Some remarks on "State Estimation and Fault Diagnosis of Labeled Time Petri Net Systems with Unobservable Transitions"

Zhou He, Zhiwu Li, Alessandro Giua, Francesco Basile, and Carla Seatzu,

Abstract

In this paper we comment on the algorithm proposed in the paper mentioned in the title to define and construct a graph, called *Modified State Class Graph* (MSCG), which summarizes all possible evolutions of a Time Petri net. We first show that under the assumptions mentioned in such a paper, the proposed graph could be infinite. Then, we underline the requirement of revising the notation and adding some information on certain edges of the graph. Finally, we remark that the current version of the algorithm does not consider all possible evolutions of the net system. In the final part of the manuscript we propose a revised algorithm for the definition and construction of the MSCG that overcomes all such limitations.

Published as:

Zhou He, Zhiwu Li, Alessandro Giua, Francesco Basile, and Carla Seatzu. "Some remarks on "State Estimation and Fault Diagnosis of Labeled Time Petri Net Systems with Unobservable Transitions"," *IEEE Trans. on Automatic Control*, Vol. 64, No. 12, pp. 5253-5259, 2019. DOI: 10.1109/TAC.2019.2910168.

Z. He is with the College of Mechanical and Electrical Engineering, Shaanxi University of Science and Technology, Xi'an 710021, China (email: hezhou@sust.edu.cn).

Z. W. Li is with the Institute of Systems Engineering, Macau University of Science and Technology, Taipa, Macau and also with School of Electro-Mechanical Engineering, Xidian University, Xi'an 710071, China (email: zhwli@xidian.edu.cn).

A. Giua is with the DIEE, University of Cagliari, Cagliari 09124, Italy (email: giua@unica.it).

F. Basile is with the Dipartimento di Ingegneria dell'Informazione, Ingegneria Elettrica e Matematica applicata, Università di Salerno, Italy (email: fbasile@unisa.it).

C. Seatzu is with the DIEE, University of Cagliari, Cagliari 09124, Italy (email: seatzu@unica.it).

I. INTRODUCTION

In a recent paper Basile *et al*. [1] introduced a graph called *Modified State Class Graph* (MSCG) to describe in a compact and exhaustive manner all possible evolutions of a labeled Time Petri net system. The definition of the MSCG was given in an algoritmic way, namely it was based on the algorithm used for its construction. In this paper we show that the algorithm presented in [1] to define and construct the MSCG is not correct. In particular:

- The assumptions in [1] do not ensure the finiteness of the MSCG, as claimed. We show this via a counterexample.
- The notation used to define the time spent by the net system in a given class (node) of the graph may affect the correctness of the way some constraints, used to describe the possible evolutions on the net system, are written.
- Finally, the algorithm in [1] does not guarantee, as claimed, that *all* possible evolutions of the net system are described in the MSCG.

In this Technical Note we provide a revised version of the algorithm in [1] that overcomes the above limitations. We also prove that, under the assumptions that the TPN is bounded and there exists no repetitive sequence that may fire in zero time, the MSCG constructed using the revised algorithm is finite, which is equivalent to prove that the algorithm terminates in a finite number of steps.

We conclude this section with three important remarks. First, the example proposed in [1] is correct since it does not violate the missing assumption, the problem in the notation does not appear there, and the MSCG actually provides all possible evolutions of the net system. Second, in [1] a procedure for the state estimation and fault diagnosis of labeled Time Petri nets is proposed based on the MSCG. The correctness of such results is not affected by the modifications introduced in the algorithm defining the MSCG. Indeed, the resulting MSCG mantains the structure and the features that were assumed when such results have been derived. Finally, the MSCG has been recently used in [2] to perform diagnosability analysis. Again, the example in [2] is correct and the results proposed there still apply when using the revised algorithm to construct the MSCG.

II. THE MODIFIED STATE CLASS GRAPH

A *Time Petri net* (TPN) is defined as a pair $N_d = (N, Q)$ where $N = (P, T, Pre, Post)$ defines the net structure and $Q : T \to \mathbb{R}_0^+ \times (\mathbb{R}_0^+ \cup \{\infty\})$ defines the set of *static* intervals associated with transitions [1]. In particular, P is the set of places, T is the set of transitions, and $Pre : P \to \mathbb{N}$ and $Post : P \to \mathbb{N}$ are the pre– and $post$– incidence functions that specify the arcs [3]. In the following, the static lower bound of transition t_i is denoted l_i, while its static upper bound is denoted as u_i.

A TPN N_d with a marking M_0 at the initial time instant $\tau_0 = 0$ is called a *marked* TPN, or a TPN system, and is denoted as $\langle N_d, M_0 \rangle$.

In this paper, as in [1], we assume that a labeling function $\bar{\mathcal{L}} : T \to L \cup \{\varepsilon\}$ assigns to each transition $t \in T$ either a symbol from a given alphabet L or the empty string ε. The label of a transition represents the output it produces when firing.

The *Modified State Class Graph* (MSCG) is a directed graph whose nodes are called *classes*. With each class is associated a *state* of the net, namely a reachable marking M and a set of inequalities Θ that define the timing constraints pertaining to all transitions enabled at M. Such inequalities depend on a certain number of variables, denoted Δ variables, which take into account how much time a transition has been enabled. The way such constraints are defined, as well as the way the Δ variables are initialized and updated, is clarified by Algorithms 1 and 2 in Section V. Edges are labeled as $(t, \gamma, \Delta \in [l^*, u^*])$, where t is the transition whose firing leads from the marking in the tail node to the marking in the head node; γ is the label associated with t, and $\Delta \in [l^*, u^*]$ is a constraint on the time spent in the tail node of the edge. The upper bounds l^* and u^* are functions of Δ variables in the path from the root node to the head node of the edge at hand. As discussed in Section V where Algorithms 1 and 2 are presented, a fourth entry is added on certain edges to link Δ variables from a class to the next one when this is needed.

The goal of the MSCG relative to a TPN system $\langle N_d, M_0 \rangle$ is that of providing a compact and exhaustive representation of all its possible evolutions. In more detail, a time-transition sequence $(t_{i_1}, \tau_1)(t_{i_2}, \tau_2) \ldots (t_{i_k}, \tau_k) \in (T \times \mathbb{R}_0^+)^*$ is firable at the initial marking if and only if the following two conditions hold: (1) there exists a path in the MSCG starting from the initial node, whose sequence of first entries in the edges is $t_{i_1} t_{i_2} \ldots t_{i_k}$; (2) the time instants τ_1, τ_2, $\ldots$, τ_k satisfy a series of constraints univocally defined, as explained in [1], by the inequalities Θ in the nodes along the path and by the third entries of the edges along the same path.

***Example** 1:* Consider the TPN system in Fig. 1. A part of its MSCG built according to Algorithm 1 in [1] is shown in Fig. 2.

The time-transition sequence $(t_1, 1)(t_3, 1.5)$ is firable at the initial marking and leads to marking $p_1 + p_2$ where t_1 and t_2 are enabled and have a firing delay satisfying the constraints $0 \leq \theta_1 \leq 2$ and $0 \leq \theta_2 \leq 0.5$, respectively. Such an evolution corresponds to the path $C_0 \xrightarrow{t_1, \varepsilon, \Delta_1 \in [0,2]} C_1 \xrightarrow{t_3, a, \Delta_3 \in [0, 2-\Delta_1]} C_2$ in the MSCG where $\Delta_1 = 1$ and $\Delta_3 = 0.5$, which imply $\tau_1 = \Delta_1 = 1$ and $\tau_2 = \Delta_1 + \Delta_2 = 1.5$.

On the contrary, the time-transition sequence $(t_1, 1)(t_3, 2.5)$ is not firable. This can be easily verified in the MSCG. Indeed, it logically corresponds to the path $C_0 \xrightarrow{t_1, \varepsilon, \Delta_1 \in [0,2]} C_1 \xrightarrow{t_3, a, \Delta_3 \in [0, 2-\Delta_1]} C_2$. However, only $\Delta_1 = 1$ satisfies the timing constraints, while $\Delta_3 = 1.5$ is not an admissible value since, according to the constraint in the second edge of the path, it should be $\Delta_3 \in [0, 2 - \Delta_1] = [0, 1]$. $\square$

III. Some remarks on Algorithm 1 in [1]

In this section we describe in detail the three issues mentioned in the introduction, which affect Algorithm 1 in [1].

In the following, in accordance with the notation in [1], the generic node C_e is labeled with (M_e, Θ_e) where M_e is the marking that characterizes it and Θ_e is the set of constraints that characterize the admissible values of the timing delays θ_i, for all transitions t_i enabled at M_e. Finally, we denote as $\Delta^{(e)}$ the set of symbolic variables Δ's involved in Θ_e.

A. The resulting MSCG may be infinite

Proposition 5 in [1] claims that, under appropriate assumptions, the Modified State Class Tree (MSCT) built according to Algorithm 1 in [1] (and consequently the MSCG) is finite. This may not be correct in some cases, as shown by the following simple counterexample.

***Example** 2:* Consider again the TPN system in Fig. 1. The repetitive sequence $t_1 t_3$ may fire arbitrarily often in zero time, while keeping unaltered the marking of place p_2. This leads to an infinite number of classes as shown in Fig. 2. This can be explained in detail as follows. Whenever a new firing of either t_1 or t_3 occurs, a new constraint is defined involving a new Δ variable which takes into account the time delay in which the firing could have occurred with respect to the previous transition firing. This leads to the definition of a new class in the graph. Since each Δ variable may take a zero value, constraints involving an ever increasing number of different variables Δ (and consequently also new classes) can be defined. As a result, the MSCG has an infinite number of classes. □

B. A problem in the notation

In [1] the generic symbolic variable Δ_i is denoted as a function of the transition t_i appearing in the edge (see e.g., Fig. 2). This may lead to an improper formalisation when transition t_i is labeling two or more edges along a path and produces multiple occurences of the same variable Δ_i in a constraint.

This appears evident looking again at the MSCG in Fig. 2. Here variable Δ_1 appears twice in the constraint

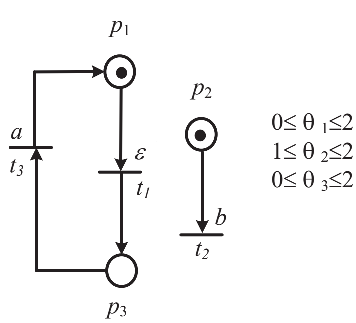

Fig. 1. The TPN system considered in Example 1.

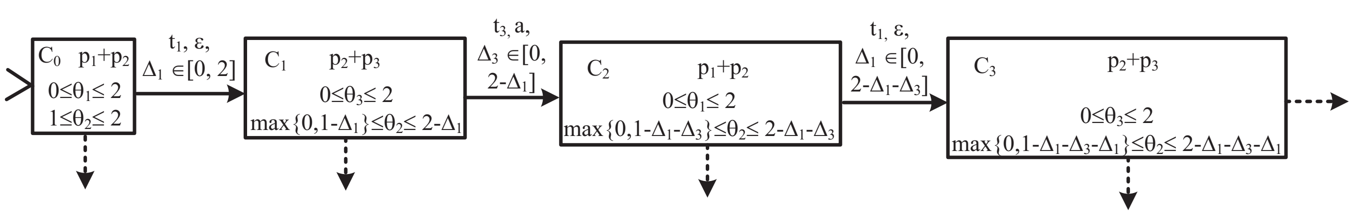

Fig. 2. A part of the MSCG associated with the TPN system in Fig. 1 constructed according to Algorithm 1 in [1].

on θ_2 in class C_3: $\max\{0, 1-\Delta_1-\Delta_3-\Delta_1\} \leq \theta_2 \leq 2-\Delta_1-\Delta_3-\Delta_1$. Apparently such a constraint could be rewritten as $\max\{0, 1-2\Delta_1-\Delta_3\} \leq \theta_2 \leq 2-2\Delta_1-\Delta_3$, which is not correct. Indeed, the same symbol Δ_1 denotes two different symbolic variables. One of these variables (let us call it Δ_1') is subject to the constraint $\Delta_1' \in [0, 2]$ labeling the edge from C_0 to C_1. The second variable (let us call it Δ_1'') is subject to the constraint $\Delta_1'' \in [0, 2-\Delta_1'-\Delta_3]$ labeling the edge from C_2 to C_3.

C. *Some possible evolutions could be missing*

The last issue concerns the representation of all possible evolutions of the net system. When a node is explored, while constructing the MSCG, it may happen that a logically enabled transition cannot fire because its occurrence will be necessarily preempted by the firing of another enabled transition. In fact, in each node C_e the admissible firing delays of enabled transitions satisfy a linear constraint set that depends on the path from the root node to node C_e. If for all paths leading to C_e the lowest admissible value for the firing delay of a logically enabled transition t is greater than the highest admissible value of another enabled transition, then t will never be able to fire from C_e. In this case we say that transition t is preempted at node C_e. This leads to the key definition of deficient node and deficient transition at a certain node.

Definition 1: A node C_e is *deficient* if there exists at least one transition that is logically enabled at M_e, but whose firing will be necessarily preempted by another transition.

Transitions that make node C_e deficient are said *deficient at node* C_e. ■

According to Algorithm 1 in [1] each node is explored only once, when it is tagged "new". Therefore, if a transition is deficient when the node is explored, it always remains deficient at that node and no output arc corresponding to it will be created. This may lead to neglect some possible evolutions of the system. Indeed, when new "duplicate" nodes are created[1], they could generate new paths leading to nodes that have already been explored. If this happens at a node that is deficient, such a node should be explored again to take the new paths into account and determine if some transitions that were previously classified as deficient at such a node, could now be enabled.

***Example** 3:* Consider the TPN system in Fig. 3. The MSCG, constructed according to the Revised Algorithm 1 illustrated in detail in the following Section V, is reported in Fig. 4.f, while the main steps of the construction of the MSCT are shown in Figs. 4.a–e.

Let us focus on node C_1. It is added when examining node C_0 (Fig. 4.a). When first analyzed, node C_1 is tagged "deficient" since only one (t_4) of the two transitions logically enabled at marking $p_2 + p_4$ (t_4 and t_5), may actually fire (Fig. 4.b). In particular, t_5 cannot fire since the smallest admissible value of its lower bound is equal to 2 (it corresponds to $\Delta_0 = 1$) and is thus greater than the upper bound of t_4, which is equal to 1.

Then, when node C_2 is examined, node C_1', which is a duplicate of C_1, is added to the tree (Fig. 4.c). Node C_1' will be merged with C_1 when constructing the graph (Fig. 4.f), thus the path leading to it also leads to node

[1]The tag "duplicate" is used here to avoid exploring nodes already considered. Duplicate nodes are then merged when constructing the graph starting from the tree.

C_1 in the graph. Therefore, differently from what is done in [1], C_1 should be examined again to see if some of the deficient transitions (t_5 in this case) cease being deficient thanks to the new path. In the case at hand, this is actually the case and node C_1 ceases being deficient as highlighted in Fig. 4.e. Indeed, according to the new path that leads to C_1' (and thus to C_1 in the graph) the smallest admissible value of the lower bound of the delay of t_5 is equal to 1 (which corresponds to[2] $\Delta_2 = 2$) and is thus equal to the upper bound of t_4, which is also equal to 1.

We finally notice that nodes C_0 and C_2 are tagged "deficient" when first examined, and remain deficient for ever. In particular, no further path leading to them is created while constructing the MSCT, so they are not examined further. ■

IV. Revised notation, equivalent nodes, and isomorphism relationship

In this section we first provide a solution to the problem in the notation discussed in the previous Subsection III-B. Then, we formalize the notion of equivalent nodes (provided only in a discursive way in [1]) and clarify the requirement of including additional information on some edges of the MSCG.

An easy solution to the problem in the notation consists in defining symbolic variables Δ's on the edges exiting from a certain node, as a function of the node itself. In this new formalism, given a node C_e all variables Δ's associated with output edges from C_e are denoted as Δ_e, regardless of the transition with which they are associated. This is also consistent with the physical meaning of such variables: Δ_e represents the time spent by the net system in class C_e. Clearly, depending on the transition associated with the edge, the constraints to which they are subject, are typically different.

This revision in the notation requires a modification in the algorithm to construct the MSCT. To explain this (and the proposed solution) we preliminarily provide a formal definition of the notion of equivalent classes.

Definition 2: Given a node C_e labeled with (M_e, Θ_e) and a node C_q labeled with (M_q, Θ_q), C_e and C_q are said to be *equivalent* if $M_e = M_q$, and the set of constraints Θ_e and Θ_q are isomorphic, i.e., there exists a bijective function that defines a one to one mapping between the symbolic variables Δ's in $\Delta^{(e)}$ and the

[2]Looking at the edge from C_2 to C_1', we read that $\Delta_2 \in [0.2]$. In the same edge we find out (as the fourth label) an isomorphism relationship, namely $\Delta_0 := \Delta_2$. As explained in detail in the following, this means that in all the constraints in the input node of the edge, namely C_1', Δ_0 should be replaced by Δ_2.

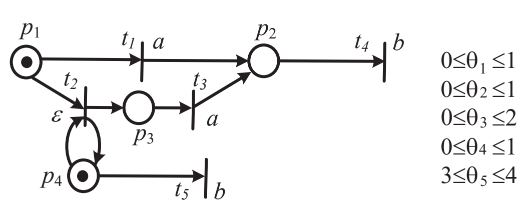

Fig. 3. The TPN system considered in Example 3.

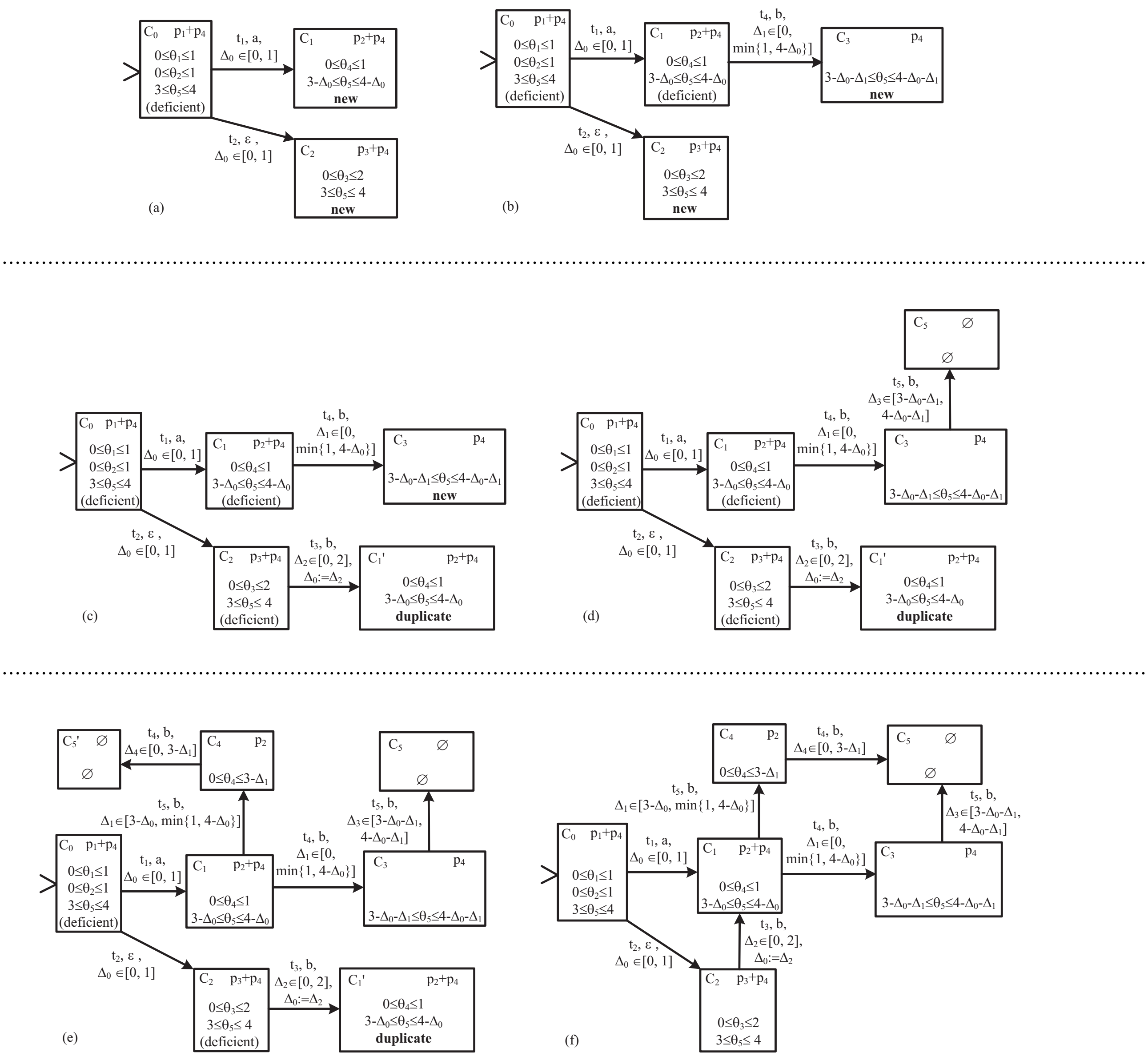

Fig. 4. The MSCG (Fig. (f)) of the TPN system in Fig. 3, built according to Revised Algorithm 1, and the most significant intermediate steps in its construction (Figs. (a) to (e)).

symbolic variables Δ's in $\Delta^{(q)}$, so that the set of constraints Θ_e is converted into the set of constraints Θ_q, and viceversa.

In the following we denote by $f_{eq} : \Delta^{(e)} \to \Delta^{(q)}$ the one to one mapping between the variables in the set $\Delta^{(e)}$ and the variables in the set $\Delta^{(q)}$. ■

When constructing the MSCT, several nodes belonging to the same class of the equivalence relation previously defined may be encountered. The first of this node, say C_e, is considered as representative of the class and the possible evolutions from it will be further explored. All other nodes are labeled "duplicate" and will not be further explored: thus they will be leaves of the MSCT. In addition, to keep track of the isomorphism between a duplicate node C_q and its representative node C_e, we add to the label of the edge entering C_q a fourth entry $\Delta^{(e)} := f_{eq}(\Delta^{(q)})$.

***Example** 4:* Consider again the TPN system in Fig. 3. As already discussed in Example 3, Fig. 4 illustrates the main steps in the construction of the MSCT using Revised Algorithm 1, as well as the resulting MSCG.

Let us focus on node C_1. It is added during the first iteration of Algorithm 1, when examining node C_0 (Fig. 4.a). Then, when exploring node C_2, a node equivalent to C_1, namely C_1', is computed. Therefore, C_1' is tagged "duplicate" (Fig. 4.c) and the isomorphism relationship $\Delta_0 := \Delta_2$ is added as the fourth entry of the edge from C_2 to C_1'.

Node C_1 is then merged with node C_1' when constructing the MSCG strarting from the MSCT (Fig. 4.f) and the isomorphism relationship mentioned above (appearing in the edge from C_2 to C_1 in the MSCG) specifies how to write the constraints involving the timing delays of transitions t_4 and t_5 when following the path $C_0 \xrightarrow{t_2,\varepsilon,\Delta_0\in[0,1]} C_2 \xrightarrow{t_3,b,\Delta_2\in[0,2]} C_1$ in the MSCG. In particular, it specifies that all constraints written in terms of Δ_0 in node C_1 and in its output edges, should be rewritten replacing Δ_0 with Δ_2 when C_1 is reached via C_2. ■

V. The revised algorithm for the construction of the MSCT

In this section we provide a revised version of Algorithm 1 in [1] and prove that, under appropriate assumptions, the resulting MSCG is finite. For the sake of clarity and conciseness, the proposed algorithm uses a function, named "Look for new nodes", which is called twice with different input arguments. In more detail, Revised Algorithm 1 summarizes the main steps of the new algorithm for the construction of the MSCT, while Algorithm 2 summarizes the main steps of the function "Look for new nodes". Such a function has two input arguments: a node C_k and a set of duplicate nodes $Duplicate$. Its role is that of looking for new nodes in the tree starting from the exploration of node C_k. Furthermore, as discussed in more detail in the rest of the section, the second argument takes different values depending on the fact that C_k is explored for the first time, or it has been already explored.

Note that in Revised Algorithm 1 and in Algorithm 2, $\mathcal{A}(M)$ denotes the set of transitions logically enabled at M.

A. Explanation of Algorithm 1

Algorithm 1 starts by defining as root node of the tree the class C_0, which is initially tagged "new".

Then two sets are introduced and initialized at the empty set (Steps 2 and 3, respectively):

- Def is the current set of deficient nodes;
- Dup is the current set of duplicate nodes.

The above two sets are then updated while executing Algorithm 2.

Revised Algorithm 1: Construction of the Modified State Class Tree

Input: A labeled TPN system

Output: Its Modified State Class Tree.

1 **Initialize:** The root node C_0 is labeled with the initial marking M_0 and a set of inequalities Θ_0 defined as follows: $\forall t_i \in \mathcal{A}(M_0)$, let $l_i^0 \leq \theta_i \leq u_i^0$ where $l_i^0 = l_i$ and $u_i^0 = u_i$. Tag the root node "new".

2 Let $Def = \emptyset$.

3 Let $Dup = \emptyset$.

4 **while** *a node tagged "new" exists* **do**

5   select a node C_k tagged "new";

6   let $T_k = \mathcal{A}(M_k)$;

7   apply function "Look for new nodes (C_k, Dup)";

8   untag node C_k.

9   **while** *no node tagged "new" exists and there exists a node* $\mathcal{C}_k \in Def$ *with* $\mathcal{L}_k \neq \emptyset$ **do**

10   **for** *all* $C_k \in Def$ *with* $\mathcal{L}_k \neq \emptyset$ **do**

11   apply function "Look for new nodes $(C_k, \mathcal{L}_k)$".

While nodes tagged "new" exist, one of such nodes is selected. Let's call it C_k (Step 5). A set of transitions T_k is associated with node C_k and it is initialized at $T_k = \mathcal{A}(M_k)$ (Step 6).

- Set T_k contains the deficient transitions at node C_k. More precisely, the first time node C_k is selected at Step 5 of Algorithm 1, namely when C_k is tagged "new", T_k is initialized at $\mathcal{A}(M_k)$ (Step 6 of Algorithm 1). Then, whenever Algorithm 2 (Step 3) finds out that a transition in T_k is not deficient at node C_k, such a transition is removed from set T_k (Step 4 of Algorithm 2). Therefore, once C_k has been explored, T_k contains the set of transitions that are currently deficient at node C_k. As a result, C_k is classified deficient at a certain iteration if and only if at that iteration it is $T_k \neq \emptyset$.

At Step 7, Algorithm 2 is executed having as input arguments C_k and Dup. In simple words, new nodes are eventually created in the tree exploring node C_k and taking into account the whole current set of duplicate nodes. After that, node C_k is untagged (Step 8).

The second while loop is based on the definition of set $\mathcal{L}_k$ which has the following physical meaning:

- $\mathcal{L}_k$ is associated with the generic deficient node C_k and contains the duplicate nodes that have been created in the tree after node C_k has been explored the last time.

The second while loop applies when no node tagged "new" exists, but there exists at least one deficient node C_k whose set $\mathcal{L}_k$ is not empty: this means that C_k is not new, but, since it is deficient and some duplicate nodes have been added to the tree after it has been examined the last time, it could happen that some transitions that are currently deficient at node C_k, cease being deficient thanks to the new paths that such duplicate nodes have

Algorithm 2: Function "Look for new nodes ($C_k, Duplicate$)"

while $T_k \neq \emptyset$ **do**

  select $t_i \in T_k$;

  if $max\{0, l_i^k\} \leq min_{j:\ t_j \in T_k}\{u_j^k\}$ *at least along one path in* $Paths(C_k, Duplicate)$, **then**

    let $T_k = T_k \setminus \{t_i\}$;

    let $M_q = M_k + C(\cdot, t_i)$ be the marking reached from M_k firing t_i;

    for *all transitions* $t_r \in \mathcal{A}(M_q)$ **do**

      if $t_r \in T_k$ *(i.e., if* t_r *was already enabled at class* C_k*) and the firing of* t_i *does not first disable* t_r *and then re-anables it* **then**

        let

$$l_r^q = l_r^k - \Delta_k, \quad u_r^q = u_r^k - \Delta_k$$

      else

        let

$$l_r^q = l_r, \quad u_r^q = u_r.$$

      Add a new node C_q labeled with marking M_q and a set of inequalities Θ_q defined as follows: $\forall t_r \in \mathcal{A}(M_q)$, let $max\{0, l_r^q\} \leq \theta_r \leq u_r^q$.

      Add an edge from C_k to C_q labeled

$$t_i, \bar{\mathcal{L}}(t_i), \Delta_k \in [max\{0, l_i^k\}, \\ min_{j:\ t_j \in T_k}\{u_j^k\}].$$

    if *there already exists a set of nodes equivalent to* C_q *in the tree* **then**

      tag node C_q "duplicate",

      let C_e be the node equivalent to C_q not tagged "duplicate", add the following forth entry to the label of the edge from C_k to C_q: $\Delta^{(e)} := f_{eq}(\Delta^{(q)})$.

      for *all* $C_j \in Def$ **do**

        $\mathcal{L}_j = \mathcal{L}_j \cup \{C_q\}$.

      Let $Dup = Dup \cup \{C_q\}$.

    else

      tag it "new".

if $T_k \neq \emptyset$ *and* $C_k \notin Def$ **then**

  Let $Def = Def \cup \{C_k\}$.

if $T_k = \emptyset$ *and* $C_k \in Def$ **then**

  Let $Def = Def \setminus \{C_k\}$.

if $C_k \in Def$ **then**

  Let $\mathcal{L}_k = \emptyset$.

generated. The role of this while loop is that of investigating if this actually occurs and, if such is the case for some node C_k, apply function "Look for new nodes" with input arguments node C_k and its corresponding set $\mathcal{L}_k$.

B. Explanation of Algorithm 2

Algorithm 2, as the name of the function highlights, looks for new nodes in the tree. This is done exploring a generic node C_k. If this happens when node C_k is tagged "new", namely it has never been explored before, the second argument of the function $Duplicate$ is equal to Dup, i.e., all duplicate nodes should be considered. On the contrary, if C_k is not tagged "new", namely it has been already explored, $Duplicate$ is equal to $\mathcal{L}_k$, i.e., only the duplicate nodes that have been created after C_k has been examined the last time, should be considered.

The effect of Algorithm 2 is not only that of eventually creating new nodes and edges. It also updates: sets T_k and $\mathcal{L}_k$ associated with the current node C_k, sets $\mathcal{L}_j$ associated with the current deficient nodes $C_j \in Def$, sets Dup and Def.

Algorithm 2 uses the function $Paths(C_k, Duplicate)$ defined as follows:

- $Paths(C_k, Duplicate)$ is a function that returns all paths in the current MSCG obtained merging duplicate nodes in the set $Duplicate$, and terminating in node C_k. In particular, such paths are defined moving backward towards C_0, untill all variables Δ's defining the set of admissible firing delays of the transitions enabled at C_k have been encountered, taking into account isomorphism relationships, if any.

We point out that $Paths(C_k, Duplicate)$ always returns a finite number of paths. In fact, even if a cycle originates while merging duplicate nodes, when moving backward along the cycle, before the cycle is completed, the involved timers are reset and no trace of the upstream variables Δ's remains. Indeed, if by absurd, one timer is not reset, while moving backward, the number of variables Δ's associated with it would grow indefinitely. However, this is in contrast with the fact that a cycle is defined via a duplicate node and an isomorphism relationship associated with the input edge closing the cycle.

Let us now explain Algorithm 2 step by step.

It examines the set of transitions in T_k (Step 1). More precisely, given a transition $t_i \in T_k$, it checks if there exists at least one path in $Paths(C_k, Duplicate)$ that enables it (Step 3). If such is the case, t_i is removed from T_k (Step 4) and the marking M_q reached firing t_i at M_k is computed (Step 5). Lower and upper bounds of all transitions logically enabled at marking M_q are updated at Step 8 or 10.

At Step 11 a new node C_q labeled with marking M_q and set of inequalities Θ_q is added. Step 13 defines the label of the edge from C_k to C_q, to whom an additional entry may be eventually added in Step 16.

Step 16 clarifies how to handle the case in which C_q is a duplicate node. In particular, it defines the isomorphism relationship to be added as a fourth entry on the edge from C_k to C_q in accordance with the discussion in the previous section.

At Step 17 all sets $\mathcal{L}_j$ associated with the current set of deficient nodes (nodes in Def) are updated, including in all of them the new node C_q. Analogously, C_q is included in the set Dup at Step 19.

If the new node C_q is not duplicate, it is tagged "new" at Step 21.

Finally, if the set T_k is not empty after all transitions in it have been examined and C_k was not already in Def, then C_k is added to Def. Note that this could only occur when node C_k is examined for the first time. A node initially assigned to Def, may later be removed if its set T_k becomes empty (Step 24).

The algorithm terminates at Step 26 which imposes that set $\mathcal{L}_k$ is reset to the empty set if C_k is still a deficient node: all duplicate nodes computed up to now have been already considered when examining C_k.

Example *5:* Consider again the TPN system in Fig. 3. The main steps in the construction of the MSCT, built according to Revised Algorithm 1, and the resulting MSCG are reported in Fig. 4 as already mentioned in the previous examples.

• At Step 1 of Revised Algorithm 1, node C_0 is created and tagged "new": it contains the initial marking M_0 and the timing constraints of the transitions logically enabled at M_0 are those defined by function $Q(\cdot)$. Sets Def and Dup are initialized at the empty set (Steps 2 and 3).

• Node C_0 is examined since it is the only node of the tree and it is tagged "new" (Steps 4 and 5). Set T_0 is initialized at the set of transitions logically enabled at $p_1 + p_4$, namely, it is $T_0 = \{t_1, t_2, t_5\}$. Only transitions t_1 and t_2 satisfy the *if* condition at Step 3 of Algorithm 2. In particular, their firing leads to nodes C_1 and C_2, respectively, which are added to the tree and tagged "new" as shown in Fig. 4.a. Set T_0 becomes equal to $T_0 = \{t_5\}$, therefore at Step 22 of Algorithm 2 set Def is updated at $Def = \{C_0\}$. At Step 26 of the same algorithm a new set $\mathcal{L}_0$ is introduced and initialized at the empty set. Finally, node C_0 is untagged at Step 8 of Revised Algorithm 1.

• Assume that node C_1 is selected at Step 5 of Revised Algorithm 1. Set T_1 is initialized at $T_1 = \{t_4, t_5\}$. However, only t_4 satisfies the *if* condition in Step 3 of Algorithm 2. Therefore T_1 is updated at $T_1 = \{t_5\}$ and a new node, denoted C_3, is added to the tree and tagged "new" as shown in Fig. 4.b. At Step 22 of Algorithm 2, the set of deficient nodes Def is updated at $Def = \{C_0, C_1\}$. Furthermore, at Step 26 of the same algorithm, a new set $\mathcal{L}_1$ is introduced and initialized at the empty set.

• Assume that node C_2 is selected at Step 5 of Revised Algorithm 1. Set T_2 is initialized at $T_2 = \{t_3, t_5\}$. However, only t_3 satisfies the *if* condition in Step 3 of Algorithm 2. Therefore T_2 becomes equal to $\{t_5\}$ and a new node is added to the tree. As shown in Fig. 4.c such a node is denoted as C_1' to point out that it is equivalent to node C_1. Such a node is tagged "duplicate". Furthermore, in accordance with Step 16 of Algorithm 2, the isomorphism relationship $\Delta_0 := \Delta_2$ is added as the fourth enrty of the edge from C_2 to C_1'. A new set $\mathcal{L}_1'$ is introduced and initialized at the current set Dup, namely, it is $\mathcal{L}_1' = Dup = \emptyset$. At Step 17, sets $\mathcal{L}_0$ and $\mathcal{L}_1$ are updated, including the duplicate node C_1', namely it becomes $\mathcal{L}_0 = \mathcal{L}_1 = \{C_1'\}$. Set Dup is updated at Step 19. In particular, it is $Dup = \{C_1'\}$. At Step 22, set Def is updated at $Def = \{C_0, C_1, C_2\}$. A new set $\mathcal{L}_2$ is introduced and initialized at the empty set at Step 26. Finally, node C_2 is untagged at Step 8 of Revised Algorithm 1.

• At this point, there exists only one node tagged "new", namely C_3 therefore it is selected at Step 5 of Revised Algorithm 1. Set T_3 is initialized at $\{t_5\}$ since t_5 is the only transition enabled at p_4. Transition t_5

satisfies the *if* condition in Step 3 of Algorithm 2. Therefore T_3 becomes equal to the empty set. A new node is computed, denoted as C_5, which corresponds to the empty marking. Steps 22 to 26 produce no effect since none of the *if* condition is satisfied. Therefore sets Def and Dup remain unaltered and node C_3 is untagged.

• Again, there exists only one node tagged "new", namely C_5 therefore it is selected at Step 5 of Revised Algorithm 1. However, since it enables no transition ($T_5 = \emptyset$), the effect of Algorithm 2 is simply that of removing the tag "new" from it as shown in Fig. 4.d.

• Now, no node tagged "new" exist. However, $Def = \{C_0, C_1, C_2\}$ and $\mathcal{L}_0 = \mathcal{L}_1 = \{\mathcal{C}_1'\}$ (while $\mathcal{L}_2 = \emptyset$), therefore the condition on the *while* loop at Step 9 of Revised Algorithm 1 is satisfied. In more detail, nodes C_0 and C_1 also satisfy the condition on the *for* loop in Step 10.

Assume that node C_0 is considered first. Since function $Path(C_0, \mathcal{L}_0) = Path(C_0, \mathcal{C}_1')$ returns no path (duplicate node $\mathcal{C}_1'$ created no new path going from $\mathcal{C}_0$ to $\mathcal{C}_0$), no new node is added. The only effect of Algorithm 2 is that of updating $\mathcal{L}_0$ at the empty set.

• Let us now consider node C_1. Function $Path(C_1, \mathcal{L}_1) = Path(C_1, \mathcal{C}_1')$ returns the path $C_0C_2C_1$. Furthermore, transition $t_5 \in T_1$ satisfies the *if* condition of Step 3 of Algorithm 2. Therefore T_1 becomes equal to the empty set. A new node, denoted as C_4 in Fig. 4.e, is computed and tagged "new". Node C_1 is removed from set Def, which becomes equal to $Def = \{C_0, C_2\}$ at Step 24 of Algorithm 2.

• Now, there exists again a node tagged "new" (C_4) so Revised Algorithm 1 executes the first *while* loop. At node C_4 only one transition is enabled (t_4) and it leads to node C_5', which is a duplicate of C_5.

At this point no node tagged "new" exist. Furthermore, $Def = \{C_0, C_2\}$ but $\mathcal{L}_0$ and $\mathcal{L}_2$ are both equal to the empty set so the conditions to enter in both *while* loops are violated. As a result Revised Algorithm 1 stops and the tree is complete.

C. Finiteness of the MSCG built using Revised Algorithm 1

We conclude this section proving that Revised Algorithm 1 terminates in a finite number of steps, which in turns implies that the resulting MSCG is finite, provided that the following two assumptions hold: Assumption A1 in [1] (the TPN is bounded), plus an additional assumption that prevents situations like the one pointed out in Subsection III-A. In particular, if we denote as

$$T_0 = \{t_i \in T \ : \ l_i = 0\},$$

the additional assumption imposes that:

(B) no sequence $\sigma \in T_0^*$ is repetitive.

By Assumption B, for a given bounded net the maximum number of consecutive firings of transitions in T_0, denoted as r_0, is finite. Clearly, r_0 is a function of the initial marking.

In simple words, the MSCG is finite if there exist no repetitive sequence may fire in zero time. We notice that such an assumption does not provide a limitation in practical applications because this is just a necessary and sufficient condition to rule out the possibility that the model is zeno and consequently not suitable to describe a real physical system. Indeed, events represent operations on the system that require a non null time to be executed. As an example, in a manufacturing system, events may be the processing of a part by a machine, or the failure of a certain operation, or some activities performed by a human operator, and so on.

We now provide a lemma which is fundamental to demonstrate the finiteness of the MSCG. It claims that no transition may remain enabled, without firing, along a path of the MSCG of arbitrary lentgh.

Lemma 1: Consider a bounded TPN $\langle N_d, M_0 \rangle$ and its MSCG built according to Revised Algorithm 1. Under Assumption B, no transition $t \in T$ remains enabled without firing along a path of length greater than or equal to

$$\mu = (r_0|T \setminus T_0| + 1) \left\lceil \frac{u_{max}}{l_{min}} \right\rceil \tag{1}$$

where

$$l_{min} = \min_{t_i \in T \setminus T_0} l_i, \qquad u_{max} = \max_{t_i \in T \setminus T_0} u_i.$$

Proof: We prove this by contradiction. Assume that there exists a transition $\bar{t}$ which remains enabled without firing, along an arbitrarily long path in the MSCG. Let

$$C_{i_0} \xrightarrow{t_{j_0}, \mathcal{L}(t_{j_0}), \Delta_{i_0} \in [\ldots]} C_{i_1} \xrightarrow{t_{j_1}, \mathcal{L}(t_{j_1}), \Delta_{i_1} \in [\ldots]} C_{i_2} \rightarrow \ldots \tag{2}$$

be such a path, where the first node of the path, C_{i_0}, corresponds to the node where transition $\bar{t}$ is newly enabled. Since $\bar{t}$ remains enabled along the above path, a constraint on its firing delay $\bar{\theta}$ appears in all nodes of the path. In particular, if we denote by $\bar{l}$ and $\bar{u}$ the static bounds of $\bar{t}$, the constraint on $\bar{\theta}$ in the generic node C_r of the path, is equal to

$$\max\{0, \bar{l} - \sum_{q=0}^{r-1} \Delta_{i_q}\} \leq \bar{\theta} \leq \bar{u} - \sum_{q=0}^{r-1} \Delta_{i_q}. \tag{3}$$

Now, by Assumption B, the number of consecutive firings of transitions with a null firing delay in the path at hand, is bounded and can be easily computed. Indeed, the longest sequence of transitions that fire consecutively in zero time and may appear in the path, have the following form:

$$\sigma_{0,1} t_{\varrho_1} \sigma_{0,2} t_{\varrho_2} \ldots t_{\varrho_k} \sigma_{0,k} \tag{4}$$

where $\sigma_{0,i} \in T_0^*$, $t_{\varrho_i} \in T \setminus T_0 \setminus \{\bar{t}\}$, for $i = 1, \ldots, k$, and $t_{\varrho_i} \neq t_{\varrho_j}$ for all $i \neq j$. The last condition follows from the fact that, when a transition fires, its timer is reset and, in the case of a transition in $T \setminus T_0$, the next firing may only occur after a time interval which is greater than zero by definition. As a result, the integer k in (4) satisfies

$$k = |T \setminus T_0| - 1$$

where -1 originates from the fact that $\bar{t}$ is not firing by assumption. Therefore, the max length of a sequence defined as in Eq. (4), is equal to $r_0(|T \setminus T_0| - 1) + r_0 = r_0|T \setminus T_0|$.

After the firing of a sequence defined as in Eq. (4), a transition in $T \setminus T_0 \setminus \{\bar{t}\}$ fires after a delay greater than or equal to l_{min}. This enables us to conclude that, after a number of firings equal to $r_0|T \setminus T_0| + 1$, all admissible values for the timer of $\bar{t}$ are certainly smaller than $\bar{u} - l_{min}$. Repeating this reasoning, after at most $\bar{\mu}$ firings, where $\bar{\mu}$ is equal to

$$\bar{\mu} = (r_0|T \setminus T_0| + 1) \left\lceil \frac{\bar{u}}{l_{min}} \right\rceil,$$

all admissible values of the timer of $\bar{t}$ is certainly smaller than l_{min}, thus the transition must fire after at most $r_0|T \setminus T_0| + 1$ additional firings. The statement follows from the fact that $\forall \bar{t} \in T \setminus T_0$, it is

$$\bar{\mu} \leq \mu = (r_0|T \setminus T_0| + 1) \left\lceil \frac{u_{max}}{l_{min}} \right\rceil.$$

■

Proposition 1: Let $\langle N_d, M_0 \rangle$ be a *bounded* TPN. Under Assumption B, the MSCT as defined by Revised Algorithm 1 is finite.

Proof: Each node C_e of the graph is labeled by pair (M_e, Θ_e) where M_e is a reachable marking and Θ_e is a constraint set. The result follows from two facts. First, being the TPN bounded, the number of reachable markings is finite. Secondly, by Lemma 1 in each inequality in Θ_e the number of Δ variables is at most equal to μ, hence, the number of possible constraint sets is finite. ■

Clearly, if the MSCT is finite, the MSCG is finite as well, since it is obtained simply merging the duplicate nodes.

We conclude this section mentioning that in [1] the static bounds of all transitions were assumed to be rational numbers. As shown in the above proposition, such a condition is not necessary to guarantee the finiteness of the MSCG.

VI. Conclusions

In this Technical Note we made some remarks on the paper by Basile *et al.* [1]. In particular, the concerns are related to a graph, called *Modified State Class Graph*, which should, under certain assumptions, summarize all possible evolutions of a labeled Time Petri net system. Three are the main criticalities in the algorithm in [1]: the resulting MSCG may be infinite, there is a problem in the notation used in [1], some possible evolutions could be missing. A revised version of the algorithm is proposed in this paper and it is proved to be finite under an additional assumption, not mentioned in [1].

We finally observed that the procedures for state estimation and fault diagnosis proposed in [1], as well as the diagnosability analysis approach in [2], which use the Modified State Class Graph as the basis tool, are not affected by the proposed modifications in its definition and in its rules of construction.

Acknowledgement

This work was supported by the National Natural Science Foundation of China under Grant Nos. 61803246, 61472295, and 61703321, the Science and Technology Development Fund, MSAR, under Grant No. 078/2015/A3, and the Natural Science Foundation of Shaanxi University of Science and Technology under Grant Nos. 2016BJ-15 and 2017BJ-39.

References

[1] F. Basile, M.P. Cabasino, and C. Seatzu. “State estimation and fault diagnosis of time labeled Petri net systems with unobservable transitions,” *IEEE Transactions on Automatic Control*, vol. 60, no. 4, pp. 997–1009, 2015.

[2] F. Basile, M.P. Cabasino, and C. Seatzu. “Diagnosability analysis of labeled time Petri net systems,” *IEEE Transactions on Automatic Control*, vol. 62, no. 3, pp. 1384–1396, 2017.

[3] T. Murata. “Petri nets: Properties, analysis and applications,” *in Proc. IEEE*, vol. 77, no. 4, pp. 541–580, 1989.