ε-TPN: definition of a Time Petri Net formalism simulating the behaviour of the timed grafcets

Médésu Sogbohossou — Antoine Vianou

Département Génie Informatique et Télécommunications
École Polytechnique d'Abomey-Calavi (EPAC), 01 BP 2009 Cotonou, BENIN
{medesu.sogbohossou,antoine.vianou}@epac.uac.bj

ABSTRACT. To allow a formal verification of timed GRAFCET models, many authors proposed to translate them into formal and well-reputed languages such as timed automata or Time Petri nets (TPN). Thus, the works presented in [Sogbohossou, Vianou, Formal modeling of grafcets with Time Petri nets, IEEE Transactions on Control Systems Technology, 23(5)(2015)] concern the TPN formalism: the resulting TPN of the translation, called here ε-TPN, integrates some infinitesimal delays (ε) to simulate the synchronous semantics of the grafcet. The first goal of this paper is to specify a formal operational semantics for an ε-TPN to amend the previous one: especially, priority is introduced here between two defined categories of the ε-TPN transitions, in order to respect strictly the synchronous hypothesis. The second goal is to provide how to build the finite state space abstraction resulting from the new definitions.

RÉSUMÉ. Afin de permettre la vérification formelle des grafcets temporisés, plusieurs auteurs ont proposé de les traduire dans des langages formels de réputation tels que les automates temporisés et les réseaux de Petri temporels (TPN). Ainsi, les travaux présentés dans [Sogbohossou, Vianou, Formal modeling of grafcets with Time Petri nets, IEEE Transactions on Control Systems Technology, 23(5)(2015)] concernent le formalisme des TPN: le réseau résultant de la traduction, dénommé ici ε-TPN, intègre des délais infinitésimaux (ε) pour simuler la sémantique synchrone du grafcet. Le premier objectif de cet article est de définir la sémantique opérationnelle d'un ε-TPN afin d'améliorer l'ancienne définition: spécifiquement, une priorité est introduite ici entre deux catégories de transitions définies pour ces réseaux, dans l'optique de respecter rigoureusement l'hypothèse synchrone. Le second but est de fournir une méthode de calcul de l'espace d'état fini qui découle des nouvelles définitions.

KEYWORDS : Time Petri Net, timed grafcet, state class, partial order execution, synchronous modelling

MOTS-CLÉS : Réseau de Petri temporel, grafcet temporisé, classe d'état, exécution ordre partiel, modélisation synchrone

1. Introduction

Formal specification of a critical system at the early stage of conception is often needed to achieve their reliability in working, by means of languages allowing simulation or formal verification on the established model of this system [6]. Graphical state-transition modeling formalisms in engineering are appreciated because of their intuitiveness. They are based on the automata theory, ensuring an unambiguous description of the behaviours of a system. Petri nets (PN) are one of these formalisms, used to model in a compact and explicit way the concurrency and the synchronization between the dynamic components of the so-called discrete-event systems [5]. In PNs, firing of transitions (with possibly multiple concurrent firings in the same instant) changes the state and express the dynamics of the modeled system. Time Petri nets (TPN) [1] are one of its extensions, suitable when quantitative time analyses are required for the real-time specifications.

Otherwise, the engineering pratices often promote less formal graphical languages, because of their increased semantic richness (for instance, litteral formulae and hierarchical modeling do not exist in the ordinary PNs) favoring more compact and fluent modeling to the detriment of unambiguous interpretations. These are the cases of formalisms derived from PNs, such as GRAFCET [1] (IEC 60848 standard) [8] and SFC (Sequential Function Chart, IEC 61131-3 standard) [9], used mainly in the world of the manufacturing control. Whereas simultaneous fireable transitions are always done by their total interleaving with PNs, the semantics of these two IEC standards considers only synchronous firings; a consequence is that the notion of transitions in conflict does not exist in GRAFCET and SFC formalisms. GRAFCET is intended for specification purposes (event-driven modeling), contrary to SFC for implementation uses (clock-driven modeling), and is considered in the sequel.

To allow a formal verification of GRAFCET (or SFC) models integrating quantitative time informations, many authors proposed to translate them into formal and well-reputed languages such as timed automata [7] or TPN [12, 11]. The work in [11] is focused on defining some transformation rules which are used to translate the entities composing a timed and not necessarily sound grafcet chart (steps, transitions, literal variables, actions) into connected blocks to obtain the resulting TPN. The method exploits the similarity between TPN and GRAFCET to avoid exponential size of the translation, and implicitly relies on a clear choice about the GRAFCET semantics.

To deal with synchronous firings inherent to GRAFCET formalism, the authors [11] introduced transitions with infinitesimal ε delays, however without redefining formally the resulting extended TPN. The first goal of this paper is to palliate this lack, by specifying a formal semantics for the so-called ε-TPN; the slight differences with the definition in [11] are also presented. Basing on this new definition, the second goal is to provide how to build the state space abstraction of an ε-TPN (which is just sketched in [11]) with the ε delays. Particularly, it is shown how to take advantage from this kind of TPN to cope with the state-space explosion problem, by avoiding useless interleaving of concurrent firings and by abstracting some state classes during the state-space construction.

The three next sections are organized as follows. In Section 2 are given definitions about TPN with ε delays on some transitions: especially, the formal operational semantics of ε-TPN is presented and compared with [11]. In Section 3, the algorithm for generating the finite abstraction (derived from the state class construction for classic TPN [1, 3])

1. Acronym in French: *GRAphe Fonctionnel de Commande Etape Transition.*

taking into account the explosion problem is provided. At last, the conclusion section summarizes the contribution of this paper, and sketches its perspectives.

2. Syntax and semantics of ε-TPN

2.1. The context

The works [11] proposed to translate a timed (and non-hierarchical) grafcet into TPN for formal verification purposes. They give some transformation rules to convert the elements of a grafcet into modules of TPN which are connected, as one goes along a suitable order of translation. An extra module named *phase sequencer* (Fig. 1(a)) is necessary to allow a transient evolution without modification of inputs as external events: it forces alternation between the reaction phase (called *evolution* with grafcets) and an external event production in a stable situation. After adding this first module, the generation of the complete TPN is done by translating sequentially (see figures in Annex B): the steps, the inputs, the timed variables, the outputs, the counter variables, the continuous and conditional actions, the stored actions and the grafcet transitions.

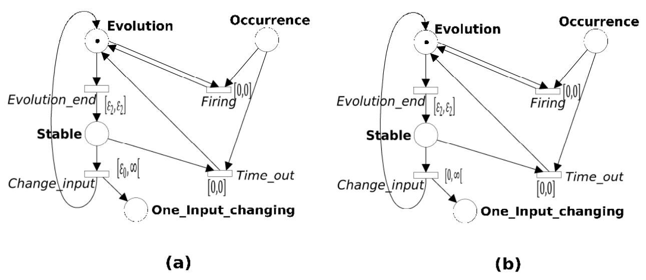

Figure 1. *Phase sequencer: former version (a), new version (b)*

All the transitions in the resulting TPN bear intervals of the form $[\delta, \delta]$ (δ is a delay), except only one: that is called $Change\_input$ in the phase sequencer (Fig. 1), of which firing allows an input event to occur at any time during a stable situation.

An illustration of a grafcet translation into TPN is given in Annex C.

2.2. Syntax

Lets ε_0, an infinitesimal constant (a delay comparable to 0^+). It follows that $\varepsilon_n \overset{\text{def}}{=} \varepsilon_0 \times n$ for any $n \in \mathbb{N}^*$, and $\mathcal{E} \overset{\text{def}}{=} \{\varepsilon_n \mid n \in \mathbb{N}^*\}$. It is assumed, for any $\varepsilon_n \in \mathcal{E}$ and any $d \in \mathbb{R}^{+*}$, that $0 < \varepsilon_n < d$ and $d + \varepsilon_n \approx d$ hold. By extension, $\mathcal{E}_0 \overset{\text{def}}{=} \mathcal{E} \cup \{0\}$.

Définition 1. *A* Time Petri net *(TPN) is a tuple* $(P, T, W, W_I, W_R, ED, LD, M_0)$ *such as:*

1) *the nodes:* P *is the set of* places *and* T *is the set of* transitions *(*$P \cap T = \emptyset$*);*
2) $W : P \times T \cup T \times P \longrightarrow \mathbb{N}$ *defines the* regular arcs *between nodes, and their weights;*
3) $W_R : P \times T \longrightarrow \mathbb{N}$ *defines the* read arcs*;*
4) $W_I : P \times T \longrightarrow \mathbb{N}^* \cup \{\infty\}$ *defines the* inhibitor arcs*;*

5) the initial marking $M_0 : P \longrightarrow \mathbb{N}$;

6) the earliest firing delays $ED : \; T \longrightarrow \mathbb{Q}^+ \cup \mathcal{E}$;

7) the latest firing delays $LD : \; T \longrightarrow \mathbb{Q}^+ \cup \mathcal{E} \cup \{\infty\}$;

8) the set T is a partition of three subsets $T_{\mathcal{E}_0}$, T_T and T_∞ such as:

a) $t \in T_{\mathcal{E}_0}$ iff $ED(t) = LD(t) \in \mathcal{E}_0$;

b) $t \in T_T$ iff $ED(t) = LD(t) \in \mathbb{Q}^{+*}$;

c) $t \in T_\infty$ iff $ED(t) = 0$ and $LD(t) = \infty$.

Items 1 to 5 correspond to the classic definition about ordinary Petri nets. Let $p \in P$ and $t \in T$. Graphically, an arc ((p, t) or (t, p)) may be qualified regular, inhibitor or read. By default, the weight is 0 when no regular arc links two nodes (one place and one transition), and is 1 when any arc is represented without its weight. Also, no read arc (resp. no inhibitor arc) directed from a place p to a transition t means the weight $W_R(p, t) = 0$ (resp. $W_I(p, t) = \infty$).

A marking M ($M : P \longrightarrow \mathbb{N}$) enables a transition t iff: $\forall p \in P, (M(p) \geq W(p, t) \wedge M(p) \geq W_R(p, t)) \wedge \nexists p \in P, W_I(p, t) \leq M(p)$. $En(M)$ designates the set of transitions enabled by the marking M. In the next subsection, the notation ${}^\bullet t$ (resp. $t^\bullet$) indicates the multiset[2] of the input (resp. output) places for a given transition t, by the relation W.

The following items (6 to 8) of the definition 1 extend the classic definition about TPN, by integrating the infinitesimal delays and the specific constraints on the static firing interval form of a transition. In practice, transitions with fixed delay in $T_\mathcal{E}$ are aimed at modeling synchronous firings, mainly useful during a reaction phase of the control part. External events to the control part are modeled by transitions T_∞ (input events) and T_T (delay events for the timed variables): it is assumed here that two transitions of these kinds may never occur simultaneously in the same instant. Then, only one firing in the set $T_\infty \cup T_T$ should trigger a reaction, that is (a sequence of) synchronous firings in the set $T_{\mathcal{E}_0}$. The transitions with interval $[0, 0]$ may be used as well in a reaction phase as for an external event production to update some informations instantaneously.

In the sequel, the notation $\delta(t)$ is related to a transition t with a fixed delay as the static interval: $\delta(t) = ED(t) = LD(t)$.

The definition 1 is more general than the informal presentation in [11] where: T_∞ is made of a single transition[3] t such as $ED(t) = \varepsilon_0$ and $LD(t) = \infty$; and $\mathcal{E} \overset{\text{def}}{=} \{\varepsilon_0, \varepsilon_1, \varepsilon_2\}$.

2.3. Semantics

The chosen operational semantics of TPN is the *standard semantics* (as in the references [1, 4]). Moreover, among the enabled transitions, $T_{\mathcal{E}_0}$ transitions always have priority over those in $T_\infty \cup T_T$ (unlike [11] which considers no such priority), according to the synchronous hypothesis: the reaction time to an external event is always smaller than the delay before any next occurrence of external events. Then, non-infinitesimal elapsing of time is only possible when no transition in $T_{\mathcal{E}_0}$ is enabled: time elapse is considered dense in such a *stable* state. A next firing in $T_\infty \cup T_T$ may allow entering in a *reaction* phase where only transitions in $T_{\mathcal{E}_0}$ are fired until reaching a new stable state.

Among the well-known two characterizations of TPN state [3], interval state and clock state, the second one (which is the more general) is used to define the semantics of ε-TPN.

2. Given a set X, a multiset on X is a function $Y : X \to \mathbb{N}$.

3. This transition represents the delay observed, waiting for some external event to be produced to allow leaving from the stable state.

The semantics of a TPN may be defined as a transition system. Let be a vector v of size $n = |T|$ and with coefficients in $\mathbb{R}^+ \cup \mathcal{E}$: $v \in (\mathbb{R}^+ \cup \mathcal{E})^T$. $v(t_i)$ (for $i \in [1, n]$) represents a quantity of elapsed time related to the transition $t_i \in T$ (*local clock*[4] for t_i). The nil vector is $v_0 \in (0)^T$. A state q of the transition system is a pair $< M, v >$.

Définition 2. *The* timed transition system $< Q, \{q_0\}, \Sigma, \to>$ *of a marked* ε*-TPN is defined by:*

1) $q_0 =< M_0, v_0 > \in Q$*: the initial state;*

2) $Q \subseteq (\mathbb{N})^T \times (\mathbb{R}^+ \cup \mathcal{E})^T$*: the set of states (reachable from* q_0*);*

3) $\Sigma = T$*: the alphabet of the discrete transitions;*

4) $\to \subseteq Q \times (T \cup \mathbb{R}^{+*} \cup \mathcal{E}) \times Q$*: the relation of the timed and instantaneous transitions:*

a) *a timed transition is such as:*

i) $\exists d \in \mathbb{R}^{+*}$ *(non-infinitesimal time elapse),* $< M, v > \xrightarrow{d} < M, v' >$ *iff :*

$$\begin{cases} \nexists t \in En(M) \cap T_{\mathcal{E}_0} \\ v' \overset{\text{def}}{=} v + d \\ \forall t_k \in En(M) \Rightarrow v'(t_k) \leq LD(t_k) \end{cases}$$

ii) $\exists d \in \mathcal{E}$ *(infinitesimal time elapse),* $< M, v > \xrightarrow{d} < M, v' >$ *iff :*

$$\begin{cases} \exists t \in En(M) \cap T_{\mathcal{E}} \\ \forall t_k \in T, \begin{cases} \textit{if } t_k \in T_{\mathcal{E}_0} \textit{ then } v'(t_k) \overset{\text{def}}{=} v(t_k) + d, \\ \quad \textit{else } v'(t_k) \overset{\text{def}}{=} v(t_k) \\ t_k \in En(M) \Rightarrow v'(t_k) \leq LD(t_k) \end{cases} \end{cases}$$

b) *an instantaneous firing* $t_i \in En(M)$, $< M, v > \xrightarrow{t_i} < M', v' >$ *iff:*

$$\begin{cases} t_i \in T_{\mathcal{E}_0} \vee (\nexists t_j \in En(M) \cap T_{\mathcal{E}_0}) \wedge (t_i \in T_T \cup T_\infty) \\ M' \overset{\text{def}}{=} M \backslash {}^\bullet t_i \cup t_i^\bullet \\ ED(t_i) \leq v(t_i) \leq LD(t_i) \\ \forall t_k \in T,\ v'(t_k) \overset{\text{def}}{=} \begin{cases} 0 \textit{ if } t_k \in En(M') \\ \quad \wedge (t_k \notin En(M \backslash {}^\bullet t_k) \vee t_k = t_i) \\ v(t_k) \textit{ else} \end{cases} \end{cases}$$

According to the definition 2, in a given state q, when some transitions in the set $T_{\mathcal{E}_0}$ are enabled, one is fired instantaneously if its clock reached its fixed delay, or the least infinitesimal delay is observed among the enabled transitions in $T_{\mathcal{E}_0}$ to make a transition fireable; meanwhile, clocks for enabled transitions in $T_T \cup T_\infty$ do not change. Otherwise, only transitions in $T_T \cup T_\infty$ are enabled, and the common semantics for TPN is applied: a transition t must be fired instantaneously if its clocks reaches $LD(t)$, otherwise a wait delay d may be observed for each transition provided that it does not increase some clock beyong its LD, or any transition is fired if its clock value is inside the static interval. After a firing, the clocks value of the transitions $En(M')$ are updated in accordance with the standard semantics.

4. The clock of an unenabled transition does not change (and does not mind). Indeed, such a clock value is not taken into account to decide equality between two states.

During a reaction phase, it happens that parallel firings and variable updatings are computed in the same instant, constituting a *step* of fired transitions in the set $T_{\mathcal{E}_0}$. In the sequel, such a step is considered in only one interleaving of its firings, in order to reduce state explosion when computing the state space.

This semantics definition is different from the one adopted in [11] where priority is not given to firings in $T_{\mathcal{E}_0}$.

2.4. Enhancements on definitions in [11]

With the new definitions about ε-TPN in this section, some few changes have to be considered about the definitions and interpretations given in [11]. First, as a minor change, in the former phase sequencer (Fig. 1(a)), $ED(Change\_input)$ is now replaced by 0 (Fig. 1(b)): this trick was used to avoid possible firing of this transition concurrently with those in $T_{\mathcal{E}_0}$, which is useless now since this transition of T_∞ type has lesser priority.

Second, in [11], when only transitions in $T_T \cup T_\infty$ are fireable, it is possible of the occurrence of an input event in T_∞ to be followed by a firing in T_T concurrently with some transitions in $T_{\mathcal{E}_0}$, meaning some possible interference between the beginning of a reaction phase and an external event occurrence[5]. The new semantics of ε-TPN avoids such a case, by always giving the priority to the reaction phase.

3. State space construction

For reminder, the specificity of an ε-TPN is simulating the behaviour of a grafcet as a synchronous modeling language: the execution is an infinite alternation of stimulus (external event) followed by a reaction phase (called *evolution*) while no deadlock occurs, and a reaction may consist of iterated firings (which are sequential and/or concurrent) constituting a *firing stage*.

Ideally, a reaction must be made of a finite number of firings (meaning no livelock), and the possible interleavings of their concurrent and synchronous firings should lead up to the same state. The state space construction should cash in on this peculiarity to limit the potential state explosion, while providing an unexpensive way to check this expectation (see Subsection 3.2).

The state space construction of a TPN is classically based on abstractions of the timed transition system (definition 2), in shape of transition systems (without time on the transitions) called state class graphs (SCG): the nodes are state classes (which are generally agglomerations of an infinite number of states) with dense time. According to the purposes of state space generation, there is several kind of abstractions [3]. Here, the focus is on the linear SCG [2], knowing that the other types of abstraction may be deduced without difficulty from this one.

3.1. Computing a state class of an ε-TPN

A state class C is a couple of a marking M and a clock domain D (meaning a conjunction of time constraints characterizing the clock values of the enabled transitions). We denote by τ (resp. τ_i), the clock variable of a transition t (resp. t_i) appearing in a domain. The initial class is $C_0 = (M_0, D_0)$, such as $D_0 = \bigwedge_{t \in En(M_0)} \tau = 0$.

5. At the same time [11], when a timed event in T_T occurs firstly, it cannot be followed by the event $Change\_input$ (with interval $[\varepsilon_0, \infty[$) in concurrency with a firing in $T_{\mathcal{E}}$, which is a bit contradictory.

The algorithm 1 (in Annex A) describes the construction of the SCG for an ε-TPN.

For computation of class bounds, with ϵ delays, the supplementary arithmetic about time quantities is obvious. For $i, j \in \mathbb{Z}^*$ ($\varepsilon_n = -\varepsilon_{-n}$ if $n < 0$) and $d \in \mathbb{R}^*$: $d \pm \varepsilon_i = d$, $\infty \pm \varepsilon_i = \infty$, $\varepsilon_i \pm 0 = \varepsilon_i$, and $\varepsilon_i \pm \varepsilon_j = \varepsilon_{i \pm j}$.

For a transition enabled in a current class C (line 6), two scenarios are possible: either $En(M) \cap T_{\mathcal{E}_0}$ is empty (lines 8-15), or not (lines 17-27).

The first scenario which is the classic case, is reminded in Annex A.

The second scenario is specific to ε-TPN. The fireability check does not change just for the first firing (line 17). Computing D' (in four steps) changes slightly for every firing $t_f \in T_{\mathcal{E}}$ in this scenario. Indeed, an adjustement is necessary to express that such a firing is only possible after a discrete time : $d = \delta(t_f) - \tau_f$ will replace $d \geq 0$ at the first of the four steps. Moreover, for all $(t_i, t_j) \in T_{\mathcal{E}_0} \times (T_T \cup T_\infty)$ (with $t_i, t_j \in En(M)$), the constraint $\tau_i \leq \tau_j$ (conjunctive with the previous one) should be added at the second step, to express that an enabled transition not in $T_{\mathcal{E}_0}$ may not occur before anyone in $T_{\mathcal{E}_0}$.

After the first firing of the second scenario, iterated firings (lines 19-21 of the algorithm 1) are applied, supposing that synchronous firings lead to the same state whatever is the interleaving used. Thus, the fireability check may be simplified for each of the iterated firings: having $\tau_f = \delta(t_f)$ is obviously sufficient to infer a positive test. For each reached class C'' (line 20), the domain D'' is computed as after the first firing (line 18). Since intermediate state classes of iterated firings should not be stored (those states being just particular to the current interleaving), only the final reached class C' is saved (lines 23-26) and the transition to reach C' is the multiset of the iterated firings (line 27).

More informations are given about the algorithm 1 in Annex A. An application is given in Annex C.

3.2. Consistency check of an iterated firing stage

The line 22 of the algorithm 1 checks (maybe by a function) if the state reached by the *stage* T_t (firings done in the same instant) does not depend on the particular interleaving which was executed. Else, the problem should be reported and it means that the model has to be improved: for instance, contradictory orders from concurrent parts of the system controller may exist (set and reset the same output in two concurrent stored actions for example). Concurrency is a factor of explosion: for n concurrent firings, there is 2^n states covered by the potential $n!$ interleavings.

In case of strong concurrency, partial order techniques may be employed to cope with the explosion. Especially, the unfolding method [10] which will not interleave concurrent events forming a stage, may be used from the state just before the first firing (line 17) as the initial state. This is eased by the finiteness of the executions of a firing stage (in spite of the presence of inhibitor and read arcs). Knowing the expected unique final state (from the current interleaving), and knowing the transitions which may be fired in the same instant (i.e. verifying $\tau = \delta(t)$), the unfolding algorithm may be adapted (and based on the solution to the coverability problems described in [10]) to answer if any other final state may appear.

4. Conclusion

In this paper, the syntax and the operational semantics of ε-TPN are formally defined, and the construction of the corresponding state class graph taking into account the ε delays is presented. That is complementary to the works in [11], by now preventing an overlap between an evolution phase and the occurrence of some external event, and by admitting only one external event before the subsequent reaction. This is achieved by introducing priority: the transitions for reaction ($T_\mathcal{E}$) have priority on the transitions for external events (T_T for quantitative timing events and T_∞ for input events).

An advantage of the proposed state space construction is the efficient elimination of the explosion due to concurrency during reactions, where no branching is displayed. To cope with the explosion caused by a multiplicity of input events, the tracks may consist in abstracting the states of inputs with no incidence on the subsequent reactions (while computing the SCG), and/or in modelling theirs dynamics to restrain the possible input changings.

Other perspectives are conceivable. One is to extend the translation rules to take into account hierarchy (macrostep, enclosure and forcing) in the grafcet: we hope that the more general definitions given in this paper will help to achieve this goal. Another one is to propose a model-checking framework for the grafcets, suitable to the specificity of the translation into ε-TPN.

5. References

[1] 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.

[2] B. Berthomieu and F. Vernadat. State class constructions for branching analysis of time Petri nets. In *TACAS*, pages 442–457, 2003.

[3] H. Boucheneb and R. Hadjidj. CTL* model checking for time Petri nets. *Theor. Comput. Sci.*, 353(1):208–227, 2006.

[4] G. Bucci and E. Vicario. Compositional validation of time-critical systems using communicating time Petri nets. *IEEE Trans. Softw. Eng.*, 21(12):969–992, 1995.

[5] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer Publishing Company, Incorporated, 2nd edition, 2010.

[6] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.

[7] D. L'Her, P. Le Parc, and L. Marcé. Proving sequential function chart programs using timed automata. *Theoretical Computer Science*, 267(1-2):141–155, 2001.

[8] IEC 60848. Grafcet specification language for sequential function charts. Technical report, International Electrotechnical Commission, 2013.

[9] IEC 61131-3. Programmable controllers - part 3: Programming languages. Technical report, International Electrotechnical Commission, 2013.

[10] K. L. McMillan. A technique of state space search based on unfolding. *Form. Methods Syst. Des.*, 6(1):45–65, 1995.

[11] M. Sogbohossou and A. Vianou. Formal modeling of grafcets with Time Petri nets. *IEEE Transactions on Control Systems Technology*, 23(5):1978–1985, Sept 2015.

[12] N. Wightkin, U. Buy, and H. Darabi. Formal modeling of Sequential function Charts with Time Petri nets. *IEEE Transactions on Control System Technology*, 19(2):455–464, 2011.

A. Algorithm to generate the state class graph of a ε-TPN

The algorithm 1 describes the construction of the state class graph for an ε-TPN.

```
Input: marked $\varepsilon$-TPN;
Output: sets $Classes$ and $Transitions$ representing the state class graph;
$Classes := \{C_0\}; Transitions := \{\}$;
Stack the initial class $C_0$;
while the stack is not empty do
    Unstack (LIFO) the class $C = (M, D)$;
    if $\nexists t \in En(M) \cap T_{\mathcal{E}_0}$ then
        foreach transition t fireable from the class C do
            Compute the successor class $C'$;
            if $C' \notin Classes$ then
                Stack $C'$;
                $Classes := Classes \cup \{C'\}$;
            end
            $Transitions := Transitions \cup \{(C, \{\{t\}\}, C')\}$;
        end
    else
        Find any fireable transition $t$ from $C$; $T_t := \{\{t\}\}$;
        Compute the successor class $C'$;
        while $\exists t' \in En(M') \cap T_{\mathcal{E}_0}$ fireable in the same instant as t do
            Compute the successor class $C''$ from $C'$; $T_t := T_t \cup \{\{t'\}\}$; $C' := C''$;
        end
        Make consistency check of the stage ($T_t$) between $C$ and $C'$;
        if $C' \notin Classes$ then
            Stack $C'$;
            $Classes := Classes \cup \{C'\}$;
        end
        $Transitions := Transitions \cup \{(C, T_t, C')\}$;
    end
end
```

Algorithm 1. *Construction of the state class graph of an ε-TPN*

For a transition enabled in a current class C (line 6), two scenarios are possible: either $En(M) \cap T_{\mathcal{E}_0}$ is empty (lines 8-15), or not (lines 17-27).

In the first scenario which is the classic case [3], fireability of each enabled transition is tested, and when it is fireable, the successor class C' is computed. A fireability check of a transition $t_f \in (T_T \cup T_\infty)$ to fire includes the test of the current *domain consistency* achieved this manner: $D \wedge (d \geq 0) \wedge (ED(t_f) \leq \tau_f + d) \wedge (\bigwedge_{t \in En(M)} (\tau + d \leq LD(t)))$. Then, the new domain D' is computed in the following steps:

1) Initially, D' is $D \wedge d \geq 0$; then, replace each variable τ in D' by $\tau - d$;

2) Add the constraints: $ED(t_f) \leq \tau_f \wedge (\bigwedge_{t \in En(M)} \tau \leq LD(t))$;

3) Eliminate τ_f, d and all τ_c such as $t_c \in En(M) \wedge t_c \notin En(M \setminus {}^\bullet t_c)$;

4) Add the constraint $\tau = 0$ for each t newly enabled by M' (that is: $t \in En(M') \wedge (t \notin En(M \setminus {}^\bullet t) \vee t = t_f)$).

For this classic scenario where no transition in $T_{\mathcal{E}}$ is enabled, time progress is only dense and ε delays don't mind: each bound in a class domain with this kind of value should be replaced by the value 0.

Line 17 (in the second scenario) supposes that a transition will be always found to continue the execution of the program; it is guaranteed by the phase sequencer (Fig. 1) which is an infinite loop execution (no dead state is possible), even if an evolution may be *void*: the transition $Evolution\_end$ may be fired without firing any other reaction transition (when the previous occurred external event does not cause a real reaction phase).

The **while** loop (line 19) may potentially be infinite when a reaction phase is not finite. This part of the algorithm can be easily modified by saving apart the states C'' to detect and report the infinite loop in order to fix the problem in the model.

It should be noted that, by abstracting the **if** line 7 and the **else** block from the lines 16 to 27 which are specific to an ε-TPN, the classic algorithm is ensued.

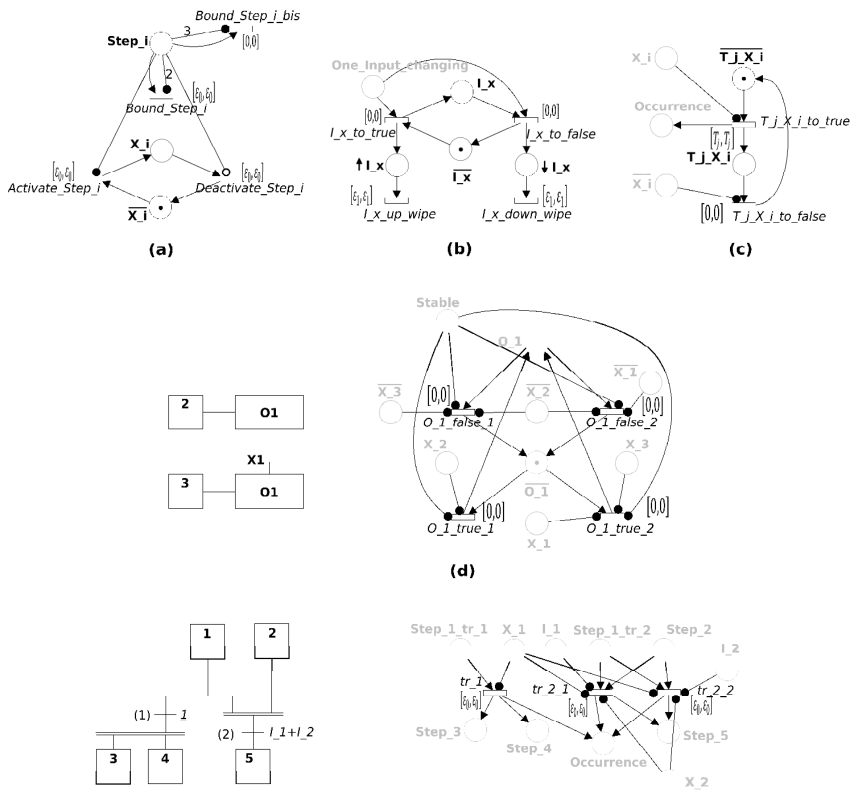

Figure 2. *Examples of translated elements of grafcet (according to [11])*

B. Translation of elements of grafcet into TPN [11]

The given elements of translation here are respectively (Fig. 2): a step i with its X_i state variables (Fig. (a)), an input with its rising and falling edges (Fig. (b)), a timed variable T_j/X_i (Fig. (c): the transition $T\_j\_X\_i\_to\_true$ is the only one type expressing an external event, with the transition $Change\_input$ Fig. 1(a) in a phase sequencer), an example of continuous and conditional actions (Fig. (d)), and an example of transitions with simultaneous divergence and convergence (Fig. (e)). Let notice that the elements in gray are added in a previous phase of the translation.

In this paper, no change concerns the translation rules proposed in [11], except slighty the phase sequencer as developped in the subsection 2.1.

C. Application

As an illustration of the newly defined semantics, an example of grafcet is provided Fig. 3(a), its translation into ε-TPN is provided Fig. 3(b) and the construction of its state class graph (SCG) is sketched Fig. 4.

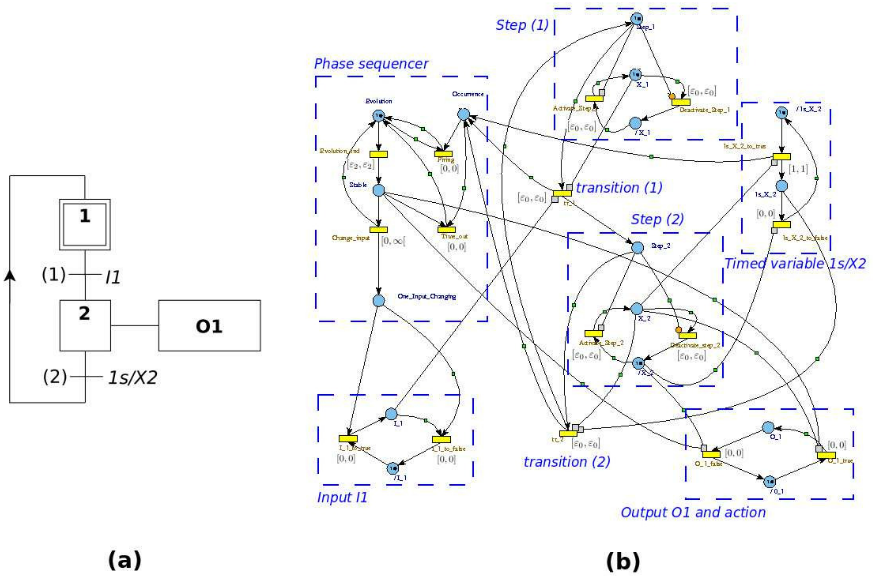

Figure 3. *An example of grafcet (a) and its translation into ε-TPN (b)*

The resulting ε-TPN (edited with Roméo[6] with superimposed images to specify intervals with ϵ delays) gets some simplified modules since the grafcet example is a safe model, and all litterals (such as the edges about the input I_1) are not useful. In general, the spatial complexity of the translation is globally polynomial with the number of nodes (steps and transitions), variables or literal terms of a grafcet [11].

6. Roméo, a tool for Time Petri Nets analysis: `romeo.rts-software.org`

The ε-TPN Fig. 3(b) is a safe model. Fig. 4 is just the initial part of the SCG (the twelve first classes). Each class is represented in a box with the upper part enumerating the marked places and the lower part giving the enabled transitions with their clock interval (the clock domain is so simplified).

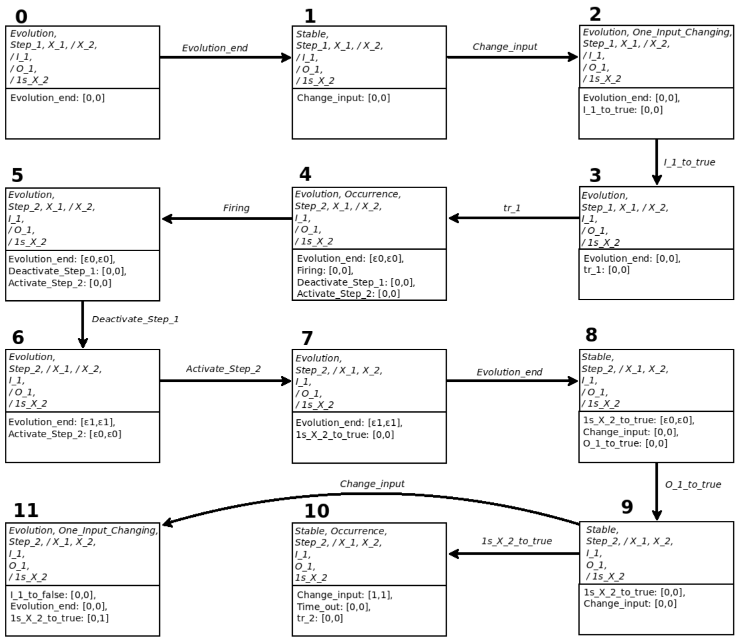
0
Evolution, Step_1, X_1, / X_2, / I_1, / O_1, / 1s_X_2
Evolution_end: [0,0]
Evolution_end
1
Stable, Step_1, X_1, / X_2, / I_1, / O_1, / 1s_X_2
Change_input: [0,0]
Change_input
2
Evolution, One_Input_Changing, Step_1, X_1, / X_2, / I_1, / O_1, / 1s_X_2
Evolution_end: [0,0], I_1_to_true: [0,0]
I_1_to_true
3
Evolution, Step_1, X_1, / X_2, I_1, / O_1, / 1s_X_2
Evolution_end: [0,0], tr_1: [0,0]
tr_1
4
Evolution, Occurrence, Step_2, X_1, / X_2, I_1, / O_1, / 1s_X_2
Evolution_end: [ε0,ε0], Firing: [0,0], Deactivate_Step_1: [0,0], Activate_Step_2: [0,0]
Firing
5
Evolution, Step_2, X_1, / X_2, I_1, / O_1, / 1s_X_2
Evolution_end: [ε0,ε0], Deactivate_Step_1: [0,0], Activate_Step_2: [0,0]
Deactivate_Step_1
6
Evolution, Step_2, / X_1, / X_2, I_1, / O_1, / 1s_X_2
Evolution_end: [ε1,ε1], Activate_Step_2: [ε0,ε0]
Activate_Step_2
7
Evolution, Step_2, / X_1, X_2, I_1, / O_1, / 1s_X_2
Evolution_end: [ε1,ε1], 1s_X_2_to_true: [0,0]
Evolution_end
8
Stable, Step_2, / X_1, X_2, I_1, / O_1, / 1s_X_2
1s_X_2_to_true: [ε0,ε0], Change_input: [0,0], O_1_to_true: [0,0]
O_1_to_true
9
Stable, Step_2, / X_1, X_2, I_1, O_1, / 1s_X_2
1s_X_2_to_true: [0,0], Change_input: [0,0]
1s_X_2_to_true
Change_input
10
Stable, Occurrence, Step_2, / X_1, X_2, I_1, O_1, 1s_X_2
Change_input: [1,1], Time_out: [0,0], tr_2: [0,0]
11
Evolution, One_Input_Changing, Step_2, / X_1, X_2, I_1, O_1, / 1s_X_2
I_1_to_false: [0,0], Evolution_end: [0,0], 1s_X_2_to_true: [0,1]

Figure 4. *The initial part of the SCG of the example Fig. 3(b)*

Altogether, 35 classes are covered (with 38 transitions) by the algorithm 1. Abstracting the intermediate classes during iterated firings eliminates 12 ones; for the given initial part: the class $\{C_4\}$ (resp. $\{C_6\}$, $\{C_8\}$) for the stage $\{\{tr\_1, Firing\}\}$ (resp. $\{\{Deactivate\_Step\_1, Activate\_Step\_2\}\}$, $\{\{Evolution\_end, O\_1\_to\_true\}\}$) do not appear in the set $Classes$. Abstracting the class C_6 avoids one of the two interleavings of the concurrent firings $Deactivate\_Step\_1$ and $Activate\_Step\_2$.

The SCG abstraction generated by Algorithm 1 holds however more informations than the first abstraction proposed in [11]: according to this last one, the class C_2 and the classes $\{C_4, C_5, C_6\}$ have to be abstracted too, and in the resulting macro-transition, only the firing $Change\_input$ (resp. tr_1) needs to be displayed for a behaviour verification, since other relevant informations are contained in the marking of the reached class C_3 (resp. C_7). With the same idea, the transition between C_7 and C_9 after the abstraction of C_8 only needs to carry the information $Evolution\_end$. Algorithm 1 is easily modifiable to enhance the abstraction in this way.

At last, in [11] was proposed a second and more compact abstraction[7] to only display the stable states of the grafcet. It will result in a SCG with only 4 classes and 7 transitions.

7. The goal of the first and the second abstractions in [11] was to only preserve the informations contained in the source grafcet.