implementation of the above firing rule is in fact based on propositions 4.2 and 4.3. In addition, the synchronous product can be computed directly with no need to construct the state class graph. To attenuate the state explosion problem, it can be also computed using an *on-the-fly* method augmented with an abstraction by inclusion (as shown in [Ben02]).

5. Conclusion

We proposed an efficient approach for contracting the generally infinite state space of the *ITCPN* model. Our contraction approach generates finite graphs for all bounded *ITCPNs* and preserves linear properties of the *ITCPNs*. So, the generated graphs are useful for *LTL* model checking.

We have first extended the *ITCPN* model by allowing unbounded intervals. This extension allows expressing that a created token could be never available (be lost) but complicates somewhat its enumerative analysis

[5] Delay variables in constraint *init* are all different from those in FT.

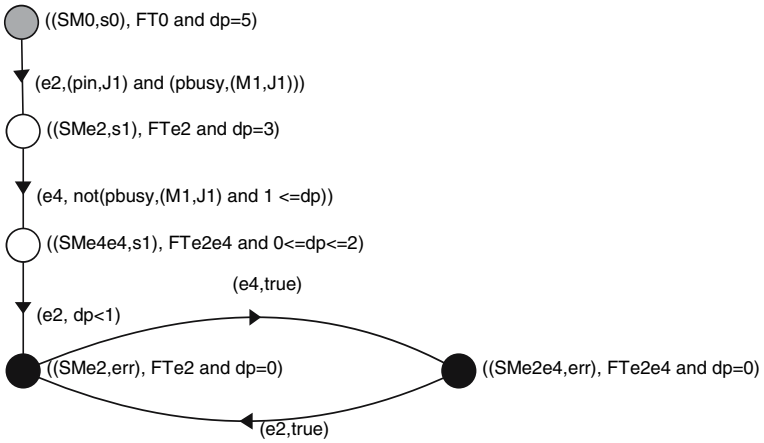

Fig. 7. A part of the synchronous product of Fig. 5 and Fig. 6

(unbounded delays). To contract the *ITCPN* state space, we first agglomerated, into one state class, all states reachable by the same firing sequence. The resulting state classes can be characterized by means of a marking and a formula combining atomic constraints and connectors $\vee$ and $\wedge$. There are two well-known data structures that make simple the representation and implementation of operations needed to compute reachable state classes: *Difference Bound Matrices DBMs* [Ben02, DOT96] and *Clock Difference Diagrams CDDs* [Ben02]. However, knowing that *DBMs* are not closed under set-union, we may need several *DBMs* to represent formulas containing connector $\vee$ which makes operations more expensive. *CDDs* are more appropriate as they allow representing, in a compact way, such formulas. But, some operations needed to compute reachable state classes are very complex when we use *CDDs* [Ben02].

In order to reduce both space and time complexities, we proposed an relation of equivalence over state classes which induces an attractive characterization and computation procedure of state classes. Indeed, only one *DBM* is needed for the representation and computation of time constraints of reachable state classes.

Our approach is more efficient than those developed in the literature [BoB02, BeB94, Van93]. Indeed, the approach developed in [Van93] is not complete and may generate infinite graphs for some bounded models while our approach has not these drawbacks. In addition, our approach is more simple than those developed in [BoB02, BeB94]. The characterization and computation of state classes are both simplified.

Finally, we have shown by means of an example how to use our approach to verify timed linear properties of the model. To attenuate the state explosion problem, this verification can be performed using an on-the-fly method augmented with an abstraction by inclusion.

References

[AlD90] Alur R, Dill D (1990) Automata for modeling real-time systems. 17me ICALP, LNCS 443, Springer, Heidelberg, pp 322–335

[ACD93] Alur R, Courcoubetis C, Dill D (1993) Model checking in dense real-time. Inf Comput 104(1):2–34

[Ben02] Bengtsson J (2002) Clocks, DBMs and states in timed systems. PhD thesis, Department of Information Technology, Uppsala University

[BBD02] Behrmann G, Bengtsson J, David A, Larsen KG, Pettersson P, Yi W (2002) UPPAAL implementation secrets. In: 7th international symposium on formal techniques in real-time and fault-tolerant systems, LNCS 2469. Springer, Heidelberg, pp 3–22

[BeD91] Berthomieu B, Diaz M (1991) Modeling and verification of time dependent systems using time Petri nets. IEEE Trans Softw Eng 17(33):259–273, 275–290

[BoB02] Boucheneb H, Berthelot G (2002) Contraction of the ITCPN state space. Theor Comput Sci 65(6):1–15

[BeB94] Berthelot G, Boucheneb H (1994) Occurrence graphs for interval timed coloured nets. In: 15th international conference on application and theory of Petri nets, Zaragoza (Spain), LNCS 815. Springer, Heidelberg

[CKM01] Christensen S, Kritensen LM, Mailand T (2001) Condensed state spaces for timed Petri nets. In: 22nd international conference on application and theory of Petri nets and 2nd international conference on application of concurrency to system design, Newcastle Upon Tyne

[DOT96] Daws C, Olivero A, Tripakis S, Yovine S (1996) The tool Kronos. Hybrid Systems III, Verification and Control, LNCS 1066. Springer, Heidelberg

[DiS94] Diaz M, Senac P (1994) Time stream Petri nets a model for timed multimedia information. In: 15th international conference on application and theory of Petri nets, LNCS 815. Springer, Zaragoza (Spain)

[HHW97] Henzinger TA, Ho PH, Wong-Toi H (1997) HyTech: a model checker for hybrid systems. Softw Tools Technol Transf 1:110–122
[HsG02] Hsiung PA, Gau CH (2002) Formal synthesis of real-time embedded software by time-memory scheduling of colored time Petri nets. Theor Comput Sci 65(6):140–153
[Jen82] Jensen K (1982) Coloured Petri nets: basic concepts, analysis methods and practical use, vols 1 and 2, EATCS Monographs on Theoretical Computer Science. Springer, Heidelberg
[MeF76] Merlin P, Farber DJ (1976) Recoverability of communication protocols. IEEE Trans Commun 24: 1036–1042
[Sif77] Sifakis J (1977) In: Beilner H, Gelenbe E (eds) Use of Petri nets for performance evaluation, measuring, modeling and evaluating computer systems. North-Holland, Amsterdam
[TKP02] Thanh CB, Klaudel H, Pommereau F (2002) Petri nets with causal time for system verification. In: 3rd international workshop on models for time-critical systems
[Van93] Van der Aalst WMP (1993) Interval timed coloured Petri nets and their analysis. In: 14th international conference of application and theory of Petri nets, Chicago
[Vic01] Vicaro E (2001) Static analysis and dynamic steering of time-dependent systems. IEEE Trans Softw Eng 2(8):728–748

Received 15 April 2004
Revised 29 October 2006
Accepted 13 August 2007 by C. B. Jones
Published online 23 October 2007