High-level Colored Time Petri Nets for true concurrency modeling in real-time software

Imane Haur, Jean-Luc Béchennec, Olivier Roux

High-level Colored Time Petri Nets for true concurrency modeling in real-time software

1st Imane HAUR
École centrale de Nantes, LS2N
Huawei, Paris Research Center
Nantes, Paris, France
imane.haur@ec-nantes.fr

2nd Jean-Luc BÉCHENNEC
CNRS
LS2N
Nantes, France
jean-luc.bechennec@ls2n.fr

3rd Olivier H. ROUX
École centrale de Nantes
LS2N
Nantes, France
olivier-h.roux@ec-nantes.fr

***Abstract*—The control of real-time systems often requires taking into account simultaneous access in true parallelism to shared resources. This is particularly the case for multi-core execution platforms. Timed automata or time Petri nets do not capture these features directly. We propose extending time Petri Nets with color and high-level functionality encompassing both timed multi-enableness of transitions and sequential pseudo code. We prove that the reachability problem is decidable for this model on which an on-the-fly TCTL model checking algorithm is efficiently implemented in the tool ROMÉO. We apply this approach to modeling a multi-core real time spinlock mechanism in order to check all possible execution paths and interleaving of service calls.**

***Index Terms*—Multi-core execution, High-level Colored Time Petri Nets, Model-checking**

I. INTRODUCTION

For the lack of data structures, Petri nets are unsuitable for modeling systems where data affects the system's behavior. High-level Petri nets [1] have been proposed for modeling scientific problems with complex structures allowing the description of both system data and control. The term High-level Petri net is then used for many Petri nets [2] such as Predicate/-Transition Nets, colored Petri nets, or hierarchical Petri nets. However, the common point is that they allow manipulating different types of expressions that use state variables. Input arcs are labeled with boolean expressions specifying conditions (guards or gates) that can also be associated with transitions. Arc annotations are expressions that can be associated with output arc. They can be viewed as computing systems that operate on shared data.

*1) **Petri nets**:* Petri nets are one of the many modeling languages used to describe distributed concurrent systems. A place can contain any number of tokens. A marking M of a Petri Net is a vector representing the number of tokens of each place. A transition is enabled (it may fire) in M if there are enough tokens in its input places for the consumptions to be possible. Firing a transition t in a marking M consumes one token from each of its input places s, and produces one token in each of its output places s.

*2) **High-level Petri nets**:* The precondition (guard) and postcondition (update) over a set of variables (X) are associated with transitions. A transition is enabled (it may fire) if there are enough tokens in its input places and if the guard is true. When the transition fires, the corresponding updates are executed, modifying the values of the variables. The variables take their values in a finite state (such as bounded integer or enumerate type...), guards are boolean expressions over X, and updates can be described as a sequence of imperative code expressed in a programming language but whose execution is atomic from the transition firing point of view.

*3) **Colored Petri nets**:* The colored extension of Petri nets allows the distinction between tokens.

Although the set X of High-level Petri nets presented in the previous paragraph can be of arbitrarily complex type, places in colored Petri nets contain tokens of one type. This type noted C is called the color set of the place.

An arc from a place to a transition (PT) specifies the color(s) that enabled the transition, and its firing will consume it. An arc from a transition to a place (TP) specifies the token color produced in that place by the firing of the transition. A particular color called *any* indicates in a PT arc that any color enabled the transition, and in a TP arc that the color consumed in the input place will be the one produced in the output place.

A marking M of a colored Petri Net represents not only the number of tokens in each place but also their respective colors. That is represented either by a multiset or by a matrix.

*4) **Time Petri Nets**:* Time Petri nets (TPN) extend Petri nets with temporal intervals associated with transitions, specifying firing delay ranges for the transitions. Assuming transition t became last enabled at time d and the endpoints of its firing interval are α and β, then t cannot fire earlier than $d+\alpha$ and must fire no later than $d+\beta$ unless disabled by the firing of another transition. Firing a transition takes no time.

*5) **Colored Time Petri Nets**:* For real parallelism or with interleaving semantics of timed systems, the notion of multiple enableness is needed. It refers to the fact that a transition is enabled at least twice in the same state, which implies a dynamic number of timers. Multiple enableness in time Petri nets is a natural way for modeling paradigms like multiple servers and multiple instances of codes [3].

For Colored Time Petri Nets, multiple enableness occurs when several combinations of colors enable a transition at a given time.

II. High-level colored Time Petri Nets

Notations: The sets $\mathbb{N}$, $\mathbb{Q}_{\geq 0}$, and $\mathbb{R}_{\geq 0}$ are respectively the sets of natural, non-negative rational, and non-negative real numbers. An interval I of $\mathbb{R}_{\geq 0}$ is a $\mathbb{Q}$-interval iff its left endpoint ${}^{\uparrow}I$ belongs to $\mathbb{Q}_{\geq 0}$ and its right endpoint $I^{\downarrow}$ belongs to $\mathbb{Q}_{\geq 0} \cup \{\infty\}$. We denote by $\mathcal{I}(\mathbb{Q}_{\geq 0})$ the set of $\mathbb{Q}$-intervals of $\mathbb{R}_{\geq 0}$. B^A stands for the set of mappings from A to B. If A is finite and $|A| = n$, an element of B^A is also a vector in B^n. The usual operators $+, -, <$ and $=$ are used on vectors of A^n with $A = \mathbb{N}, \mathbb{Q}, \mathbb{R}$ and are the point-wise extensions of their counterparts in A.

A. Definition and semantics

We consider a Petri Nets model, which encompasses both color and High-level functionalities. We consider a set C of colors. An arc is either associated with a color of C or can take on a particular color called *any*. For the firing of a transition, all its arcs associated with the *any* color must match to instantiate *any* at the same color taken from C.

If several values of *any* allow its enabling, the transition is multi-enabled, and in this case, many clocks are associated with the transition, allowing several firing dates depending on the enabling date and the time interval.

We now give the formal definition.

Definition 1 (High-level Colored Time Petri Net): A *High-level Colored Time Petri Net* (HCTPN) is a tuple $\mathcal{N} = (P, T, X, C, \mathsf{pre}, \mathsf{post}, (m_0, x_0), guard, update, I)$ where

- P is a finite set of *places*,
- T is a finite set of *transitions* such that $T \cap P = \emptyset$,
- X is a finite set of *variables* taking their value in the finite set $\mathbb{X}$ (such as bounded integer),
- C is a finite set of *colors* and $C_{any} = C \cup \{any\}$,
- $\mathsf{pre} : P \times T \to \mathbb{N}^{C_{any}}$ is the backward incidence mapping,
- $\mathsf{post} : P \times T \to \mathbb{N}^{C_{any}}$ is the forward incidence mapping,
- $guard : T \times \mathbb{X}^X \to \{true, false\}$ is the guard function,
- $update : T \times \mathbb{X}^X \to \mathbb{X}^X$ is the update function,
- $(m_0, x_0) \in \mathbb{N}^{P \times C} \times \mathbb{X}^X \to$ is the initial values m_0 of the marking and x_0 of the variables,
- $I : T \to \mathcal{I}(\mathbb{Q}_{\geq 0})$ is the *static firing interval* function,

1) Discrete behavior: For a marking $m \in \mathbb{N}^{P \times C}$, $m(p)$ is a vector in $\mathbb{N}^C$ and $m(p)[c]$ represents a number of *tokens* of color $c \in C$ in place $p \in P$. A valuation of the set of variables X is noted $x \in \mathbb{X}^X$. (m, x) is a discrete state of HCTPN.

a) Enabling of a transition: An arc $\mathsf{pre}(p,t) \in \mathbb{N}^{C_{any}}$ is a vector such that $\mathsf{pre}(p,t)[c]$ is the number of token of color $c \in C$ in place p needed to enable the transition t and $\mathsf{pre}(p,t)[any] > 0$ represents the fact that any color can enabled the transition. Let $T_{any} \subseteq T$ the set of transitions that can be enabled by *any* color: i.e. $T_{any} = \{t \in T, \exists p \in P, \text{ s.t. } \mathsf{pre}(p,t)[any] > 0\}$. Moreover we define the set $T_{\overline{any}} = T \setminus T_{any}$.

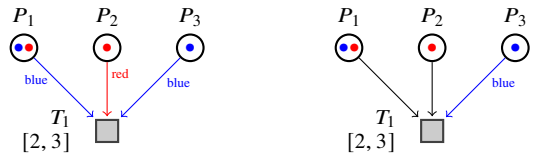
A transition $t \in T$ is said to be *enabled* by a given marking $m \in \mathbb{N}^{P \times C}$ in two cases depending on whether $t \in T_{any}$ or not:

- if $t \in T_{\overline{any}}$, and $\forall p \in P$ and $\forall c \in C$, $m(p)[c] \geq \mathsf{pre}(p,t)[c]$. We denote $\mathsf{en}(m,t) \in \{true, false\}$, the true value of this condition.
- if $t \in T_{any}$, and $\exists c_a \in C$ such that $\forall p \in P$, $m(p)[c_a] \geq \mathsf{pre}(p,t)[any]$ and $\forall c \in C$, $m(p)[c] \geq \mathsf{pre}(p,t)[c]$. The corresponding set of color c_a is noted $\mathsf{colorSet}_{\mathsf{any}}(m,t) \subseteq C$

Finally, a transition $t \in T$ is said to be *enabled* by a given marking $m \in \mathbb{N}^{P \times C}$ and a valuation $x \in \mathbb{X}^X$ if $\mathsf{en}(m,t) = true$ or $\mathsf{colorSet}_{\mathsf{any}}(m,t) \neq \emptyset$ and $guard(t,x) = true$.

We illustrate the enabling condition by two examples with two colors $C = \{blue, red\}$. On the figures, a black arc means that it is associated with the color *any*. For the HCTPN given in Figure 1.a, the transition $T_1 \in T_{\overline{any}}$.

We have $\mathsf{pre}(T_1) = \begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \end{array} \begin{array}{c} \begin{array}{ccc} red & blue & any \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{array}$. The initial marking is $m_0 = \begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \end{array} \begin{array}{c} \begin{array}{cc} red & blue \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array}$ that enables the transition T_1 and $\mathsf{en}(m_0, T_1) = true$.



1.a: $T_1 \in T_{\overline{any}}$ 1.b: $T_1 \in T_{any}$

Fig. 1. Enabling transition

Now we consider the HCTPN given in Figure 1.b with the same initial marking m_0 but where the transition $T_1 \in T_{any}$ since at least one arc (here two) is associated with the color *any*.

We have $\mathsf{pre}(T_1) = \begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \end{array} \begin{array}{c} \begin{array}{ccc} red & blue & any \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{array}$. The transition is enabled only if *any* takes the *red* value then $\mathsf{colorSet}_{\mathsf{any}}(m_0, T_1) = \{red\}$. If the place $P2$ had two tokens with one token per color, then the transition would be multi-enabled by the 2 colors leading to $\mathsf{colorSet}_{\mathsf{any}}(m_0, T_1) = \{blue, red\}$.

b) Firing of a transition: An arc $\mathsf{post}(p,t) \in \mathbb{N}^{C_{any}}$ is a vector such that $\mathsf{post}(p,t)[c]$ is the number of token of color $c \in C$ produced in place p by the firing of the transition t and $\mathsf{post}(p,t)[any]$ gives the number of token produced in p with the color $c \in \mathsf{colorSet}_{\mathsf{any}}(m,t)$ used for the firing of t.

Firing an enabled transition $t \in T_{\overline{any}}$ from (m,x) such that $\mathsf{en}(m,t) = true$ and $guard(t,x) = true$ leads to a new marking m' defined by $\forall c \in C, \forall p \in P$, $m'(p)[c] = m(p)[c] - \mathsf{pre}(p,t)[c] + \mathsf{post}(p,t)[c]$ and a new valuation $x' = update(t,x)$. This new marking is denoted $m' = \mathsf{firing}(m,t,\bullet)$ where $\bullet$ denote the fact that no *any* color has to be instantiated for this firing.

Firing an enabled transition $t \in T_{any}$ from (m,x) with the *any* color $c_a \in \mathsf{colorSet}_{\mathsf{any}}(m,t)$ leads to a new marking defined by $\forall c \in C \setminus \{c_a\}, \forall p \in P,\ m'(p)[c] = m(p)[c] - \mathsf{pre}(p,t)[c] + \mathsf{post}(p,t)[c]$ and $\forall p \in P,\ m'(p)[c_a] = m(p)[c_a] - \mathsf{pre}(p,t)[c_a] - \mathsf{pre}(p,t)[any] + \mathsf{post}(p,t)[c_a] + \mathsf{post}(p,t)[any]$ and a new valuation $x' = update(t,x)$. This new marking is denoted $m' = \mathsf{firing}(m,t,c_a)$.

We denote by $\mathsf{newen}((m,x),t,c)$ the set of transitions that are newly enabled by the firing of t from (m,x) with the color c ($c = \bullet$ if $t \in T_{\overline{any}}$).

Let us go back to the HCTPN of Figure 1.a, the firing of $T_1 \in T_{\overline{any}}$ from m_0 leads to the marking $m_1 = \begin{matrix} \\ P_1 \\ P_2 \\ P_3 \end{matrix} \begin{matrix} \begin{matrix} red & blue \end{matrix} \\ \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \end{matrix}$. It is noted $m_0 \xrightarrow{(T_1,\bullet)} m_1$

For the HCTPN of Figure 1.b, the firing of $T_1 \in T_{any}$ is possible only for $any = red$ and leads to the marking $m_2 = \begin{matrix} \\ P_1 \\ P_2 \\ P_3 \end{matrix} \begin{matrix} \begin{matrix} red & blue \end{matrix} \\ \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \end{matrix}$. It is noted $m_0 \xrightarrow{(T_1,red)} m_2$.

2) Time behavior: For any $t \in T_{any}$, $v(t,c)$ is the valuation of the clock associated with t and the color $c \in C$. i.e. it is the time elapsed since the transition t has been newly enabled by m with $c \in \mathsf{colorSet}_{\mathsf{any}}(m,t)$. For other transitions $t \in T_{\overline{any}}$, $v(t,\bullet)$ is the valuation of the clock associated with t.

$\bar{0}$ is the initial valuation with $\forall t \in T, \forall c \in C \cup \{\bullet\}, \bar{0}(t,c) = 0$.

As an example, if we keep only the useful clocks, the initial valuation of the HCTPN of Figures 1, is $v_0 = \begin{matrix} \\ T_1 \end{matrix} \begin{matrix} \begin{matrix} \bullet & red & blue \end{matrix} \\ \begin{pmatrix} 0 & & \end{pmatrix} \end{matrix}$ for Figure 1.a, and $v_0 = \begin{matrix} \\ T_1 \end{matrix} \begin{matrix} \begin{matrix} \bullet & red & blue \end{matrix} \\ \begin{pmatrix} & 0 & 0 \end{pmatrix} \end{matrix}$ for Figure 1.b,

A *state* of the net $\mathcal{N}$ is a tuple $((m,x),v)$ in $\mathbb{N}^{P\times C} \times \mathbb{X}^X \times {\mathbb{R}_{\geq 0}}^{T\times C}$, where: m is a marking, x is a variable valuation and v is a valuation of the clocks.

Definition 2 (Semantics of a HCTPN): The semantics of a HCTPN is a timed transition system $(Q, Q_0, \rightarrow)$ where:

- $Q \subseteq \mathbb{N}^{P\times C} \times \mathbb{X}^X \times {\mathbb{R}_{\geq 0}}^{T\times C}$
- $Q_0 = ((m_0,x_0),\bar{0})$
- $\rightarrow \in Q \times ((T \times C \cup \{\bullet\}) \cup \mathbb{R}_{\geq 0}) \times Q$ consists of two types of transitions:
  - discrete transitions (firing t from $((m,x),v)$) iff:
    - $((m,x),v) \xrightarrow{(t\in T_{\overline{any}},\bullet)} ((m',x'),v')$ with
      - $\mathsf{en}(m,t) = true$ and $v(t) \in I(t)$,
      - $m' = \mathsf{firing}(m,t,\bullet)$
    - $((m,x),v) \xrightarrow{(t\in T_{any},c)} ((m',x'),v')$ with
      - $c \in \mathsf{colorSet}_{\mathsf{any}}(m,t)$ and $v(t,c) \in I(t)$,
      - $m' = \mathsf{firing}(m,t,c)$
    - $guard(t,x) = true$ and $x' = update(t,x)$
    - $\forall t'$ s.t. $\mathsf{en}((m',t') = true)$
      - $v'(t',\bullet) = v(t',\bullet)$ if $t' \notin \mathsf{newen}((m,x),t,\bullet)$,
      - $v'(t',\bullet) = 0$ otherwise
    - $\forall t'$ and $\forall c \in \mathsf{colorSet}_{\mathsf{any}}(m',t')$
      - $v'(t',c) = v(t',c)$ if $t' \notin \mathsf{newen}((m,x),t,c)$,
      - $v'(t',c) = 0$ otherwise
  - time transitions: $((m,x),v) \xrightarrow{d\in\mathbb{R}_{\geq 0}} ((m,x),v')$, iff:
    - $\forall t \in T_{\overline{any}}$ s.t. $\mathsf{en}(m,t) = true$,
      - $v'(t,\bullet) \leq I(t)^{\downarrow}$
      - $v'(t,\bullet) = v(t,\bullet) + d$
    - $\forall t \in T_{any}$ and $\forall c \in \mathsf{colorSet}_{\mathsf{any}}(m,t)$,
      - $v'(t,c) \leq I(t)^{\downarrow}$
      - $v'(t,c) = v(t,c) + d$

We now illustrate the main features of HCTPN in an example. The guards are in green in the Figures and the update in purple.

B. Examples of HCTPN

We give two examples. The first one illustrates the high-level functionalities, and the second illustrates the notion of color and multi-enableness.

a) Example 1: The HCTPN given in Figure 2 illustrates time behavior and high-level manipulation of variables. This HCTPN has only one color and a single variable cpt. We assume that *random(1,10)* returns an integer between 1 and 10.

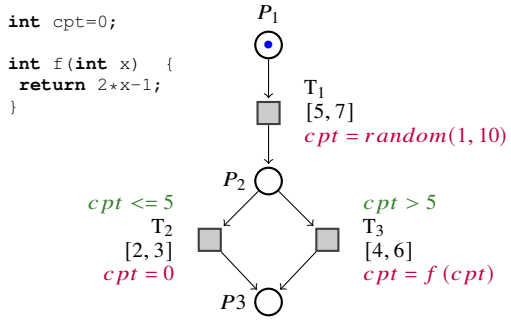


Fig. 2. HCTPN illustrating high-level manipulation of variables

A marking is written by the matrix $(|P|,|C|)$. Since there is only one color, the marking is a vector and the initial marking is then $m_0 = \begin{matrix} P_1 \\ P_2 \\ P_3 \end{matrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ and enables the transition T_1. The valuations of the clocks are given by the matrix (here a vector) such that the initial valuation is $v_0 = \begin{matrix} T_1 \\ T_2 \\ T_3 \end{matrix} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$. Since the set of variables is $X = \{cpt\}$, we note a state $s = (m, cpt, v)$. The initial state is $q_0 = (m_0, 0, v_0)$. The transition T_1 can fire after elapsing 5 time units. We now consider the run where the random function called by the update of the firing of T_1 returned the value 7. Then the transition's guard T_2 is false, and the transition T_3 is enabled. We assume that the transition T_3 took 4.6 time units for this run. The firing of the transition T_3 executes the corresponding update and calls the function f that returns 13. The corresponding run is as follows: $\left(\begin{pmatrix}1\\0\\0\end{pmatrix}, 0, \begin{pmatrix}0\\0\\0\end{pmatrix}\right) \xrightarrow{5} \left(\begin{pmatrix}1\\0\\0\end{pmatrix}, 0, \begin{pmatrix}5\\0\\0\end{pmatrix}\right) \xrightarrow{(T_1,\bullet)} \left(\begin{pmatrix}0\\1\\0\end{pmatrix}, 7, \begin{pmatrix}0\\0\\0\end{pmatrix}\right) \xrightarrow{4.6} \left(\begin{pmatrix}0\\1\\0\end{pmatrix}, 7, \begin{pmatrix}0\\0\\4.6\end{pmatrix}\right) \xrightarrow{(T_3,\bullet)} \left(\begin{pmatrix}0\\0\\1\end{pmatrix}, 13, \begin{pmatrix}0\\0\\0\end{pmatrix}\right)$

*b) **Atomicity** :* An update can be described as a sequence of imperative code expressed in a programming language such as C. This code is evaluated sequentially w.r.t. the semantics of the C language; however, its execution is considered atomic from the HCTPN point of view.

Hence, if x and x' are respectively the values of the variables before and after the execution of the code of an update of a transition t from x, the firing of t leads atomically to $x' = update(t, x)$.

c) Example 2: The model given in Figure 3 is a HCTPN with a set of two colors $C = \{red, blue\}$. Several combinations of color usage, on guards and in updates, via the $\$any$ variable are presented.

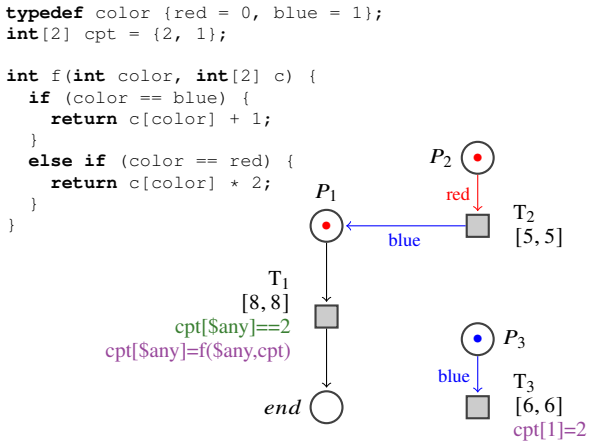


Fig. 3. HCTPN model illustrating colored multi-enableness

In the sequel a marking is written by the matrix $(|P|, |C|)$. The initial marking is then $m_0 = \begin{matrix} \\ P_1 \\ P_2 \\ P_3 \\ end \end{matrix} \begin{matrix} red & blue \\ \end{matrix} \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$ and enables the transitions T_1, T_2 and T_3. The variable cpt of the model is an array indexed by the color. Its initial value is $x_0 = \begin{matrix} red & blue \\ \end{matrix}$ $cpt \begin{pmatrix} 2 & 1 \end{pmatrix}$ The valuations of the clocks are given by the matrix such that the initial valuation is $v_0 = \begin{matrix} \bullet & red & blue \\ T_1 \\ T_2 \\ T_3 \end{matrix} \begin{pmatrix} & 0 & 0 \\ 0 & & \\ 0 & & \end{pmatrix}$ (We omit the insignificant values). We note a state $s = (m, x, v)$. The initial state is $q_0 = (m_0, x_0, v_0)$. Since the time intervals are points, we have an unique run:

$$\left(\begin{pmatrix}1&0\\1&0\\0&1\\0&0\end{pmatrix},\begin{pmatrix}2&1\end{pmatrix},\begin{pmatrix}0&0\\0&0\end{pmatrix}\right) \xrightarrow{5} \left(\begin{pmatrix}1&0\\1&0\\0&1\\0&0\end{pmatrix},\begin{pmatrix}2&1\end{pmatrix},\begin{pmatrix}5&0\\5&5\end{pmatrix}\right) \xrightarrow{(T2,\bullet)}$$

$$\left(\begin{pmatrix}1&1\\0&0\\0&1\\0&0\end{pmatrix},\begin{pmatrix}2&1\end{pmatrix},\begin{pmatrix}0&5&0\\5&&\end{pmatrix}\right) \xrightarrow{1} \left(\begin{pmatrix}1&1\\0&0\\0&1\\0&0\end{pmatrix},\begin{pmatrix}2&1\end{pmatrix},\begin{pmatrix}0&6&0\\6&&\end{pmatrix}\right) \xrightarrow{(T3,\bullet)}$$

$$\left(\begin{pmatrix}1&1\\0&0\\0&0\\0&0\end{pmatrix},\begin{pmatrix}2&2\end{pmatrix},\begin{pmatrix}0&6&0\\0&&\end{pmatrix}\right) \xrightarrow{2} \left(\begin{pmatrix}1&1\\0&0\\0&0\\0&0\end{pmatrix},\begin{pmatrix}2&2\end{pmatrix},\begin{pmatrix}0&8&2\\0&&\end{pmatrix}\right) \xrightarrow{(T1,red)}$$

$$\left(\begin{pmatrix}0&1\\0&0\\0&0\\1&0\end{pmatrix},\begin{pmatrix}4&2\end{pmatrix},\begin{pmatrix}0&0&2\\0&&\end{pmatrix}\right) \xrightarrow{6} \left(\begin{pmatrix}0&1\\0&0\\0&0\\1&0\end{pmatrix},\begin{pmatrix}4&2\end{pmatrix},\begin{pmatrix}0&0&8\\0&&\end{pmatrix}\right) \xrightarrow{(T1,blue)}$$

$$\left(\begin{pmatrix}0&0\\0&0\\0&0\\1&1\end{pmatrix},\begin{pmatrix}4&3\end{pmatrix},\begin{pmatrix}0&0&0\\0&&\end{pmatrix}\right)$$

The time elapses from the initial marking until reaching date 5. T_2 is fired, and a blue token is dropped in the place P_1. The clock of T_1 associated with the red color has reached the value 5. The clock of T_1 associated with the blue color cannot start yet because the guard is false for this color. At date 6, T_3 is fired, causing a change in the variable `cpt` that makes the guard of T_1 true for the blue color. The clock associated with the blue color for T_1 can therefore start. Both colors enable the transition T_1, and the corresponding clocks give the time from the two enabling. After two more time units, T_1 is fired for the red color; at this moment, the clock of T_1 for the blue color has reached 2. Finally, after 6-time units, T_1 is fired for the blue color, ending the run.

C. Decidability and complexity

Let us recall that a *High-level Colored Time Petri Net* (HCTPN) is a tuple $\mathcal{N} = (P, T, X, C, \mathsf{pre}, \mathsf{post}, (m_0, x_0), guard, update, I)$ such that the set C of *colors* is finite and X is a finite set of *variables* taking their value in a finite set $\mathbb{X}$.

Theorem 1: Reachability problem for bounded High-level Colored Time Petri Net is decidable

Proof: From HCTPN semantics, a transition can be multi-enableded a maximum of $|C|$ times at a given time. Hence, firing domains can be symbolically abstracted with state classes using difference bound matrix (DBM) over $|C| \times T$ variables. As in [4], [5], the number of DBM is finite. Moreover, the number of markings of a k-bounded Petri net is bounded by $(k+1)^{|P|}$ then the number of discrete states of a k-bounded HCTPN is bounded by $(k+1)^{|P|} \times \mathbb{X}^X$. Hence, computable finite abstractions of the state space exist, and the reachability problem is decidable. □

Temporal logics were introduced by Pnueli [6] as specification languages to express the behaviors of sequential and concurrent systems and TCTL (Timed Computation Tree Logic), introduced in [7], is a real-time extension of the branching-time temporal logic CTL (Computation Tree Logic).

We can prove, as in [8] for bounded Time Petri Nets, that the theoretical complexity of TCTL model-checking for bounded High-level Colored Time Petri Nets is PSPACE-complete. However, as for Timed Automata and Time Petri Nets, no effective PSPACE algorithm exists in practice, and real implementations are with exponential algorithms.

Moreover, HCTPN can be extended with stopwatches allowing the modeling of preemptive scheduling as in [9]. In the stopwatch setting, the reachability problem is undecidable but efficient semi-algorithms are implemented in ROMÉO [10] that converges for almost all practical cases.

III. APPLICATION

The application chosen as an example is the modeling of the spinlocks mechanism present in the PowerPC MPC5643L dual-core microcontroller from NXP [11] and used to build critical sections for parallel program executions. This mechanism is based on a hardware unit, the SEMA4 for *Semaphore Unit*. For the software, this unit is materialized as an array of 16 registers implementing 16 locks. The exclusive access to the

bus regulates the concurrent accesses to one of these registers. If a register contains the value 0, the lock is available, and it is possible to write to it. If the value contained is different from 0, the lock is occupied, and it is only possible to write the value 0 to it, and writing any other value has no effect. Therefore, getting a lock consists in writing a value different from 0 and releasing it consists in writing 0. Thus, using this unit requires respect of a protocol, and an example of implementation is given on page 1322 of [11]. Algorithm 1 reproduces it.

Algorithm 1 Lock acquisition protocol. *gate* is one of the hardware registers of the SEMA4 unit.

```
cn ← core_number                    ▷ (1 .. N)
do
    lock ← gate
while lock ≠ 0
do
    gate ← cn
    lock ← gate
while lock ≠ cn
```

A. Modeling the spinlocks mechanism

The modeling takes advantage of the possibilities of the HCTPN. The hardware part, which by virtue of the exclusive access to the bus allows operations that are intrinsically atomic, is modeled using functions. To simplify the presentation, only one register of the SEMA4 unit, *gate*, is modeled but the model could just as well use an array to accurately model the hardware. The listing 1 shows this part of the model. *gate* is initialized to the UNLOCKED state (line 2) and is accessible through three functions. *lock* (line 4) mimics the behavior of the hardware by only allowing writing to *gate* if its value is UNLOCKED. The core number corresponding to a color and a color among N being coded by an integer from 0 to N-1, a core locks by writing $color$+1 in *gate*. *unlock* (line 10) simply writes the value UNLOCKED into *gate*. *isLocked* (line 14 returns 1 if the *gate* is locked, 0 otherwise). Finally, *isLockedBy* (line 22) returns 1 if *core* holds the lock and returns 0 otherwise.

Each function of the software is modeled by an HCTPN reproducing the control flow graph of the function. Two HCTPNs model the functions *GetSpinLock* and *RelSpinLock* (see Figure 4). *GetSpinLock* corresponds to the algorithm 1 and *$any* allows to represent on which core the function is executed. The call of a function modeled by a HCTPN is done by dropping a token of the core color in the initial place. Thus, "calling" the function *GetSpinLock* is performed by the update `GetSpinLock[color] = 1` on a transition of the HCTPN of the calling function. This is identical to drawing an arc of the corresponding color between the transition and the initial place of *GetSpinLock*. The function return requires a synchronization. This one is implemented by a variable of type array and of size equal to the number of colors and indexed by the color, i.e. the core on which the function call is made. We have therefore for our two function models the two variables `endOfGSL` and `endOfRSL`, see listing 2.

Listing 1. Modeling of the SEMA4 hardware

```
1  const int UNLOCKED = 0;
2  int gate = UNLOCKED;
3
4  void lock(int core, int &ioGate) {
5    if (ioGate == UNLOCKED) {
6      ioGate = core + 1;
7    }
8  }
9
10 void unlock(int &ioGate) {
11   ioGate = UNLOCKED;
12 }
13
14 int isLocked(int &inGate) {
15   if (inGate == UNLOCKED) {
16     return 0;
17   } else {
18     return 1;
19   }
20 }
21
22 int isLockedBy(int core, int &inGate) {
23   if (inGate == core + 1) {
24     return 1;
25   } else {
26     return 0;
27   }
28 }
```

B. Verification of the system

The spinlock model is completed by an application model. Two tasks, τ_0, running on core 0 (red color) and τ_1, running on core 1 (blue color), are modeled as shown in Figure 5. The task τ_0 takes then releases the spinlock while the task τ_1 has the possibility to take it, as τ_0 does, or to reach the final state without taking the spinlock. We want to check that τ_0 and τ_1 cannot occupy simultaneously and respectively the places P_{12} and P_{22} by the CTL formula $A\Box(\neg(P_{12}[0] == 1 \wedge P_{22}[1] == 1))$. Here $P_{12}[0]$ denotes the marking of P_{12} for the red color and $P_{22}[1]$ denotes the marking of P_{22} for the blue color. ROMÉO answers *true* for this CTL formula.

IV. CONCLUSION

This paper has presented High-level Colored Time Petri Nets. This formalism allows to model complex systems and is well adapted to multi-core hardware and software modeling, as shown in the case study. The high-level features allow the modeling of the software, the timed transitions model the execution times, and the colors specify the hardware where the software is executed. A timed transition enabled by more than one color allows true concurrency modeling. The model checking of this formalism is implemented in the ROMÉO tool. Future work will focus on using this formalism for the certification of an OSEK and AUTOSAR compliant embedded OS.

Listing 2. Synchronization variables for the function return

```
1 int[2] endOfGSL = {0, 0};
2 int[2] endOfRSL = {0, 0};
```

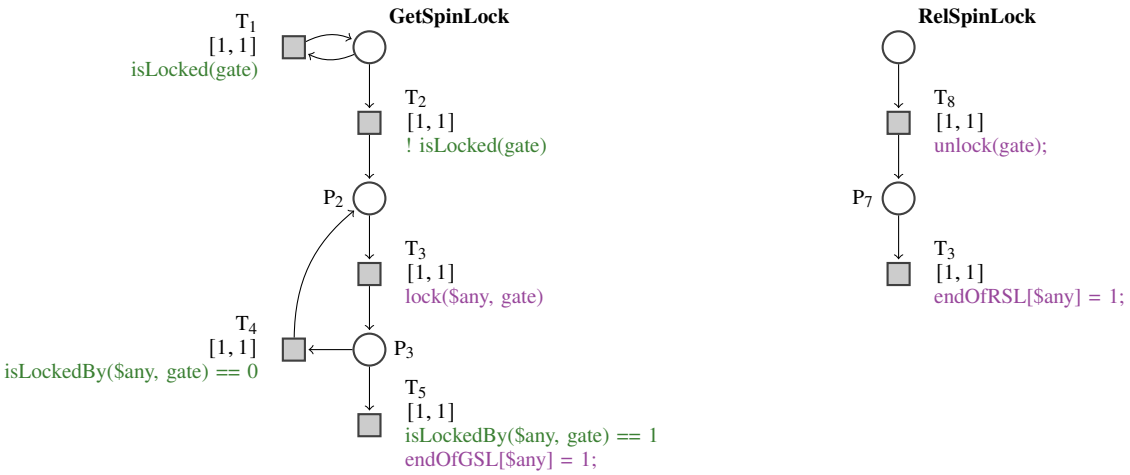


Fig. 4. The GetSpinLock and RelSpinLock function models

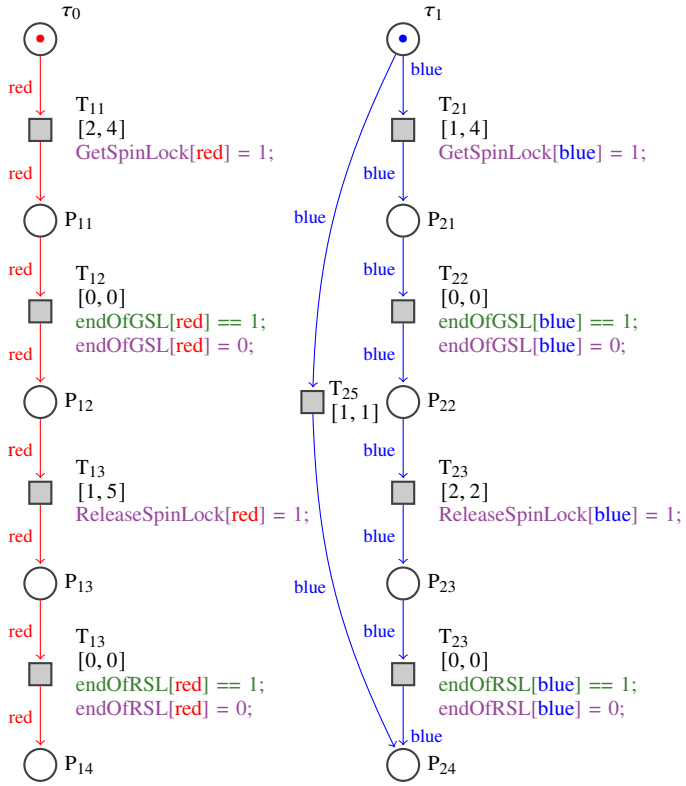


Fig. 5. The tasks models

REFERENCES

[1] L. Hillah, F. Kordon, L. Petrucci, and N. Trèves, "Pn standardisation: A survey," in *Formal Techniques for Networked and Distributed Systems - FORTE 2006*, ser. Lecture Notes in Computer Science, vol. 4229. Springer Berlin Heidelberg, 2006, pp. 307–322.

[2] E. Kindler and L. Petrucci, "A framework for the definition of variants of high-level petri nets," in *Proceedings of the Tenth Workshop and Tutorial on Practical Use of Coloured Petri Nets and CPN Tools (CPN '09)*, 2009, pp. 121–137.

[3] M. Boyer and M. Diaz, "Multiple enabledness of transitions in petri nets with time," in *Proceedings of the 9th International Workshop on Petri Nets and Performance Models, PNPM 2001, Aachen, Germany, September 11-14, 2001*, R. German and B. R. Haverkort, Eds. IEEE Computer Society, 2001, pp. 219–228.

[4] B. Berthomieu and M. Menasche, "An enumerative approach for analyzing time petri nets," in *Information Processing: proceedings of the IFIP congress 1983*, ser. IFIP congress series, R. E. A. Mason, Ed., vol. 9. Elsevier Science Publishers, Amsterdam, 1983, pp. 41–46.

[5] B. Berthomieu and M. Diaz, "Modeling and verification of time dependent systems using time petri nets," *IEEE transactions on software engineering*, vol. 17, no. 3, pp. 259–273, March 1991.

[6] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA*. IEEE Computer Society, 1977, pp. 46–57.

[7] R. Alur, C. Courcoubetis, and D. Dill, "Model-checking in dense real-time," *Information and Computation*, vol. 104, no. 1, pp. 2–34, 1993.

[8] H. Boucheneb, G. Gardey, and O. H. Roux, "TCTL model checking of time Petri nets," *Journal of Logic and Computation*, vol. 19, no. 6, pp. 1509–1540, Dec. 2009.

[9] I. Haur, J.-L. Béchennec, and O. H. Roux, "Formal schedulability analysis based on multi-core rtos model," in *29th International Conference on Real-Time Networks and Systems*, ser. RTNS'2021. New York, NY, USA: Association for Computing Machinery, 2021, pp. 216–225. [Online]. Available: https://doi.org/10.1145/3453417.3453437

[10] D. Lime, O. H. Roux, C. Seidner, and L.-M. Traonouez, "Romeo: A parametric model-checker for petri nets with stopwatches," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2009, pp. 54–57.

[11] Freescale Semiconductor, *MPC5643L Microcontroller Reference Manual*, rev. 10 ed., NXP, High Tech Campus 60, 5656 AG Eindhoven, The Netherlands, June 2013.