Towards the Exhaustive Verification of Real-Time Aspects in Controller Implementation

Carlo A. Furia[*], Marco Mazzucchelli[†], Paola Spoletini[‡] and Mara Tanelli[§]

January 29, 2008

Abstract

In industrial applications, the number of final products endowed with real-time automatic control systems that manage critical situations as far as human safety is concerned has dramatically increased. Thus, it is of growing importance that the control system design flow encompasses also its translation into software code and its embedding into a hardware and software network. In this paper, a tool-supported approach to the formal analysis of real-time aspects in controller implementation is proposed. The analysis can ensure that some desired properties of the control loop are preserved in its implementation on a distributed architecture. Moreover, the information extracted automatically from the model can also be used to approach straightforwardly some *design* problems, such as the hardware sizing in the final implementation.

[*]Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy; `furia@elet.polimi.it`

[†]Dipartimento di Matematica, Università di Pisa, Italy; `mazzucchelli@mail.dm.unipi.it`

[‡]Università dell'Insubria, Como, Italy; `paola.spoletini@uninsubria.it`

[§]Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy; `tanelli@elet.polimi.it`

Contents

1 Introduction and Motivation

In industrial applications, the number of final products endowed with real-time automatic control systems has dramatically increased in the last few years. Most of such systems, moreover, are responsible of managing critical situations as far as human safety is concerned, see *e.g.*, the aeronautical and the automotive industries. Thus, besides guaranteeing model-based structural stability properties of the closed-loop system proved and proving to meet the desired performance specifications by means of Control Theory methodologies, it is of growing importance to complete the control system design flow taking care also of formally tackling its translation into software code and its embedding in a larger hardware and software network. Doing this becomes harder when the embedded system behavior needs to be analyzed considering both functional and timing properties, as it is the case for real-time controllers. As such, it seems particularly useful and promising to try and bridge the gap between system-theoretic and information-theoretic skills, devising new tools which allow to carry out the whole design flow and provide new information to optimize the final system both in the controller design phase and in its code and hardware realization. Recently, in the control community, some preliminary considerations about the possible role for formal methods to tackle control software dependability properties have started to appear, see *e.g.*, [14], [12]. In [14], the authors discuss how probabilistic model checking can be used to analyze dependability properties of controller-based systems, concentrating on its use for analyzing fault detection schemes. [12], instead, offers a wide discussion on how formal methods in general may help to develop dependable manufacturing control systems. Note that, in these works, the focus in mainly on PLC program verification, rather than on safety-critical real time applications. Another work in this direction is [16], where algebraic equivalence is used to verify the correct operation of control systems. Here, real-time systems are modeled using Timed Transition Model, and the final outcome is a technique to formally verify the behavioral correctness of a control system.

In the formal methods community, a vast gamut of techniques, methods, and tools has been studied by computer scientists for over two decades [5]. In particular, since the early 1990's, several techniques have been developed to analyze *real-time* systems with respect to both behavioral and timing properties [8]. Once a formal model of the system—including exact timing information—has been built, it can be analyzed with various degrees of exhaustiveness, ranging from simple partial simulation up to exhaustive automated verification. In particular, model-checking techniques for real-time systems [9, 30, 15] have become a very popular solution to the system verification problem. In the model-checking framework, a system model—consisting of some form of timed automaton—is checked against some required properties—formalized through metric temporal logic formulas. A great strength of the model-checking approach is that it is a fully automated and exhaustive technique: once the system has been built and its requirements have been formalized, a software tool decides whether the requirements hold for the system. However, besides the well-known intrinsic scalability problems of such techniques, full automation must always be heavily traded-off with expressiveness. Consequently, the temporal logic language in which the desired properties can be formalized, and fed to a model-checker, is usually of rather limited expressive power. In particular, it is impossible to formalize complex properties that have to be analyzed in the design flow of embedded systems towards their implementation. In the same vein, but more generally, formal techniques are usually mostly focused on *a posteriori* analysis, and provide instead little or no support to *design* [18, 10, 13]. There has been some recent work addressing this problem, see *e.g.*, [20], for instance by devising suitable mathematical models to embedded software design, which allow to incorporate constraints and specifications at a high level of abstraction and to formally derive semantics-preserving software implementations.

In this paper, we propose a tool-supported approach to the formal analysis of real-time aspects in controller implementation. Our methodology builds upon an automated algorithmic enumeration technique of system behavior to pursue a guided examination of such real-time aspects. The analysis can ensure that some desired properties of the control loop are preserved in their implementation on a distributed architecture. Moreover, the information extracted automatically from the model can also be used to approach straightforwardly some *design* problems, such as the hardware sizing in the final implementation.

Our technique uses timed Petri nets as abstract system modeling paradigm. Petri nets are formal real-time models which are especially suitable to capture rather intuitively aspects such as asynchronous data transmission, which are fundamental in physical implementation architectures. For illustration purposes, however, we will present the aforementioned algorithms for behavior enumeration using the more abstract, and essential, model of state/transition systems (STSs). Roughly, a STS behavior is simply an alternation between residences in states, when time elapses, and instantaneous occurrences of transitions, which trigger the system to a new state. We introduce an analysis technique to enumerate all possible timed behaviors of some considered STS, by presenting them succinctly grouped into equivalent classes, which share the same behavior. Afterward, we will see how to extend the analysis technique for STS to timed Petri nets, and then how the results of such analyses can be used proficiently in the iterative development of controller implementations.

The structure of the paper is as follows. Section 2 introduces the notion of state/transition system (STS) and describes a technique to represent and analyze the behavior of a given STS. Section 3 demonstrates the usefulness of such formalism to analyze real-time control systems, considering—as case study—an active braking controller. In particular, after a brief description of the control system, we explicitly consider the final hardware lay-out—*i.e.*, sensors, actuators and data transmission—and we apply the proposed formal analysis technique the given model. The paper ends with a discussion of the most important "lessons" learned in developing the case study (Section 4) and with concluding remarks and an outlook to future work (Section 5).

2 State/Transition Systems

This section introduces the notion of state/transition system (STS), and describes a technique to represent and analyze the behavior of a given STS. More precisely, the following Subsection 2.1 provides an informal overview to the whole section. Then, Subsection 2.2 defines formally transition systems (TSs), a simpler model which is the basis for full STSs, and Subsection 2.3 describes an analysis technique for such systems. Finally, Subsection 2.5 introduces full STSs and timed Petri nets as an extension of TSs, and it shows how the technique of the previous sections can be adapted to these formalisms. Some of these results have been presented in preliminary form in [21] and in [4] with focus on the SDL formalism.

2.1 Informal Overview

STSs are an abstract modeling paradigm suitable for a large variety of timed systems. As it is apparent from their name, they are based on the notions of *state* and *transition*. Informally, the evolution of a STS is characterized by an alternation between residences in states, where time elapses, and instantaneous occurrences of transitions, which trigger the system to a new state. At any instant of time, the current state of the system determines which transitions are *enabled*, *i.e.*, can *fire*; conversely, any fired transition triggers a change of state in the system.

As a simple example, consider a producer and a consumer which communicate through a (bounded) buffer, where the producer puts new items and the consumer takes them away. A STS modeling such a system would introduce a transition c which occurs whenever the consumer takes an item from the buffer. Note that c would be enabled only when the system is in a state where at least one item is in the buffer. Also, when c is taken and the buffer has some $n > 0$ items in it, the state of the buffer changes to represent the fact that $n - 1$ items are now stored in the buffer. This example sketch will be developed in Section 2.5, by means of a Petri net model (Figure 3).

When modeling *real-time* systems as STSs, the notion of state must include a *timing* information about when the current state has been entered. Consequently, the triggering of transitions depends also on the time elapsed in the current state. Therefore, transitions are decorated with a lower bound and an upper bound denoting, respectively, the minimum and maximum time that should be consumed in a state enabling the transition before the transition is taken.

STSs are a useful abstraction of timed systems, in that they have a sufficient expressive power to model the essential features of the real-time behavior of such systems. All the more, such essential features can be often further abstracted by representing explicitly only transitions, and by leaving the notion of state only implicit: such models are called transition systems (TSs). Nonetheless, both TSs and STSs are usually considered a bit too abstract for practical modeling purposes, because they represent only indirectly the features of the actual systems under modeling. Hence, more standardized and user-friendly notations—such as (timed) Petri nets [7, 3] or SDL [27, 6]—-are usually preferred to model real-time systems. Let us remark, however, that it is straightforward to represent a Petri net, or an SDL diagram, as a STS, and to extend to a STS the same techniques used for TSs. Thus, it is advisable to present our results and techniques on the neater and more abstract model of TSs first. Afterward, we will show the minimal adaptations required to apply the method first on full STSs, and then directly on our formalism of choice (*i.e.*, timed Petri nets).

Analyzing the behavior of STSs. A fully formalized STS can be seen as an implicit, complete description of a set of timed behaviors, namely all those which are compatible with all the constraints included in the STS. For several analysis purposes, it would be extremely useful to have a characterization of the same behaviors which is explicit, rather than implicit. In other words, we would like to possess a representation which allows answering rather directly questions about the overall behavior of the system.

For instance, going back to the example of the producer and consumer, one may consider questions such as:

- given certain producing and consuming rates, is it true that the buffer never overflows?
- is it true that every item stays in the buffer for at least 10 time units before it is consumed?
- if we double the consuming rate, can we halve the buffer size without risking overflow?

Simulation techniques are a way to answer such questions. However, simulation techniques give necessarily incomplete answers, because they consider only a subset of all possible behaviors of the system. In this paper, on the contrary, we focus on exhaustive techniques which guarantee that no behavior, however unlikely, is neglected. So, we develop like a technique to extract and represent all possible behaviors of a given STS.

More precisely, we describe an algorithm which, given a STS, extracts all timed behaviors the system can exhibit. Such behaviors can then be analyzed directly to answer some of the questions about the system. Notice that, if time is modeled with a dense domain (such as the real numbers), we have an infinite number of different behaviors even if we limit the total number of transitions that the system can take. Consequently, an enumeration of such behaviors would be extremely impractical and possibly totally useless. Still, an explicit enumeration is still feasible if we group behaviors into equivalence classes and represent compactly every class. We consider two behaviors equivalent when they comprise the same sequence of transitions, in the same order; in other words, equivalent behaviors differ only by exact timing information, not by the ordering of transitions. Each class is represented as a sequence of taken transitions, together with the intervals of time in which each of them can occur. As we will demonstrate, such a representation is finite, given an upper bound on the total number of taken transitions, and practically useful.

2.2 Definition of Transition Systems

As the name suggests, transition systems are based on the notion of transition. Transitions are defined as follows.

Definition 1. A *transition* τ is a triple $\langle [\alpha, \beta], E, D \rangle$ where:

- $[\alpha, \beta]$ is an interval of time called *static firing interval*, where α and β are named respectively (static) earliest firing time and (static) latest firing time. They denote the lower and upper bound on the transition firing time, after it has become enabled.
- E (resp. D) is the set of transitions which become enabled (resp. disabled) when τ fires.

Note that, at a given time, more than one occurrence of the same transition can be enabled, each with its own enabling time. To represent this fact we introduce the following.

Definition 2. An *occurrence* or *instance* of a transition τ is a triple $\langle \mathrm{id}, x, [\hat{\alpha}, \hat{\beta}] \rangle$ where:

- id is a unique identifier, to distinguish different instances of the same transition τ;[1]
- the (absolute) enabling time x;
- the dynamic firing interval $[\hat{\alpha}, \hat{\beta}]$, with $\hat{\alpha} = x + \alpha$ and $\hat{\beta} = x + \beta$, within which the transition *must* fire if it is not disabled.

In the following, we will freely use the term "transition" to denote indifferently a transition or an instance thereof, whenever the actual meaning will be clear from the context. Also, we will distinguish relative firing times from absolute ones by putting a hat over the latter.

Finally, we introduce the notion of TS.

Definition 3. A *transition system* S is a pair $\langle T, T_0 \rangle$ where:

- T is a set of transitions;
- $T_0 \subseteq T$ is the set of transitions which are enabled *initially*.

Any TS is an implicit representation of a set of behaviors, namely all those which respect the enabling/disabling relations and the firing time intervals. More precisely, we introduce the following.

Definition 4. An *evolution* e (or *trace*) of a TS $S = \langle T, T_0 \rangle$ is given by two finite sequences:

- the sequence $\langle \tau_1, \tau_2, \ldots, \tau_k \rangle$ of instances of fired transitions; and
- the sequence $\langle \hat{\tau}_1, \hat{\tau}_2, \ldots, \hat{\tau}_k \rangle$ of firing times of the transitions.

The two sequence must respect the following constraints: for all $i = 1, \ldots, k$, there exists a $0 \leq j < i$ such that:

- *(enabling/disabling)* either $j > 0$, $\tau_i \in E_{\tau_j}$, and for all $j < k < i$ it is $\tau_i \notin D_{\tau_k}$; or $j = 0$, $\tau_i \in T_0$, and for all $1 \leq k < i$ it is $\tau_i \notin D_{\tau_k}$;
- *(enabling times)* either $j > 0$ and $x_{\tau_i} = \hat{\tau}_j$; or $j = 0$ and $x_{\tau_i} = 0$;
- *(firing interval)* $\hat{\tau}_i \in [\alpha_{\tau_i} + x_{\tau_i}, \beta_{\tau_i} + x_{\tau_i}]$.

If we want to stress the fact that the number of taken transitions in e is bounded by some natural $k > 0$, we call e an *NT-evolution* (*i.e.*, "Number of Transitions"). Conversely, if we want to stress the fact that the time at which any transition in e is taken is bounded by some positive time $\hat{\tau}_k$, we call e a *MET-evolution* (*i.e.*, "Maximum Execution Time").

In the following we will focus solely on NT-evolutions, although extending the results to MET-evolutions is routine.

[1]For simplicity, in this paper we distinguish among different instances of the same transition by priming the transition name.

Equivalence classes of evolutions. From the definitions above, it should be clear that the number of NT-evolutions of a TS are, in general, infinite, when the time domain is a dense set. In practice, this is the case whenever there is at least one enabled transition whose firing time is non-deterministic, that is whose firing interval is not a singleton.

In order to overcome this unpleasantness, we introduce a notion of equivalence between evolutions: two evolutions are *equivalent* if they have the same sequence of fired transitions, in the same order. Consequently, we introduce a notion of equivalence class as follows.

Definition 5. For a natural $k > 0$, a *k-NT-class* is an equivalence class over the set of all NT-evolutions of length k.

A k-NT-class can be represented as a tuple $\langle \tau_1, \ldots, \tau_k \rangle$ of the ordered sequence of transitions.

It is not difficult to show that, regardless of the nature of the time domain, and in particular even if it is dense, the quotient set of the NT-evolutions of a TS by the equivalence relation is finite.

Theorem 6. *Given a transition system S, for each integer $k > 0$, the number of k-NT-classes is finite.*

Proof. Let $S = \langle T, T_0 \rangle$, with $|T| = n$, and let $\mathcal{D}$ be the set of all k-permutations with repetitions of the set T:

$$\mathcal{D} = \{\langle \tau_1, \ldots, \tau_k \rangle | \ \tau_1, \ldots, \tau_k \in T\}$$

It is immediate that $|\mathcal{D}| = n^k$ is finite. The set $\mathcal{E}$ of all k-NT-classes is a subset of $\mathcal{D}$, so also of finite size. $\square$

In the rest of the paper, we will assume the time domain to be the nonnegative rational numbers $\mathbb{Q}_{\geq 0}$. Intuitively, this restriction is necessary to ensure that all different time bounds have a common multiple, or, in other words, that a notion of "minimum time granularity" is definable. This would not be possible over the whole real domain, where rational and irrational numbers are incommensurate. For all practical purposes, however, it is clear that the restriction to rational numbers is irrelevant, as any irrational number can be approximated by rational numbers with arbitrary precision.

2.3 Exhaustive Enumeration of TSs Behavior

Theorem 6 suggests that an exhaustive enumeration of NT-classes may be feasible. This subsection presents an algorithm to build such an enumeration. The algorithm is based on similar techniques introduced by Berthomieu, Diaz, and Menasce for timed Petri nets [2, 1]. It has been implemented in a very prototypal tool which has been used in analyzing the case study of Section 3.

Let us consider a TS $S = \langle T, T_0 \rangle$. It is convenient to represent the NT-classes of S as a tree, named *relative tree*. Roughly, nodes represent states of the system evolutions, and every node is connected to all its children, which correspond to states that are reachable from the parent state by firing an enabled transition.

More precisely, every node in the tree hosts a triple of the form:

$$\langle ET_i^j, D_i^j, mM_i^j \rangle$$

where:

- the subscript i denotes the depth of the node within the tree;
- the superscript j denotes that the node has been reached by firing transition τ_j;[2]
- ET_i^j is the set of the transitions that are currently enabled;
- D_i^j is the set of firing constraints of transitions in ET_i^j, described as a set of inequalities in these forms:

$$\begin{cases} \alpha_x \leq \tau_x \leq \beta_x & \text{with } \tau_x \in ET_i^j,\ \alpha_x \in \mathbb{Q}_{\geq 0},\ \beta_x \in \mathbb{Q}_{\geq 0}, \alpha_x \leq \beta_x \\ \tau_y - \tau_z \leq \gamma_{yz} & \text{with } \tau_y, \tau_z \in ET_i^j,\ \gamma_{yz} \in \mathbb{Q},\ \tau_y \neq \tau_z \end{cases}.$$

  Note that the generic symbol τ_w also denotes the generic instant at which transition τ_w fires, relative to the firing of the previous transition τ_j; *i.e.*, $\hat{\tau_w} = \hat{\tau_j} + \tau_w$;
- mM_i^j is the least upper bounds among the transition times of transitions enabled in the current node, *i.e.*, $mM_i^j = \min\left\{\beta_x | (\alpha_x \leq \tau_x \leq \beta_x) \in D_i^j\right\}$.

Let us now show how to build the relative tree, starting from the root node.

2.3.1 The Root Node

The root of the relative tree hosts the triple $\langle T_0, D_0, mM_0 \rangle$, where:

- $\{\tau_1, \ldots, \tau_m\} = T_0 \subseteq T$ is the set of the transitions that are enabled initially (*i.e.*, at absolute time 0, when the system starts)
- D_0 is the firing range of the transitions in T_0, described by the following set of inequalities:

$$D_0 = \begin{cases} \alpha_1 \leq \tau_1 \leq \beta_1 \\ \quad \vdots \\ \alpha_m \leq \tau_m \leq \beta_m \end{cases}$$

  where $[\alpha_i, \beta_i]$ is the static firing interval of transition τ_i, for all $i = 1, \ldots, m$.

  Note that, in the root node, inequalities in the other form $\tau_i - \tau_j \leq \gamma$ are redundant.
- $mM_0 = \min\{\beta_i | (\alpha_i \leq \tau_i \leq \beta_i) \in D_0\} = \min\{\beta_i | \tau_i \in T_0\}$.

2.3.2 Generic Nodes

Let us consider a generic node $\langle ET_w, D_w, mM_w \rangle$ at depth w. A transition $\tau_f \in ET_w$ can fire if and only if the following set of inequalities admits a solution:

$$\begin{cases} \alpha_x \leq \tau_x \leq \beta_x & \text{for all } (\alpha_x \leq \tau_x \leq \beta_x) \in D_w \\ \tau_y - \tau_z \leq \gamma_{yz} & \text{for all } (\tau_y - \tau_z \leq \gamma_{yz}) \in D_w \\ \tau \leq \tau_e & \text{for all } \tau_e \in ET_w \setminus \{\tau_f\} \end{cases}$$

Such a solution exists if and only if:

[2]This information will be omitted whenever unnecessary.

1. $(\alpha_f \leq \tau_f \leq \beta_f) \in ET_w$; and
2. $\alpha_f \leq mM_w$; and
3. there is no transition $\tau_e \in ET_w$ such that $\tau_e - \tau_f \leq \gamma_{ef}$ for some $\gamma_{ef} < 0$.

In particular, 3 is necessary as if $\tau_e - \tau_f \leq \gamma_{ef} < 0$ for some τ_e, then $\tau_e < \tau_f$, that is τ_e must fire before τ_f. Hence, the latter cannot fire in the current node.

Assume that transition τ_f can indeed fire from the current node at depth w. Then, we build a child node $\langle ET^f_{w+1}, D^f_{w+1}, mM^f_{w+1}\rangle$ at depth $w+1$, as follows.[3]

- $ET^f_{w+1} = (ET_w \setminus D_{\tau_f}) \cup E_{\tau_f}$, that is we update ET_w by deleting all occurrences of disabled transitions and by adding all enabled transitions[4]
- D^f_{w+1} is obtained by updating D_w according to the following procedure:
  1. Inequalities corresponding to transitions in $(ET_w \setminus \{\tau_f\})$ are all temporarily added to D^f_{w+1} with the following modifications:
  $$\begin{array}{rcl} \tilde{\alpha}_i & = & \max\{0, \alpha_i - mM_w, -\gamma_{fi}\} \\ \tilde{\beta}_i & = & \min\{\beta_i - \alpha_f, \gamma_{if}\} \\ \tilde{\gamma}_{jk} & = & \min\{\beta_j - \alpha_k, \gamma_{jk}\}. \end{array}$$
  2. Inequalities corresponding to transitions disabled by the firing of τ_f, *i.e.*, transitions in $(ET_w \setminus ET^f_{w+1} - \{\tau_f\})$, are removed in D^f_{w+1}. For each disabling of a transition τ_e the remaining inequalities in D^f_{w+1} are changed as follows:
  $$\begin{array}{rcl} \tilde{\alpha}_i & = & \max\{\tilde{\alpha}_i, \tilde{\alpha}_e - \tilde{\gamma}_{ei}\} \\ \tilde{\beta}_i & = & \min\left\{\tilde{\beta}_i, \tilde{\beta}_e + \tilde{\gamma}_{ie}\right\} \\ \tilde{\gamma}_{jk} & = & \min\{\tilde{\gamma}_{jk}, \tilde{\gamma}_{je} + \tilde{\gamma}_{ek}\}. \end{array}$$
  3. Inequalities corresponding to transitions enabled by the firing of τ_f, *i.e.*, transitions in $(ET^f_{w+1} \setminus ET_w)$, are added to D^f_{w+1}. For each enabling of a transition τ_a, the following inequalities are added:
  $$\left\{\begin{array}{lcll} \alpha_a \leq \tau_a & \leq & \beta_a & \\ \tau_a - \tau_k & \leq & \gamma_{ak} = \beta_a - \tilde{\alpha}_k & \text{for all } \tau_k \in (ET_w \cap ET^f_{w+1}) \\ \tau_k - \tau_a & \leq & \gamma_{ka} = \tilde{\beta}_k - \alpha_a & \text{for all } \tau_k \in (ET_w \cap ET^f_{w+1}). \end{array}\right.$$
  The following inequalities are also added to D^f_{w+1}, for all pair of transitions $\tau_{a_1}, \tau_{a_2} \in E_{\tau_f}$ with $\tau_{a_1} \neq \tau_{a_2}$:
  $$\left\{\begin{array}{lcl} \tau_{a_1} - \tau_{a_2} & \leq & \gamma_{a_1a_2} = \beta_{a_1} - \alpha_{a_2} \\ \tau_{a_2} - \tau_{a_1} & \leq & \gamma_{a_2a_1} = \beta_{a_2} - \alpha_{a_1}. \end{array}\right.$$
- $mM_{w+1} = \min\left\{\tilde{\beta}_i | (\tilde{\alpha}_i \leq \tau_i \leq \tilde{\beta}_i) \in D^f_{w+1}\right\}$.

[3]For clarity, we decorate interval bounds α, β in the child node with a tilde.

[4]Note that a transition which is both in D_{τ_f} and in E_{τ_f} and is enabled in the parent node becomes disabled, and a new instance of the same transition is enabled in the child node.

2.3.3 Properties of the Relative Tree

The following theorems, whose proof can be obtained similarly as in [1], establish the fundamental properties of the relative tree construction.

Theorem 7. *Let $\mathcal{T}$ be the relative tree of a TS S; all paths of length k from the root of $\mathcal{T}$ represent all and only k-NT-classes of S.*

Theorem 8. *Let $\mathcal{T}$ be the relative tree of a TS S, and let $\langle \tau_1, \ldots, \tau_i, \ldots, \tau_n, \ldots, \tau_k \rangle$ be the path in $\mathcal{T}$ corresponding to some k-NT-class of S, for some $i < n \leq k$, such that $\tau_i \in ET_{i-1}$ and $\tau_n \in ET_i$. Then, every evolution in the same k-NT-class respects the constraint:*

$$\hat{\tau}_i + \alpha_n \leq \hat{\tau}_n \leq \hat{\tau}_i + \beta_n.$$

*In addition, if $n = i + 1$ (*i.e.*, τ_n fires immediately after τ_i), every evolution in the same k-NT-class respects the stronger constraint:*

$$\hat{\tau}_i + \alpha_n \leq \hat{\tau}_n \leq \hat{\tau}_i + mM_i.$$

Theorem 9. *Let $\mathcal{T}$ be the relative tree of a TS S, and let $\langle \tau_1, \ldots, \tau_{k-1}, \tau_k \rangle$ be the path in $\mathcal{T}$ corresponding to some k-NT-class of S, for some $k > 1$, such that $\tau_{k-1} \in ET_{k-2}$ and $\tau_k \in ET_{k-1}$. Then, for all (real) times:*

$$t \in [\hat{\tau}_{k-1} + \alpha_k, \hat{\tau}_{k-1} + mM_{k-1}]$$

there exists some NT-evolution in the chosen k-NT-class where τ_k fires at t.

2.3.4 Complexity of the Algorithm

Let us briefly evaluate the time complexity of the algorithm for the construction of the relative tree. The relevant section is the one in 3, where an inequality $\tau_x - \tau_y \leq \gamma_{xy}$ is added for every ordered pair $\langle \tau_x, \tau_y \rangle$ of enabled transitions; there are $O(n^2)$ such ordered pairs, if n is the number of enabled transitions. Then, the worst case occurs when in every node all enabled transitions can fire. Let $\overline{n} = \max\{|ET_i^j|\}$ be the maximum number of enabled transitions in a node of the tree; then the total number of nodes in a tree of height k[5] is bounded by: $\sum_{i=0,\ldots,k-1} \overline{n}^i = (\overline{n}^k - 1)/(\overline{n} - 1) = O(\overline{n}^{k-1})$. Overall, building the relative tree has complexity $O(\overline{n}^2 \overline{n}^{k-1}) = O(\overline{n}^{k+1})$, exponential in k.

2.4 Real-Time Profiling of TS Behavior

In the previous subsection we introduced an algorithm which builds all k-NT-classes of a given TS. In this subsection we build upon that algorithm to design a technique to create the *real-time profile* of any NT-class. In other words, we show how to construct an interval—of absolute time—for every transition in an NT-class, which describes the minimum and maximum absolute firing times that every evolution in the chosen NT-class may have.

Once the relative tree has been built, real-time profiling is a two-phase process. First, the *absolute tree* for the TS is built; then the actual profiling is performed for a chosen NT-class of the TS. These two phases are described in the rest of this subsection.

[5] As it is customary, the root node has height 1.

2.4.1 Absolute Tree

For any TS S, the absolute tree is a tree isomorphic to the relative tree, but where every node hosts information about the absolute—rather than relative—firing times of transitions. In order to avoid ambiguities between the relative and absolute trees, and in accordance with the notation introduced in Subsection 2.2, we are going to decorate every symbol appearing in the absolute tree with a hat sign.

As in the relative tree, every node in the absolute tree hosts a triple $\langle \hat{ET}_i^j, \hat{D}_i^j, \hat{mM}_i^j \rangle$, where:

- $\hat{ET}_i^j$ is the same as ET_i^j in the relative tree;
- $\hat{D}_i^j$ is a set of absolute time constraints for the transitions in $\hat{ET}_i^j$, in the form:
$$\hat{\alpha}_x \leq \hat{\tau}_x \leq \hat{\beta}_x \quad \text{with } \hat{\tau}_x \in \hat{ET}_i^j$$
. Note that inequalities in the other form $\tau_j - \tau_k \leq \gamma_{jk}$ are not included in the absolute tree;
- $\hat{mM}_i^j = \min \left\{ \hat{\beta}_k | (\hat{\alpha}_i \leq \hat{\tau}_i \leq \hat{\beta}_i) \in \hat{D}_i^j \right\}$.

Similarly as for the relative tree, the absolute tree is built starting from the root node, by adding subsequent nodes recursively. Every edge connecting a parent node at depth i to one of its children is conventionally labeled with the inequality corresponding to the fired transition $\hat{\tau}_f \in \hat{ET}_i^j$, *e.g.*, $\hat{\alpha}_f \leq \hat{\tau}_f \leq \hat{mM}_i^j$.

Root node. In the root node, $\hat{D}_0$ is the same as D_0 in the relative tree, because the base time is 0 and thus relative and absolute times coincide at the beginning.

Generic nodes. Let us consider a generic node $\langle \hat{ET}_w, \hat{D}_w, \hat{mM}_w \rangle$ at depth w. Let $\tau_f \in \hat{ET}_w$ be a transition which can fire. Then, we build a child node $\langle \hat{ET}_{w+1}^f, \hat{D}_{w+1}^f, \hat{mM}_{w+1}^f \rangle$ at depth $w+1$, as follows.

- $\hat{ET}_{w+1}^f$ and $\hat{mM}_{w+1}^f$ are the same as ET_{w+1}^f and mM_{w+1}^f in the relative tree;
- $\hat{D}_{w+1}^f$ contains the following inequalities
$$\begin{cases} \hat{\hat{\alpha}}_a \leq \hat{\tau}_a \leq \hat{\hat{\beta}}_a & \text{with } \hat{\hat{\alpha}}_a = \alpha_a + \hat{\alpha}_f, \ \hat{\hat{\beta}}_a = \beta_a + \hat{mM}_w \\ & \text{for all } \tau_a \in (ET_{w+1}^f \setminus ET_w) \\ \hat{\hat{\alpha}}_i \leq \hat{\tau}_i \leq \hat{\hat{\beta}}_i & \text{with } \hat{\hat{\alpha}}_i = \max\{\hat{\alpha}_i, \hat{\alpha}_f\}, \ \hat{\hat{\beta}}_i = \hat{\beta}_i \\ & \text{for all } \tau_i \in (ET_w \cap ET_{w+1}^f). \end{cases}$$

2.4.2 Properties of the Absolute Tree

The following theorems establish the fundamental properties of the absolute tree construction.

Theorem 10. *Let $\langle \tau_1, \ldots, \tau_k \rangle$ be any k-NT-class of some TS S, whose absolute tree $\mathcal{T}$ hosts the inequality $\hat{\alpha} \leq \hat{\tau}_k \leq \hat{\beta}$ for τ_k in $\hat{D}_{k-1}$. Then, for all (real) times t, there exists some NT-evolution in the chosen NT-class where τ_k fires at t if and only if:*

$$t \in \left[\hat{\alpha}_f, m\hat{M}_{k-1}\right].$$

Proof. The proof goes by induction on the length k of the chosen NT-class.

Base case. Let $k = 1$; then the NT-class is just $\langle \tau \rangle$, and $\hat{D}_0$ contains the inequality $\hat{\alpha} \leq \hat{\tau} \leq \hat{\beta}$ with $\hat{\alpha} \leq m\hat{M}_0$. Any transition $\tau_x \in \hat{ET}_0$ such that $\hat{\alpha}_x \leq \hat{\tau}_x \leq \hat{\beta}_x = m\hat{M}_0$ is in $\hat{D}_0$ must fire within $m\hat{M}_0$. The same holds for any transition that fires before τ_x. Hence, τ can fire anywhere in the interval $[\hat{\alpha}_f, m\hat{M}_0]$, which proves the base case.

Inductive step. Assume that the theorem holds for all NT-classes of length n. Let us consider a generic NT-class $\langle \tau_1, \ldots, \tau_{\text{last}}, \tau_{n+1} \rangle$ of length $n + 1$. Since $\langle \tau_1, \ldots, \tau_{\text{last}} \rangle$ is a valid NT-class of length n, for every instant $u \in [\hat{\alpha}_{\text{last}}, m\hat{M}_{n-1}]$ there exists a valid evolution in the class such that τ_{last} fires at u, as $(\hat{\alpha}_{last} \leq \hat{\tau}_{last} \leq \hat{\beta}_{last}) \in D_{n-1}$.

Now, we consider two cases for $\tau_{n+1} = \tau_f$.

1. τ_f has been enabled by the firing of τ_{last}, *i.e.*, $\tau_f \in \hat{ET}_n \setminus \hat{ET}_{n-1}$. Then, its absolute firing interval is $[\hat{\alpha}_{\text{last}} + \alpha_f, m\hat{M}_{n-1} + \beta_f]$. However, the actual firing interval of τ_f is also bounded above by the firing time bounds of the other enabled transitions, so it is actually $[\hat{\alpha}_{\text{last}} + \alpha_f, m\hat{M}_N]$. Recall that $(\hat{\alpha}_f \leq \hat{\tau}_f \leq \hat{\beta}_f) \in \hat{D}_n$ and $\hat{\alpha}_f = \hat{\alpha}_{\text{last}} + \alpha_f$ by construction; this closes the current case.

2. τ_f was already enabled when τ_{last} fired, *i.e.*, $\tau_f \in (\hat{ET}_{n-1} \cap \hat{ET}_n)$. Now, the firing interval of τ_f is bounded from below by the firing interval of τ_{last}—which fires immediately before τ_f—and from above by the firing time bounds of the other enabled transitions. Hence, the firing interval of τ_f is $[\max\{\hat{\alpha}_{\text{last}}, \hat{\alpha}_f\}, m\hat{M}_n]$. Recall that $\hat{\hat{\alpha}}_f = \max\{\hat{\alpha}_{\text{last}}, \hat{\alpha}_f\}$ by construction, which concludes this case as well. □

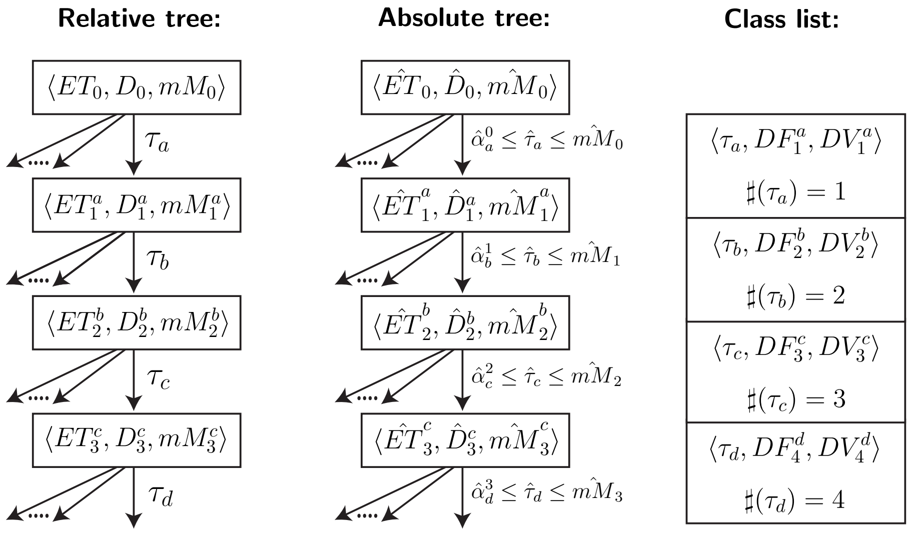
Corollary 11. *Let $\langle \tau_1, \ldots, \tau_i, \ldots, \tau_k \rangle$ be any k-NT-class of some TS S, with $i < k$, whose absolute tree $\mathcal{T}$ hosts the inequality $\hat{\alpha}_i \leq \hat{\tau}_i \leq \hat{\beta}_i$ for τ_i in $\hat{D}_{i-1}$. Then, there exist two real numbers m, M such that $\hat{\alpha}_i \leq m \leq M \leq m\hat{M}_{i-1}$ and, for all NT-evolutions in the chosen NT-class, τ_i fires within the interval $[m, M]$.*

2.4.3 Combining the Relative and Absolute Tree

Let us now show how to combine the information hosted by the relative and absolute trees to build a complete temporal profiling for NT-evolutions in an NT-class. The procedure can be repeated for all k-NT-classes to have a complete profiling of all NT-evolutions.

For clarity, we introduce the additional notational convention of superscripting bounds of the relative tree with a w, such as in $\alpha^w \leq \tau \leq \beta^w$, whereas the corresponding bounds in the absolute tree are hatted. Also, we introduce the operator $\sharp$ such that $\sharp(\tau) = i$ iff τ is the i-th transition firing in the considered NT-class.

The class list. Let us consider some TS S, its relative and absolute trees $\mathcal{T}^r, \mathcal{T}^a$, and some k-NT-class $\langle \tau_1, \dots, \tau_k \rangle$. We introduce a new data structure, called *class list*. See Figure 1 for reference.



Relative tree:
Absolute tree:
Class list:
$\langle ET_0, D_0, mM_0 \rangle$
τ_a
$\langle ET_1^a, D_1^a, mM_1^a \rangle$
τ_b
$\langle ET_2^b, D_2^b, mM_2^b \rangle$
τ_c
$\langle ET_3^c, D_3^c, mM_3^c \rangle$
τ_d
$\langle \hat{ET}_0, \hat{D}_0, \hat{mM}_0 \rangle$
$\hat{\alpha}_a^0 \leq \hat{\tau}_a \leq \hat{mM}_0$
$\langle \hat{ET}_1^a, \hat{D}_1^a, \hat{mM}_1^a \rangle$
$\hat{\alpha}_b^1 \leq \hat{\tau}_b \leq \hat{mM}_1$
$\langle \hat{ET}_2^b, \hat{D}_2^b, \hat{mM}_2^b \rangle$
$\hat{\alpha}_c^2 \leq \hat{\tau}_c \leq \hat{mM}_2$
$\langle \hat{ET}_3^c, \hat{D}_3^c, \hat{mM}_3^c \rangle$
$\hat{\alpha}_d^3 \leq \hat{\tau}_d \leq \hat{mM}_3$
$\langle \tau_a, DF_1^a, DV_1^a \rangle$
$\sharp(\tau_a) = 1$
$\langle \tau_b, DF_2^b, DV_2^b \rangle$
$\sharp(\tau_b) = 2$
$\langle \tau_c, DF_3^c, DV_3^c \rangle$
$\sharp(\tau_c) = 3$
$\langle \tau_d, DF_4^d, DV_4^d \rangle$
$\sharp(\tau_d) = 4$

Figure 1: Portions of the relative and absolute tree, and of the class list for the NT-class $\langle \tau_a, \tau_b, \tau_c, \tau_d \rangle$.

The list has size k, and every element is a triple:

$$\langle \tau_f, DF_i^f, DV_i^f \rangle$$

with $i = \sharp(\tau_f)$, where:

- τ_f is the i-th transition in the NT-class;
- DF_i^f is a set of constraints on the absolute firing time of τ_f, in the form:

  $$m_f^i \leq \hat{\tau}_f \leq M_f^i$$

  with m_f^i and M_f^i rational constant. Note that, since all inequalities in DF_i^f are in the only variable $\hat{\tau}_f$, we can actually consider DF_i^f as a unique inequality;
- DV_i^f is a set of constraints on the absolute firing times of pairs of transitions, in the form:

  $$\hat{\tau}_x + m_x^i \leq \hat{\tau}_f \leq \hat{\tau}_x + M_x^i$$

  with m_x^i and M_x^i rational constants, and τ_x such that $\sharp(\tau_x) < \sharp(\tau_f)$. Similarly as for DF_i^f, we can regard DV_i^f as a unique inequality.

Building the class list. First, all k elements in the class list are initialized as follows. For every triple τ_f, DF_i^f, DV_i^f:

- τ_f is such that $\sharp(\tau_f) = i$;
- $DF_i^f = \left\{\hat{\alpha}_f^{i-1} \leq \hat{\tau}_f \leq m\hat{M}_{i-1}\right\}$;
- $DV_i^f = \emptyset$.

Then, there are $k-1$ iterations of the following procedure, starting respectively from the last element in the list, from the second-to-last element, and so on down to the second one.

For the n-th element, let τ_f denote the corresponding transition such that $\sharp(\tau_f) = n$. The procedure performs the following steps:

1. To update DV_n^f, we walk backward on the path in the relative tree which starts from the node of depth $n-1$, until we reach some node at depth h such that $\tau_f \notin ET_h$. Then, for every node on the walked path (excluding possibly the root node) which hosts some inequality $\alpha_f^{\bar{x}} \leq \tau_f \leq \beta_f^{\bar{x}}$ in $D_{\bar{x}}$, where $\bar{x} = \sharp(\tau_x)$, we add the following inequality to DV_N^f:
$$\hat{\tau}_x + \alpha_f^{\bar{x}} \leq \hat{\tau}_f \leq \hat{\tau}_x + \breve{\beta}_f^{\bar{x}} \quad \text{where } \breve{\beta}_f^{\bar{x}} = \begin{cases} mM_{\bar{x}} & \text{if } \bar{x} = N-1 \\ \beta_f^{\bar{x}} & \text{if } \bar{x} < N-1. \end{cases}$$

2. For each pair of inequalities in DV_n^f of the form:
$$\begin{cases} \hat{\tau}_x + m_x^n & \leq \hat{\tau}_f & \leq \hat{\tau}_x + M_x^n \\ \hat{\tau}_y + m_y^n & \leq \hat{\tau}_f & \leq \hat{\tau}_y + M_y^n \end{cases}$$
with $\sharp(\tau_y) > \sharp(\tau_x)$ and $\bar{y} = \sharp(\tau_y)$, the following inequality is added to $DV_{\bar{y}}^y$:
$$\hat{\tau}_x + \max\left\{0, m_x^n - M_y^n\right\} \leq \hat{\tau}_y \leq \hat{\tau}_x + (M_x^n - m_y^n).$$

3. For each transition τ_x appearing in an inequality of DV_n^f of the form $\hat{\tau}_x + m_x^n \leq \hat{\tau}_f \leq \hat{\tau}_x + M_x^n$ with $DF_n^f = \{m_f \leq \hat{\tau}_f \leq M_f\}$, the following inequality is added to $DF_{\bar{x}}^x$:
$$m_f - M_x^n \leq \hat{\tau}_x \leq M_f - m_x^n.$$

Building an NT-evolution. Let us now briefly discuss how a particular NT-evolution can be built relying on the information the class list provides. We start from the first element in the list, where $\hat{\tau}_{f_1}$ is the only free variable in the inequalities in DF_1^1, and we pick a firing time $\hat{\tau}_{f_1}$ which respects all such inequalities. Then, we replace any occurrence of $\hat{\tau}_{f_1}$ in all inequalities in all DV sets in the class list with the chosen value, and we move the resulting inequality in the corresponding DF sets. Next, we move to the second element of the class list, and so on until we have built a complete set of firing times for our NT-evolution.

2.4.4 Properties of the Class List

The correctness of the class list construction is proved by the following lemmas.

Lemma 12. *Let $\mathcal{L}$ be the class list for a k-NT-class of a TS S. All NT-evolutions of the class respect the constraints of $\mathcal{L}$.*

Proof. From the properties of the relative and absolute trees, the inequalities in the class list are necessary conditions for every NT-evolutions. More precisely, at the initialization of the list the necessary constraints from Corollary 11 are added to the DF, while in step 1 of the algorithm for building the list the necessary constraints from Theorem 8 are added to DV.

Let us now consider a generic triple $\langle \tau_f, DF_i^f, DV_i^f \rangle$ in the class list. For every pair of constraints in DV_i^f of the form:

$$\begin{cases} \hat{\tau}_x + m_x \leq \hat{\tau}_f \leq \hat{\tau}_x + M_x \\ \hat{\tau}_y + m_y \leq \hat{\tau}_f \leq \hat{\tau}_y + M_y \end{cases} \tag{1}$$

such that $\sharp(\tau_y) > \sharp(\tau_x)$, we immediately realize that:

$$\begin{cases} \hat{\tau}_x + m_x \leq \hat{\tau}_y + M_y \\ \hat{\tau}_y + m_y \leq \hat{\tau}_x + M_x. \end{cases}$$

Moreover, from $\hat{\tau}_y > \hat{\tau}_x$, we get the constraint corresponding to step 2 of the algorithm:

$$\hat{\tau}_x + \max\{0, m_x - M_y\} \leq \hat{\tau}_y \leq \hat{\tau}_x + (M_x - m_y). \tag{2}$$

Thus, if an NT-evolution does not respect (2) it would not respect (1) either. The latter contradicts Theorem 8, hence every valid NT-evolution respects the constraints introduced by step 2. Finally, let $DF_i^f = \{m_f \leq \hat{\tau}_f \leq M_f\}$, and let us consider the constraints introduced in step 3 of the algorithm for building the class list. For every τ_e such that $(\hat{\tau}_e + m_e \leq \hat{\tau}_f \leq \hat{\tau}_e + M_e) \in DV_i^f$, the constraints are built as follows:

$$\begin{cases} m_f \leq \hat{\tau}_f \leq M_f \\ \hat{\tau}_e + m_e \leq \hat{\tau}_f \leq \hat{\tau}_e + M_e \end{cases} \Rightarrow \begin{cases} m_f \leq \hat{\tau}_e + M_e \\ \hat{\tau}_e + m_e \leq M_f \end{cases} \Rightarrow m_f - M_e \leq \hat{\tau}_e \leq M_f - m_e. \tag{3}$$

Hence, if an NT-evolution does not respect (3) it would not respect the constraints in DF_i^f either, and correspondingly the initialization constraints as well. The latter contradicts Corollary 11, hence every valid NT-evolution respects the constraints introduced by step 3 of the algorithm building the class list. □

Lemma 13. *Let $\mathcal{L}$ be the class list for a k-NT-class of a TS S. All NT-evolutions respecting the constraints of $\mathcal{L}$ are valid NT-evolutions of the chosen class.*

Proof sketch. The proof is straightforward from the proof of Lemma 12.

In particular, the iterations of the algorithm for building the class list start from the last element of the list, corresponding to the last transition. From Theorem 10 there exists a valid NT-evolution in the chosen class for every instant of the interval defined by the inequality for τ_{f_k} in $DV_k^{f_k}$, where $\sharp(\tau_{f_k}) = k$. Also, Theorem 9 entails that there exists some valid NT-evolution for the chosen class for every firing time of τ_{f_k} satisfying the inequality which links it to the firing time of $\tau_{f_{k-1}}$, with $\sharp(\tau_{f_{k-1}}) = k - 1$.

Let us still focus on the first iteration of the algorithm. Step 1 adds to $DV_k^{f_k}$ all constraints from Theorem 8, which are necessary for every NT-evolution. Step 2 combines these constraints in all possible ways to obtain constraints for transitions firing before τ_{f_k}. Step 3 combines the constraints generated in step 1 with the absolute firing time τ_{f_k} suggested by Theorem 10, so that the absolute firing interval of transitions firing before τ_{f_k} is suitably restricted.

Then, let us move to the iteration of the algorithm corresponding to the firing of τ_{f_i}, with $\sharp(\tau_{f_i}) = i < k$. As above, step 1 adds all constraints from Theorem 8. Also, in any i-NT-evolution of the chosen class (restricted to the first i steps) τ_{f_i} can fire at any instant of the interval given by the corresponding inequality in $DF_i^{f_i}$, and also at any instant satisfying the inequality added in step 1 which links it to the firing time of $\tau_{f_{i-1}}$. As a result of the previous iterations, every k-NT-evolution where τ_{f_i} has a firing time satisfying the constraints in $DV_i^{f_i}$ is a valid NT-evolution for the chosen class. Otherwise, τ_{f_i} could not fire at any instant of time satisfying the constraints in both $DV_k^{f_k}$ and $DF_k^{f_k}$, in contradiction with Theorems 9 and 10. $\square$

Finally, the two lemmas can be collected in the following.

Theorem 14. *Let $\mathcal{L}$ be the class list for a k-NT-class of a TS S. Any NT-evolution is valid for the chosen NT-class if and only if it respects all constraints in $\mathcal{L}$.*

2.4.5 Complexity of Real-Time Profiling

Let us quickly sketch a time complexity analysis for the class list building algorithm. The worst-case scenario is when all transitions are initially enabled. Then, every iteration of step 1 walks from the current node to the root node, hence overall step 1 takes a time proportional to $\sum_{i=1,\dots,k} i = O(k^2)$. Step 2 combines all pairs of inequalities in DV; in the worst case the i-th element in the list has exactly $i-1$ inequalities in DV. Hence overall step 2 takes a time proportional to $\sum_{i=1,\dots,k}(i-1)^2 = O(k^3)$. In step 3 every iteration takes a time proportional to the number of inequalities in DV, hence overall it takes time $\sum_{i=1,\dots,k}(i-1) = O(k^2)$.

All in all, building the class list for a single k-NT-class takes time $O(k^3)$, while building the class list for all possible k-NT-classes of a TS S takes time $O(n^k k^3)$.

2.5 Timed Petri Nets

Timed Petri Nets (TPNs) [22] are a popular real-time extension of the classical Petri Nets [24], introduced by C. A. Petri in 1962 as a modeling tool for discrete event systems. TPNs are devoted to deal with real time systems and they can be seen as a form of STSs. In the following, we describe first how to generalize TSs to STSs, and then how to describe TPNs as STSs.

2.5.1 From TSs, to STSs, to TPNs

In TSs, each transition, when it fires, disables all the occurrences of some transitions and enables a certain number of other occurrences. Such a firing semantics can be extended by introducing explicitly the concept of *state*.

Definition 15. A *state/transition system* (STS) S is a quadruple $\langle T, Q, q_0, \delta \rangle$, where:

- T is the set of transitions of the system;
- Q is the set of states of the system (possibly infinite);
- $q_0 \in Q$ is the initial state of the system, *i.e.*, the state of the system at time $t = 0$;
- $\delta : Q \times T \times \mathbb{Q}_{\geq 0} \to Q$ is the (partial) transition function; $\delta(q, \tau, t) = q'$ denotes that the system in state q, with transition τ firing at time t moves instantaneously to state q'.

The state of a system is defined as follows.

Definition 16. A *state* $q \in Q$ of a STS $S = \langle T, Q, q_0 \rangle$ is a pair $\langle \mathrm{id}, \{\langle \tau_i, x_i \rangle\}_i \rangle$ where:

- id is a unique identifier;
- $\{\langle \tau_i, x_i \rangle\}_i$ a set of couples $\langle \tau_i, x_i \rangle$, where:
  - τ_i is an occurrence of a transition in T, which is enabled in q;
  - x_i is the absolute enabling time of τ_i.[6]

The system evolves from a current state q_n to the next state q_{n+1} as a result of the firing of some transition τ_f, enabled in q_n. The firing time must be feasible, *i.e.*, it must be included in the dynamic firing interval of τ_f and it must be less than or equal to the minimum of the maximum firing times of the transitions enabled in q_n. The firing will disable τ_f, and possibly some other transitions enabled in q_n, and it will enable some new transitions.

We remark that STSs are a generalization of TSs. In fact, according to the above definition, two different states may have the very same set of transitions and enabling time instants, and differ only by their identifiers. This was not possible with TSs where an implicit "state" was simply identified by a set of enabled transitions and enabling times. Another difference of STSs with respect to TSs is that in the former the effect of a firing (that is the choice of the next state) depends in general on the current state, the fired transition, and its firing time, rather than just on the firing transition as it was the case with TSs.

Now, we introduce timed Petri nets and show how to describe them as STSs.[7]

Definition 17. A *timed Petri net* is a tuple $\langle P, T, B, F, M_0, \mathrm{SIM} \rangle$, where,

- P is a finite non-empty set of places (a.k.a. locations);
- T is a finite non-empty set of transitions;
- $B : P \times T \to \mathbb{N}$ is the backward incidence function (BIF);

[6]Correspondingly, the dynamic firing interval of τ_i is $[x_i + \alpha_i, x_i + \beta_i]$.

[7]The terminology used in the real-time modeling literature sometimes distinguished between *timed* Petri nets and *time* Petri nets. Both are real-time extensions of standard Petri nets, but with different semantics. For uniformity, in this paper we call our model *timed* Petri, and we refer to [3] for a discussion of the subtle differences between model variants.

- $F : T \times P \to \mathbb{N}$ is the forward incidence function (FIF);
- $M_0 : P \to \mathbb{N}$ is the initial marking;
- SIM : $T \to \mathbb{Q}_{\geq 0} \times \mathbb{Q}_{\geq 0} \cup \{\infty\}$ is the static interval function.

Figure 2 provides some intuition about the graphical representation of a TPN. Places are pictured as discs and labeled $P_1, P_2, \ldots$; a shaded box represents transition T. Arrows link transition to places (according to the forward incidence function), and places to transitions (according to the backward incidence function). The initial marking is represented by tokens filling some places, according to the mapping M. Finally, the static firing interval is depicted nexto to each transition, in accordance with the function SIM.

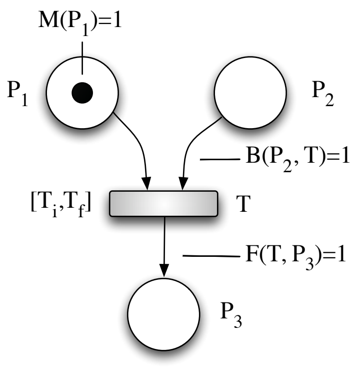


Figure 2: A timed Petri net fragment

Let us now describe how to formalize a TPN by means of a STS. Basically, for each transition $\tau_i \in T$, the static interval function gives the static firing interval, *i.e.*, SIM$(\tau_i) = [\alpha, \beta]$. If x_i is the enabling instant of a transition τ_i, the dynamic firing interval is $[\alpha + x_i, \beta + x_i]$. Correspondingly, we can associate introduce a notion of state to describe the evolution of TPNs.

Definition 18. A *state* s of a TPN is defined as a pair $\langle M, I \rangle$, where:

- $M : P \to \mathbb{N}$ is the current marking function;
- I is the set of firing intervals of the transitions that are enabled in the current state, described as $\hat{\alpha}_i \leq \tau_i \leq \hat{\beta}_i$. These intervals are relative to the instant at which the TPN reached the current state; $\hat{\alpha}_i$ and $\hat{\beta}_i$ are the earliest and the latest firing time, respectively.

Given a current state $s = \langle M,\ I \rangle$ of a TPN, for all transitions $\tau_i \in T$, the number n of instances of enabled transitions τ_i in s must be such that $M(p_j) - nB(p_j, \tau_i) \geq 0$, for all places $p_j \in P$. This gives a necessary condition for a transition to fire. Then, unless no other transition firing modifies the current state, a transition cannot fire before its earliest firing time and must fire before the minimum latest firing time of the transitions enabled in s. If many occurrences of the same transitions are enabled in a state, only one can fire in s. This description is enough to formalize a TPN as a STS.

2.5.2 Profiling Timed Petri Net Behavior

We have shown that TPNs fit the definition of STSs. As a consequence, let us now show how to apply the analysis technique of Section 2.3 directly to TPNs.

All we need is a procedure to update the state of a TPN as a consequence of a transition firing. So, let us consider a TPN in a state $s = \langle M, I \rangle$ at time t_s. Let us assume that some transition τ_x can fire at time $t_s + \theta$. The state $s' = \langle M', I' \rangle$, reached as a consequence of τ_x firing at the relative time instant θ, is computed as follows.

1. for all places $p_i \in P$: $M'(p_i) = M(p_i) - B(p_i, \tau_x) + F(\tau_x, p_i)$;
2. I' is obtained from I as follows:
   (a) remove from I all inequalities corresponding to the transitions disabled by τ_x; such transitions are those enabled in $M(\cdot)$ but not in $M(\cdot) - B(\cdot, \tau_x)$, including τ_x;
   (b) translate the time bounds of every inequality by θ; that is every inequality $\alpha \leq \tau \leq \beta$ becomes:
   $$\begin{aligned} \alpha' &= \max\{0, \alpha - \theta\} \\ \beta' &= \beta - \theta \end{aligned}$$
   (c) add to I all inequalities corresponding to transitions enabled by τ_x; such transitions are those not enabled in $M(\cdot) - B(\cdot, \tau_x)$ and enabled in $M'(\cdot) = M(\cdot) - B(\cdot, \tau_x) + F(\tau_x, \cdot)$.

An example of TPN modeling and (informal) analysis. Consider, as example, the consumer/producer model presented in the introduction. It can be modeled formally by the TPN of Figure 3. Such a TPN is defined by the tuple $\langle P, T, B, F, M_0, \text{SIM} \rangle$, where:

- $P = \{L_0, L_1\}$;
- $T = \{p, c\}$;
- $B(L_0, p) = 1$, $B(L_0, c) = 0$, $B(L_1, p) = 0$, and $B(L_1, c) = 1$;
- $F(p, L_0) = 1$, $F(p, L_1) = 1$, $F(c, L_0) = 0$, and $F(c, L_1) = 0$,
- $M_0(L_0) = 1$ and $M_0(L_1) = 0$;
- $\text{SIM}(p) = [A_p, B_p]$ and $\text{SIM}(c) = [A_c, B_c]$

Let us assume initially $A_p = A_c = B_p = B_c = 1$. Then, no transition is initially enabled, but after exactly 1 time unit, since L_0 is initially marked, p will become enabled. At that time instant, p has to fire because it is the only enabled transition and its latest firing time equals its earliest firing time. When p fires, the net goes to a new state where L_0 and L_1 are marked and no transition is enabled. After 1 additional time unit, both p and c are enabled and they both have to fire, again because their firing intervals are singletons. Conceptually, they both fire at the same time, but technically one of the two fires first, without consuming any time, and the systems goes in a new state

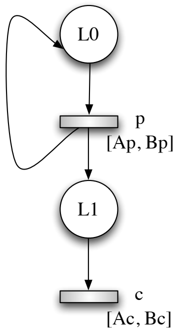


Figure 3: Consumer/producer example modeled with a TPN

in which only the one that did not fire is still enabled. In this new state the enabled transition has to fire immediately and the TPN goes back to the initial marking.

This trivial behavior changes significantly if we enlarge the firing time intervals. Take, as second example, $A_p = 1$, $B_p = 2$, $A_c = 3$ and $B_c = 4$. Transition p is enabled after 1 time unit from the beginning and it can fire at any time instant between 1 and 2. After it fires at some $t_0 \in [1, 2]$, both L_0 and L_1 are marked, and consequently p can fire again between at any time between $t_0 + 1$ and $t_0 + 2$, while c will become enabled only at $t_0 + 3$. This means that p will have fired twice before c will have fired for the first time. This behavior accumulates over time so that it is easy to see that the number of markings in L_1 grows unbounded over time.

3 Case Study

In this section we employ the proposed technique to analyze the behavior of a TPN, modeling the implementation of an active braking control system. Specifically, we will show how the analysis tools presented in this work can provide valuable information for the verification of several aspects of embedded safety-critical control systems, such as a braking controller.

The idea is that, once the structural properties of the closed-loop system have been formally proved via Control Theory methods, before moving to the industrialization of such control system several other issues need to be taken into account.

In our case, the control code would be sufficiently simple to guarantee that, once the controller is turned into control code, no structural problems, such as deadlocks, occur. In more complex cases this implementation step could be validated through techniques such as Model Checking.

Then, it is necessary to fix the hardware lay-out of the Electronic Control Unit (ECU) which hosts the control code, and to design the whole vehicle network, *i.e.*, the physical connection between the sensors, the ECU itself and the actuators. Usually, in Automotive applications, a CAN bus is used for signal transmission, [17]. Like many serial communications protocols, the CAN protocol transmits frames of data as a temporal sequence of bits, whereas information is encoded by alternating the medium between two possible voltage levels.

In control networks, communication protocols can be divided according to two fundamentally different principles of message transfer. These are synchronous (time-triggered) and asynchronous (event-triggered) models. In an event-triggered communication system, transactions are triggered by the occurrence of some external events, such as the pressing of a button or the firing of a fuel injector. This type of communication is essentially sporadic in nature where, for real-time operation, the message release period is quantified between worst-case and best-case time scenarios.

In a time-triggered system, message transactions are initiated based on the progression of time. Using such a principle each node member of the network is assigned a time when it may transfer a message on the network. Such a model is similar to a TDMA (Time Division Multiple Access) scheme with each node respecting the sense of global time which exists in the network ensemble. Each node possesses a local clock and a message transaction schedule which dictates when it may transmit and when it may expect to receive a particular message.

Recently, the CAN buses devoted to safety-critical applications employ a dedicated protocol, the Time Tiggered CAN (TTCAN), [19], which enforces real-time synchrony among the signals. The TTCAN is a time-triggered protocol, where the exchange of messages is controlled essentially by the temporal progression of time. The exchange of a specific message may only occur at a predefined point in *relative* time during time-triggered operation of the protocol. The static schedule sequence is based on a Time Division Access (TDA) scheme whereby message exchanges may only occur during specific time slots or time windows.

It is worth noting that, when a controller has been proved to provide stability and robustness properties of the closed-loop system in a control-theoretic sense, these properties hold with the implicit assumption that the inputs and outputs of the control system are available synchronously at predefined time instants, and that the controller performs the needed computations in a *nominal* predefined time interval. Clearly, when moving to the final hardware, the overall embedded system must be proved to preserve such properties. So, as both signal transmission and ECU processing can only be in fact quantified to happen within a certain time interval, modeled as a best-case/worst-case scenario, the TPN-based formal verification can provide crucial information on all the possible system behaviors (note that, now, with *system* we indicate the control code embedded in the vehicle network). First of all it provides the equivalence classes to which the *nominal* behavior belongs. By looking at these classes in terms of upper-bound on the time intervals, the designer is given a formal indication which may guide hardware sizing; in fact, if cost-constraints are tight, as it is usually the case in Automotive applications, the idea is to seek for the cheapest hardware which is capable of guaranteeing the needed performance level.

Further, it is interesting to model and prove the fact that, when the control code routine starts, all the data provided as inputs refer to the same measurement interval (*input* consistency) and that the output sent to the actuators are also read with correct timing (*output* consistency).

The section is now devoted to briefly illustrate the considered application, which will be then formally described with a model suitable for the TPN analysis framework. Then, we explicitly consider the upper and lower bounds on the various steps of the code execution—from sensor measurements to actuators inputs—and we extract the NT-classes for the given model. Finally, we discuss how, thanks to these analysis tools, we can formally prove the semantic-preservation properties of the overall control code and data flow and gain information which can guide the hardware sizing for the final implementation.

3.1 Application Description

The considered application is an ABS controller based on a Hydraulic Braking System (HAB) actuator, which is the standard braking system on most commercial cars. Clearly, ABS controllers are indeed a safety critical application, as they have to take care of managing panic-brakes which cannot be handled by common drivers. The design of an appropriate controller which is capable of inducing a stable limit cycle on the wheel slip, *i.e.*, the difference between the vehicle and wheel speed (which is the controlled variable in most ABS systems) has been presented in [29] and [28]. In these works it has been shown that the proposed controller can guarantee the desired stability properties of the closed loop system based on system theoretic techniques. Here, we only briefly illustrate the controller structure, which is described by means of a Finite State Machine (FSM), as it is the starting point for the following formal analysis of its behavior when implemented on the vehicle ECU. In this respect, we will also present the physical layout of the considered vehicle network, as it will serve as a basis for the NT-classes derivation.

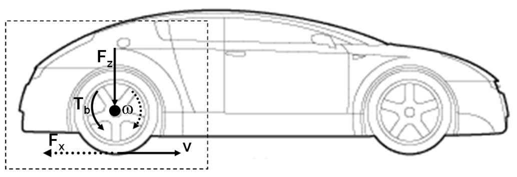


Figure 4: Schematic view of the single-corner model.

System and actuator model. The braking dynamics have been modeled based on a quarter car model, [11], [26]. The model is given by the following equations (see also Figure 4)

$$\dot{\lambda} = -\frac{1-\lambda}{J\omega}\left(\Psi(\lambda) - T_b\right), \tag{4}$$

with $\omega > 0$ and

$$\Psi(\lambda) = \left(r + \frac{J}{rm}(1-\lambda)\right) F_z \mu(\lambda). \tag{5}$$

where λ is the longitudinal slip, which—during braking—is defined as $\lambda = (v - \omega r)/v$, $\lambda \in [0, 1]$, $\omega\,[rad/s]$ is the angular speed of the wheel, $v\,[m/s]$ is the longitudinal speed of the vehicle body, the function $\mu(\lambda)$ describes the tire-road friction conditions, $T_b\,[Nm]$ is the braking torque, $F_z\,[N]$ is the vertical load, $J\,[kg\,m^2]$, $m\,[kg]$ and $r\,[m]$ are the moment of inertia of the wheel, the quarter-car mass, and the wheel radius, respectively.
As mentioned, the considered actuator is an HAB, [25] in which the pressure exerted by the driver on the pedal is transmitted to the hydraulic system via a Build Valve which communicates with the brake cylinder (see Figure 5). The

hydraulic system has a second valve, the Dump Valve, which discharges the pressure and which is connected to a low pressure accumulator. According to

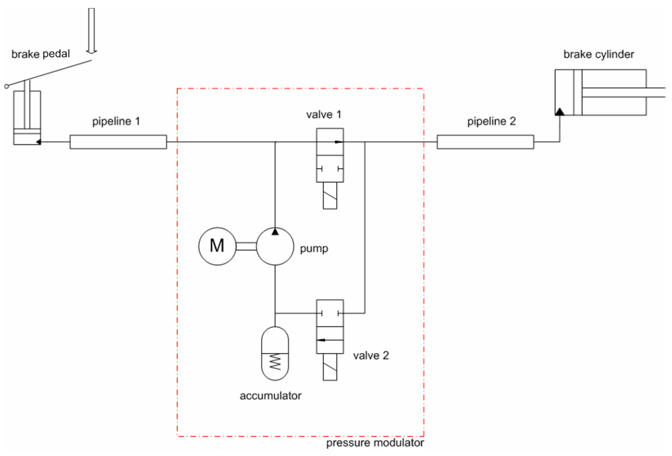


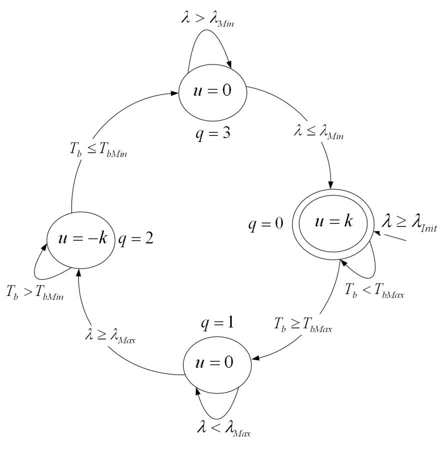
Figure 5: Qualitative scheme of the considered braking system.

its physical characteristics, the HAB actuator is only capable of providing three control actions. *Increase* the brake pressure: in this case the Build valve is open and the Dump one closed; *Hold* the brake pressure: in this case both valves are closed; *Decrease* the brake pressure: in this case the Build valve is closed and the Dump one open. Finally, we assume a static brake-pads friction model, *i.e.*, the braking torque T_b is computed from the measured brake pressure p_b as $T_b = r_d\,\chi\,A\,p_b$, where r_d is the brake disk radius; χ is the (constant) brake pad friction coefficient; A is the brake piston area and p_b is the measured brake pressure. The *increase* and *decrease* pressure actions are physically limited by the actuator rate limit, *i.e.*, $\frac{dT_b}{dt} = k$, where the rate limit $k \in \mathbb{R}^+$ is a known parameter. Its nominal value in the following simulations will be set to $10\,kNm/s$, [23]. Accordingly, the controller design is based on the hybrid system made of the connection between the wheel dynamics (4) and the hydraulic actuator, namely

$$\dot{\lambda} = -\frac{1-\lambda}{J\omega}\left(\Psi(\lambda) - T_b\right); \quad \dot{T}_b = u, \tag{6}$$

where $u = \{-k, 0, k\}$ and $\Psi(\lambda)$ is as in (5). According to the value of the control variable u we have three possible closed-loop dynamics, where $u = -k$ corresponds to the *Decrease* control action, $u = 0$ to the *Hold* control action and $u = k$ to the *Increase* control action.
As shown in [29], a controller which guarantees closed-loop stability of the braking system is that shown in Figure 6.

By analyzing Figure 6 one can see that we employ the braking torque T_b and the wheel slip λ as switching variables and that the switching conditions change according to the current discrete controller state $q = \{0, 1, 2, 3\}$. The thresholds values T_{bMin}, T_{bMax}, λ_{Min}, and λ_{Max} which enable the transitions between the different states are chosen by the designer to guarantee a good trade-off between performance and safety in all driving conditions. Moreover note that, due to the specific application, we can assume that (in normal operating conditions) the controller—upon activation—enters the state $q = 0$ associated with the *Increase*



$\lambda > \lambda_{Min}$
$u = 0$
$q = 3$
$\lambda \leq \lambda_{Min}$
$T_b \leq T_{bMin}$
$q = 0$
$u = k$
$\lambda \geq \lambda_{Init}$
$u = -k$
$q = 2$
$T_b > T_{bMin}$
$T_b < T_{bMax}$
$q = 1$
$T_b \geq T_{bMax}$
$\lambda \geq \lambda_{Max}$
$u = 0$
$\lambda < \lambda_{Max}$

Figure 6: Finite State Machine description of the hybrid controller.

control action. In fact, when the braking maneuver begins, the controller is usually activated when the wheel slip λ reaches a predefined threshold value $\lambda_{Init} < \lambda_{Min}$. Finally, note that the hybrid controller is composed of four discrete states $q = \{0, 1, 2, 3\}$, each of which has an associated control action, that is $u_{q=i} = \{k, 0, -k, 0\}$, $i = 0, \ldots, 3$.

3.2 Formalization and Analysis

Before introducing the formalization of the considered controller, we briefly describe the hardware/software lay out in the final vehicle implementation, which is shown in Figure 7. As can be seen, the vehicle is equipped with 5 sensors, *i.e.*, four encoders to measure the wheel speeds, by means of which the wheel slip λ is obtained and a pressure sensor, which allows to compute the braking torque T_b. By processing such sensor measurements, the control algorithm can be executed to provide the control action. As far as computational resources are concerned, the vehicle is equipped with an ECU which processes sensors outputs and executes the controller routine. Finally, the controller output is sent to the HAB which actuates it. The signal communication is managed by the vehicle BUS, which is assumed to work according to the TTCAN protocol. As for nominal performances—stated in terms of timing of the different actions—generally on car networks the CAN bus works with a nominal time interval which is either of 5 or 10 ms (100-200 Hz of sampling frequency), and the braking controller should guarantee to complete the computations accordingly. Note that, usually, vehicle ECU possess very simple operating systems, in which there are different time slices whose duration is an integer multiple of the smallest (generally down to 1 ms), and each application is assigned one fixed time slice according to the priority of the performed operation, which is related to the associated safety level, *e.g.*, air conditioning is computed every—say—1 s, while the braking controller every 5 ms.

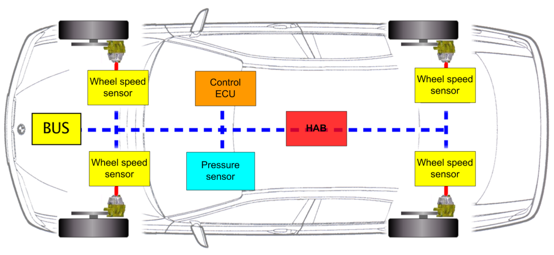


Figure 7: Vehicle network lay-out.

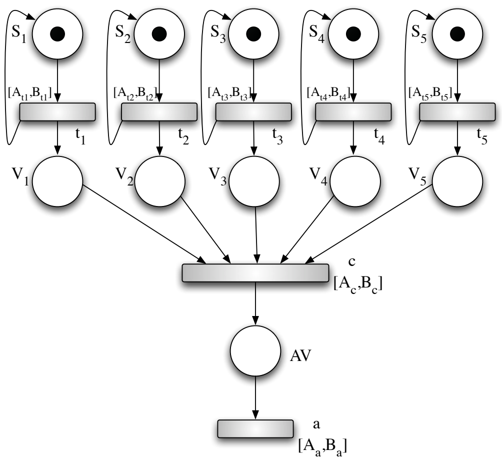


Figure 8: TPN modeling the case study.

3.2.1 TPN Model

Considering the description of the hardware/software lay out of the vehicle in Figure 7 and of the controller, the introduced case study can be modeled using the TPN shown in Figure 8.

The TPN is composed of 11 places and 7 transitions. Each place labeled with S_i represents a sensor devoted to measuring either the speed of a wheel or the braking pressure. The marking in this kind of place means that the correspondent sensor has just collected a datum. Each place S_i is connected with a transition t_i that is enabled with a firing interval $[A_{t_i}, B_{t_i}]$. A transition t_i models the transmission of the collected datum to the control system and, once the datum is collected, it has to fire after A_{t_i} time units and within B_{t_i} time units. When such a transition fires, a fresh token is placed both in the place S_i and in the place V_i. The token in S_i means that the sensor collects a new datum as soon as it has sent the old one, while the token that reaches V_i represents the acquisition, made by the control system, of the last datum provided by the sensor i. Notice that we do not distinguish among different tokens. Thus, if a token is already present in V_i, once t_i fires and a new token reaches V_i, the two tokens in V_i are undistinguishable.
Transition c represents the computation of the value to be actuated and this is why it needs a token in all the locations labeled with V_i to be enabled. Indeed,

in order to perform this calculus the control system needs the data from all the sensors. The firing of c generates a token in place AV, that means the actuation value is ready to be applied and it will be actuated when transition a fires. Currently we consider the static firing intervals as symbolic variables where, for each transition k, $A_k \leq B_k$, and henceforward we will assign to them different values in order to show how they can influence the system behavior.

3.2.2 TPN Analysis

Since, as already noticed, we cannot distinguish among tokens in a place according to their arrival time, in many parts of the net the values assigned to the firing intervals are crucial. In other words, the temporal behavior of the components of the braking system influence the correctness of its implementation. Let us now carry on a guided analysis of such correctness features by discussing informally the results of various runs of our enumeration algorithm on the presented TPN model. We will see how these results map directly back on features of the controller implementation that should be assessed at design time. Throughout this section, all time units are in milliseconds, unless otherwise specified.

Consider for example the following values for the marking function $M(V_1) = M(V_2) = M(V_3) = 1$, $M(V_4) = 1$, $M(V_5) = 2$ and M equal to 0 for all the other places. This situation can be easily obtained from the initial marking if the upper bound B_{t_5} of transition t_5 is half of the lower bound A_{t_i}, with $i = 1, 2, 3, 4$ of transitions t_1, t_2, t_3 and t_4. In this situation, when transition c fires, it consumes the single tokens in V_1, V_2, V_3 and V_4, and one of the two tokens in V_5. However, these tokens correspond to two different acquisitions of data by the corresponding sensor, whereas the calculation performed by c would be coherent only if using the more recent one.

This example suggests the introduction of requirements on the firing intervals such that the following properties are met by all evolutions:

- once a datum is collected, the control action is either performed or the datum is discarded and the actuation is eventually performed within a fixed amount of time;
- the calculation of the value to be actuated is performed on coherent data (*input consistency*) and using the most recent ones.

A very simplified hypothesis that meets these requirements considers only deterministic firing intervals. Determinism is obtained by defining, for each interval, the lower bound equal to the upper bound. Moreover we impose all sensors intervals to be equal[8], $A_c = B_c \geq A_a = B_a$, and $A_{t_i} \geq A_c + A_a$. This means that we are assuming a controller that computes the actuation value more slowly than the actuator performs the control action, and sensors that produce data more slowly than the whole sequential process of computing and performing the actuation. In this case, one can check through the analysis tool that the number of possible tokens in each place will be bounded by 1. This means that, considering a place p with a token in it, the preceding transition takes more time to produce a new token than the time needed by the following transition to consume the one in p. Since by definition two transitions cannot

[8]Notice that this constraint is not necessary to guarantee the system requirements but it is useful for illustration purposes.

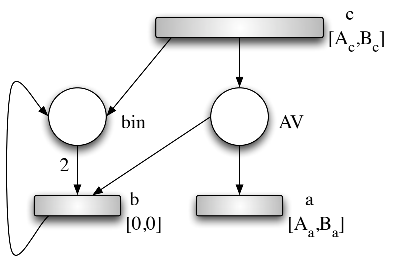


Figure 9: A part of the TPN modified to implement the bin technique.

fire simultaneously, working with these constraints, the number of obtained NT-classes is $(5!)^n$, where n is the number of times the control action is performed. Indeed, the constraint $A_{t_i} \geq A_c + A_a$ guarantees that no datum is collected before the actuation due to the previous data is performed. Hence the only possible inversion among the firings of the transitions is due to the firing of the transitions representing the transmission of the data collected by the sensors. This means that practically there is only one equivalence class where all the sensors fire at the same time instant. A run of our algorithm with the sample values $A_{t_i} = B_{t_i} = 10$, $A_c = B_c = 5$ and $A_a = B_a = 4$ confirms the analysis sketched above.

Let us now relax the hypothesis on the sensors and only require $A_c = B_c \geq A_a = B_a$, $\max_{i\in\{1,2,3,4,5\}}(A_{t_i}) \geq A_c + A_a$ and $\max_{i\in\{1,2,3,4,5\}}(A_{t_i}) - \min_{i\in\{1,2,3,4,5\}}(A_{t_i}) \leq \min_{i\in\{1,2,3,4,5\}}(A_{t_i})$. In this case, we cannot guarantee the bound on the number of tokens in the places. The last constraint suggests that the time taken by the sensors to collect data varies within a small interval; however, since the sensors work in parallel, after a while the faster sensor will collect data that it cannot consume. Let us notice, however, that the hypothesis of having identical sensors is quite realistic, whereas the hypothesis of determinism stated above is quite unrealistic, because it implies that all the element of the system are flawless instruments that always perform with exact timing. Hence, consider the case in which all the sensors are identical, $A_c \geq B_a$ and $A_{t_i} \geq B_c + B_a$, but the determinism constraint is relaxed. Now, the number of tokens in each place is still bounded by 1, and furthermore we can find interesting information about the NT-classes. Namely, even in this case we have $(5!)^n$ NT-classes, but we notice that a complete control flow, from data acquisition to actuation, can be performed by different control actions that are significantly different in terms of execution time.

Even more significant are the cases in which we relax all the constraints and, as a consequence, we cannot guarantee a bound on the number of tokens in the places. Consider for example the case in which, all the sensors and actuators behave equally with [8.5, 10] as firing interval (the CAN bus work nominally at 100 Hz), and the computation (transition c) occurs within the time interval [4, 5] (the control routine is assigned a nominal time slice of 5 ms by the operating system). We find NT-traces of length 8 such as $\langle t_1, t_2, t_3, t_4, t_5, c, t_1, t_2 \rangle$, consuming from 17 to 18 time instants overall. The corresponding classes are constrained by absolute intervals such that all traces within the interval are equivalent. and these constraints guarantee all the firing time instant in the obtained intervals are equivalent. The absolute firing intervals for these example traces are: [8.5, 9.5] (for t_1), [8.5, 9.5] (for t_2), [8.5, 10] (for t_3), [8.5, 10] (for t_4), [9, 10] (for t_5), [14, 15] (for c), [17, 18] (for the second occurrence of t_1), and [17, 18] (for the second occurrence of t_2).

In these cases, we have to guarantee that the accumulated extra (old) tokens are discarded. Consider for instance the previous numerical example, where more than one token can accumulate in AV because the actuator is slower than the computer. These situations are likely to occur in practice, where the transmission delays from the controller out to the actuators are usually longer than the time it takes to complete a control cycle.

In a proper implementation, only the most recent datum should be retained, while the others have to be discarded. In an actual hardware implementation, a straightforward solution to this problem would allocate a single memory slot containing the latest provided datum. Then, when a new datum is computed it simply overwrites the old one.

In terms of TPN, this solution can be modeled as shown in Figure 9. We call this approach *bin technique*, since the stale data is discarded through a new transition b which acts as a bin. This additional transition b is devoted to consuming the extra tokens in AV: the idea is to leave a enabled by one token in AV but, at the same time, additional tokens should be consumed by b as soon as possible. In fact, whenever a new token is produced by transition c, two copies of it simultaneously go in AV and bin. Then, the weight 2 on the arc that connects bin with b means that transition bin is enabled by, and consumes when it fires, exactly two tokens from bin. Since every token produced by c is in both bin and AV, b is enabled if and only if there are (at least) two tokens in both places bin and AV. When this is the case, b fires immediately, consumes two tokens from bin and one from AV, and puts one new token in bin. Overall, AV is left with one token and so is bin, so the two places always have the same number of tokens, and always contain at most two tokens. Hence, the bin technique guarantees a bound for the number of tokens in every place. Also, the token in AV has enabled continuously transition a, which will fire nondeterministically according to its firing interval. In other words, the token discarding mechanism has been implemented in such a way that it does not interfere with the regular functioning of the net.

Additional numeric experiments on the modified TPN show clearly that the desired properties of a correct data collection and coherent control action are guaranteed in a system where $B_{t_i} = B_{t_j} = B_a > A_c$ for all $i, j = 1, \ldots, 5$, as long as we adopt the bin technique. This means that a realistic scenario, where: (1) the times needed to read data from sensors are roughly the same as those needed to transmit the control choices to the actuators, and (2) the time to compute one cicle of control action is smaller than the latter times, is compatible with a correct overall implementation of the control algorithm.

4 Lessons Learned

Now that the proposed methodology has been discussed with reference to a realistic case study, some remarks in the form of a critical summary of the overall analysis tools and of their expressiveness are in order.

First of all it is worth noticing that, to the best of our knowledge, this is one of the first attempts to use formal methods (in the software-engineering sense [5]) with the aim of supporting a real design step, rather than providing only yes/no answers on some specific properties of a system, or about the satisfaction of some requirements. As a matter of fact, using the proposed tool, the information it provides may induce to reconsider the control design if certain timing constraints are proved to be more or less severe than others or if they need higher priority in the final system. These additional insights pay back the loss in terms of full automation of the analysis phase. Moreover, the analysis also highlighted the flexibility of the approach, as we showed how custom properties *e.g.*, input and output consistency, could be enforced by extending the base model.

Moreover, as it was shown by means of the case study analysis, it is rather straightforward to exploit the information provided by the analysis tool as a guide to hardware sizing. Section 3 revealed, in fact, that for the considered application the nominal controller behavior can be guaranteed by choosing an ECU which guarantees that the control routine executes in a shorter time with respect to the sensor and actuators signal transmission. This suggests that more resources should be devoted to the microprocessor selection, while a slower (hence cheaper) bus can be selected. Also, by evaluating the upper and lower bounds on the admissible transition firings, a certain degree of nondeterminism in communication protocols can also be accepted.

Finally, note that the computational time needed to enumerate the NT-classes can also be significantly reduced by clustering components which are known to share the same behavior. Consider, as an example, the five sensors of our case study. To evaluate possible differences in their timing behavior, two instances of such sensors would in principle have been sufficient. In this way, the number of NT-classes significantly reduces, without losing any significant information.

Of course, the final goal is that such tools can find place in industrial practice. To this aim, the most crucial step would be to work on the development of software interfaces which can provide the possibility of performing formal analysis of the control code at a high level of abstraction, for example communicating directly with control software, such as Matlab/Simulink and the like. For such purposes, specific wrapper tools should be developed, capable of translating specific control architectures specified, for example, in terms of finite state machines.

5 Concluding Remarks

This paper presented a tool-supported approach to the formal analysis of real-time aspects in controller implementation. The analysis showed how to model the overall embedded control architecture, considering sensors, actuators and signal communications, and to prove that some desired properties of the control loop are preserved in its implementation on a distributed architecture. Notably, the information extracted automatically from the model can also be used to approach some *design* problems, such as the hardware sizing in the final implementation. The effectiveness of the approach has been assessed on a realistic case study describing an active braking controller.

References

[1] Bernard Berthomieu and Michel Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, 1991.

[2] Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time Petri nets. In R. E. A. Mason, editor, *Proceedings of the IFIP 9th World Computer Congress on Information Processing*, pages 41–46. North-Holland/IFIP, 1983.

[3] Antonio Cerone and Andrea Maggiolo-Schettini. Time-based expressivity of time Petri nets for system specification. *Theoretical Computer Science*, 216(1–2):1–53, 1999.

[4] S. Cigoli, P. Leblanc, S. Malaponti, D. Mandrioli, M. Mazzucchelli, A. Morzenti, and P. Spoletini. An experiment in applying UML 2.0 to the development of an industrial critical application. In *Proceedings of CSDUML'03*, 2003.

[5] Edmund M. Clarke and Jeannette M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[6] J. Ellsberger, D. Hogrefe, and A. Sarma. *SDL: Formal Object-Oriented Language for Communicating Systems*. Prentice Hall, 2nd edition, 1997.

[7] Claude Girault and Rüdiger Valk. *Petri Nets for System Engineering*. Springer-Verlag, 2003.

[8] Constance Heitmeier and Dino Mandrioli, editors. *Formal Methods for Real-Time Computing*. John Wiley & Sons, 1996.

[9] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[10] Thomas A. Henzinger and Joseph Sifakis. The embedded systems design challenge. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *Proceedings of the 14th International Symposium on Formal Methods (FM'06)*, volume 4085 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2006.

[11] T.A. Johansen, I. Petersen, J. Kalkkuhl, and J. Lüdemann. Gain-scheduled wheel slip control in automotive brake systems. *IEEE Transactions on Control Systems Technology*, 11(6):799–811, November 2003.

[12] T.L. Johnson. Improving automation software dependability: A role for formal methods? *Control Engineering Practice*, 15:1403–1415, 2007.

[13] Bruce H. Krogh. From analysis to design. In Jean-François Raskin and P. S. Thiagarajan, editors, *Proceedings of the 5th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'07)*, volume 4763 of *Lecture Notes in Computer Science*, page 4. Springer-Verlag, October 2007. Invited talk.

[14] M. Kwiatkowska, G. Norman, and D. Parker. Controller dependability analysis by probabilistic model checking. *Control Engineering Practice*, 15:1427–1434, 2007.

[15] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1–2), 1997.

[16] M. Lawford and W. M. Wonham. Equivalence preserving transformations for timed transition models. *IEEE Transactions on Automatic Control*, 40(7):1167–1179, 1995.

[17] W. Lawrenz. *CAN System Engineering: From Theory to Practical Applications*. Springer, 1997. ISBN 0-387-94939-9.

[18] Edward A. Lee. Absolutely positively on time: What would it take? *IEEE Computer*, 38(7):85–87, 2005.

[19] G. Leen and D. Heffernan. A new time-triggered controller area network. *Microprocessors and Microsystems Journal*, 26(2):77–94, 2002.

[20] L. Mangeruca, M. Baleani, A. Ferrari, and A. Sangiovanni-Vincentelli. Semantics-preserving design of embedded control software from synchronous model. *IEEE Transactions on Software Engineering*, 33(8):497–509, 2007.

[21] Marco Mazzucchelli. Esperienze nell'uso di strumenti per l'analisi di sistemi in tempo reale. "Laurea triennale" degree thesis, October 2003. In Italian.

[22] P. Merlin and D. Farber. Recoverability of communication protocols – implications of a theoretical study. *IEEE Transactions on Communications*, pages 1036–1043, 1976.

[23] L. Petruccelli, M. Velardocchina, and A. Sorniotti. Electro-Hydraulic Braking System Modelling and Simulation. In *Proceedings of the SAE 21st Annual Brake Colloquium & Exhibition, October 2003, Hollywood, FL, USA.*, 2003. SAE Paper 2003-01-3336.

[24] Wolfgang Reisig. *Petri Nets: An Introduction.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1985.

[25] SAE Society of Automotive Engineers, Warrendale, PA. *Antilock Brake Review*, 1992. SAE Standard: J2246.

[26] S.M. Savaresi, M. Tanelli, and C. Cantoni. Mixed Slip-Deceleration Control in Automotive Braking Systems. *ASME Journal of Dynamic Systems, Measurement and Control*, 129(1):20–31, 2006.

[27] Specification and description language SDL. ITU-T Reccomendation Z.100.

[28] M. Tanelli. *Active Braking Control Systems Design for Road Vehicles.* PhD thesis, Politecnico di Milano, May 2007.

[29] M. Tanelli, G. Osorio, M. di Bernardo, S.M. Savaresi, and A. Astolfi. Limit cycles analysis in hybrid anti-lock braking systems. In *Proceedings of the 46th Conference on Decision and Control, CDC 2007, New Orleans, Louisiana, USA*, pages 3865–3870, 2007.

[30] Sergio Yovine. Kronos: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1(1–2):123–133, 1997.