On the performance evaluation of multi-guarded marked graphs with single-server semantics[*]

Jorge Júlvez[†] Jordi Cortadella
Universitat Politècnica de Catalunya
Barcelona, Spain

Michael Kishinevsky
Intel Corporation
Hillsboro, USA

Abstract

In discrete event systems, a given task can start executing when all the required input data are available. The required input data for a given task may change along the evolution of the system. A way of modeling this changing requirement is through multi-guarded tasks. This paper studies the performance evaluation of the class of marked graphs extended with multi-guarded transitions (or tasks). Although the throughput of such systems can be computed through Markov chain analysis, two alternative methods are proposed to avoid the state explosion problem. The first one obtains throughput bounds in polynomial time through linear programming. The second one yields a small subsystem that estimates the throughput of the whole system.

Keywords: Early evaluation, throughput bounds, Petri nets, marked graphs.

1 Introduction

The modeling of discrete event systems often relies on a producer/consumer pattern. According to this pattern, a given consumer action cannot start until all the required input items have been produced by some previous producer actions. For example, in a digital circuit an adder that computes the sum of two integer numbers cannot start operating until both numbers are available at the input of the adder. Nonetheless, there are actions that do not always require all its input data to be available to start operating. A typical component exhibiting this behavior is a multiplexer. A multiplexer is a device that selects one of many input channels and outputs the value of that channel. The values of the rest of the channels are useless. Hence, a multiplexer could start operating as soon as the value of the selected channel is available without waiting for the rest of the values. The availability of the selected channel is the precondition, or *guard*, required by the multiplexer to start working. Given that different input channels may be selected along time, the multiplexer guard is not constant but changes along time.

Consider a computer program that processes the data records of a given file stored in a hard disk. In order to speed up the program execution, some fields of the records are stored in memory. Let us assume that with probability γ the program just requires the fields stored in memory to process a record, and with probability $1-\gamma$ it requires

[*] This work has been partially supported by a gift from Intel Corp. and CICYT TIN 2004-07925.

[†] Supported by the Spanish Ministry of Education and Science (Juan de la Cierva fellowship) and by the European Communitys Seventh Framework Programme under project DISC (Grant Agreement n. INFSO-ICT-224498).

all the fields of the record. Once a data record is processed, the program requests a new record. Such request activates two actions in parallel: one searches the record in the memory and the other one in the hard disk. If the fields stored in memory are sufficient to continue processing, the data provided by hard disk will be discarded. The behavior of this system can be modeled by a Petri net, see the net $S1$ in Figure 1.

The transitions (bars) of the system represent actions that consume the tokens (black dots) stored in the input places (circles), and after completion, store new tokens in the output places (see [Mur89, Sil93] for a tutorial on Petri nets). The token in place p_1 describes the situation in which a record has just been processed and that a new record is to be requested. Once the request is issued, the token from p_1 is removed, a token is placed in p_2, and a token is placed in p_4: accessing memory and hard disk is being done in parallel. Getting record fields from memory just needs one task, *Memory*, after which memory fields are ready to be processed. Getting the whole record from hard disk needs three tasks: *Access*, *Search* and *Deliver*. With probability γ, memory fields are enough to continue processing, and therefore, the data provided by disk will be ignored. Hence, the *Process* action not only processes data, but acts as a multiplexor by selecting either the memory fields or the whole record. It will be assumed that all tasks last the same amount of time.

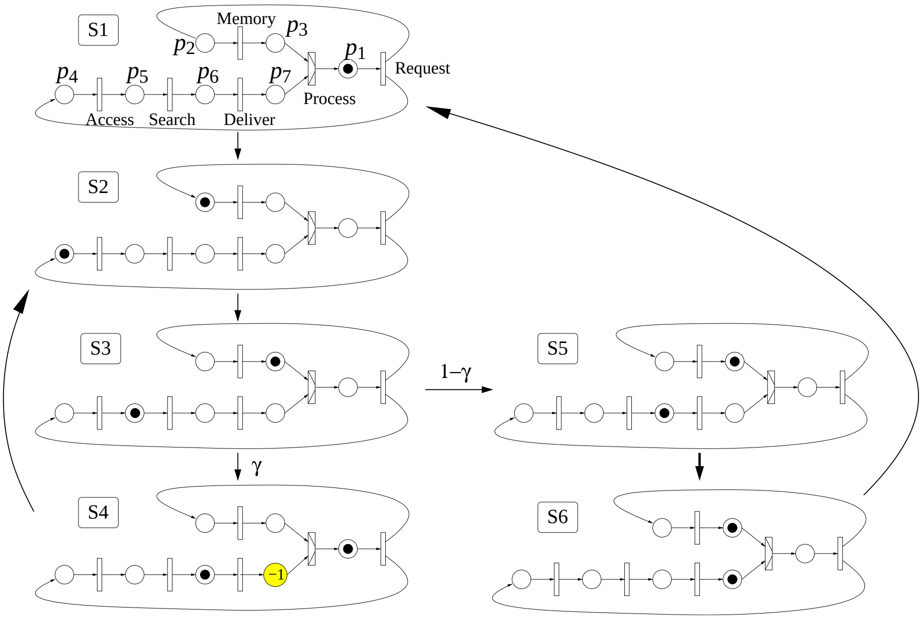

Figure 1: Reachable states of an early evaluated system that processes data records of a file stored in a hard disk..

The graph in Figure 1 depicts the set of reachable states. A directed arc connects a given state to a successor state. Arcs are labeled with the probability of being taken. If the probability of taking the arc is 1, the label is omitted. Let us focus on state S3 at which memory fields are already available but the hard disk has not provided yet all fields. With probability $1-\gamma$ it is necessary to wait for the hard disk to deliver the data, i.e., the path (S3, S5, S6) is taken. At S6 all data is available and can be processed. When moving from S6 to S1 the tokens in p_3 and p_7 are consumed (although only the one is p_7 is required) and a new token in p_1 is put meaning that a new request must be done. On the other hand, from S3 there is a probability γ of not having to wait for the disk, i.e., the arc from S3 to S4 is taken. In such case, the memory fields are processed (the token in p_3 is consumed) and a new record request is issued (a token in p_1 is put). Notice that in parallel to this processing, the disk is working in order to get the whole record which must not be processed. A simple way of discarding the token that will be delivered by the *Deliver* task is by putting a negative token in p_7 at S4. This way, the token that will produce the *Deliver* task at p_7 will vanish when meeting the negative token, state S2.

The main new feature of the model in Figure 1, with respect to conventional Petri nets, is that it is not strictly necessary to finish every previous task before starting the *Process* task. This fact provides more flexibility to the modeling of discrete event systems and usually increases the system performance (if *Process* is executed only after *Memory* and *Deliver*, the system state will loop on S1, S2, S3, S5 and S6, ignoring the short cut provided by S4).

A natural consequence of performing a task, like *Process*, without having finished some previous tasks, *Deliver*, is that negative tokens appear after the completion of the task, state S4. Negative tokens are necessary to be consistent with the producer/consumer pattern, in which a token is consumed from every input place. Moreover, they have the responsibility of getting rid of non required tokens, like the token in place p_7 at S4.

Negative tokens have been considered in several works for different purposes. In [MY90], Petri nets with negative tokens are used to study automated reasoning programs. Negative event graphs are introduced in [LP05] to analyze time window constraints. In [OM05], a negative token represents a temporary buffer overflow in a manufacturing system. Differential places of differential Petri nets [DK98] are allowed to store real quantities (positive or negative) of tokens in order to model hybrid systems.

1.1 Framework and Contribution

This paper focuses on the class of Multi-Guarded Marked Graphs (GMG). The underlying structure of a GMG is that of a conventional marked graph (MG), a Petri net subclass that offers an interesting trade-off between modeling power and analysis capability [Mur89]. In contrast to conventional MGs, a transition of a GMG can have several guards, i.e., it is multi-guarded, which are randomly selected. With respect to the preliminary results obtained in [JCK06], the present paper discusses the boundedness property of GMGs, analyzes the resulting throughput bounds, and proposes a heuristics to improve the obtained throughput bounds.

The main goals of the paper are the following:

- Provide an efficient method to compute steady state throughput bounds.
- Describe a heuristics to obtain a small subsystem that estimates the steady state throughput.

Before focusing on these goals, essential system properties as liveness and boundedness of GMGs are studied to ensure the well-behavior of the system. The approach suggested in this paper to compute steady state throughput bounds is based on Linear Programming (LP). Such an approach allows us to naturally manage structural and steady state constraints of GMG.

LP techniques have been intensively used to compute throughput bounds in the framework of queueing networks and Petri nets. In the queueing network setting, some representative works in the literature are the following: [KK94] establishes a set of linear equality constraints on the mean values of some variables and conservation laws allow to bound the performance of the system. A similar approach is taken in [BC99], where the region of achievable performance vectors is characterized in order to formulate a linear programming problem to obtain bounds. In [YCT84], performance bounds for bulk arrival queues are computed by making use of the bounds for single arrival queues.

In the Petri nets framework (notice that with respect to queueing networks, Petri nets include a synchronization primitive), two main approaches that make use of LP techniques can be distinguished. In [Liu98], the uniformization technique is used to derive linear equalities and then LP is applied to obtain throughput bounds. In [CAC$^+$95, CS92], the use of operational laws allows one to obtain linear constraints that are included in a LP problem whose solution is an upper bound for the throughput.

Similarly to [KK94, BC99], the approach presented here establishes some linear constraints related to conservation laws and mean values of throughput and average markings [CS92] to design a Linear Programming Problem (LPP). The solution of the LPP is an upper bound for the throughput in the steady state. In contrast to previous approaches, in order to handle multi-guarded transitions, a new expression relating the throughput and average marking of multi-guarded transitions has been developed and included in the mentioned LPP.

With respect to the second goal, the heuristics is based on obtaining a small subsystem whose throughput is similar to the throughput of the whole system. The subsystem is built from the solution obtained by a LPP and an iterative process. As far as we know, no similar approach has been proposed before in the literature.

The rest of the paper is organized as follows: Section 2 defines the class of Multi-Guarded Marked Graphs (GMG) and presents some basic properties. Section 3 shows how to compute the throughput of a GMG by using Markov chains. Section 4 presents a linear programming problem to compute upper bounds for the throughput. A procedure to obtain a small bottleneck subsystem of a GMG is explained in section 5. Section 6 shows the obtained experimental results. Conclusions are drawn in section 7.

2 Multi-Guarded Marked Graphs

2.1 Definition and Semantics

In the following, the reader is assumed to be familiar with Petri nets (PN s) (see [Mur89, Sil93] for a tutorial on Petri nets).

Definition 1 (GMG) *A* Multi-Guarded Marked Graph *(GMG) is a tuple $\mathcal{N} = \langle P, T, Pre, Post, G, \mathbf{m}_0 \rangle$ where:*

- *P is a finite set of places, and T is a finite set of transitions. The* preset *and* postset *of a node $x \in P \cup T$ are denoted as ${}^\bullet x$ and $x^\bullet$. The following condition holds: $\forall p \in P, |{}^\bullet p| = |p^\bullet| = 1$.*
- *$Pre : P \times T \rightarrow \mathbb{N} \cup \{0\}$ and $Post : P \times T \rightarrow \mathbb{N} \cup \{0\}$ are the* pre- *and* post- *incidence functions that specify the arc weights. The incidence matrix of the net is $\mathbb{C} = Post - Pre$.*
- *$G : T \rightarrow 2^{2^P}$ assigns a set of guards to every transition. The following conditions must be satisfied: a) $\forall\, g \in G(t)$ it holds $g \subseteq {}^\bullet t$; b) $\underset{g \in G(t)}{\cup} g = {}^\bullet t$.*
- *$\mathbf{m}_0 : P \rightarrow \mathbb{N} \cup \{0\}$ is the initial marking.*

A conventional Marked Graph (MG) is simply a GMG in which $G(t) = \{\{^\bullet t\}\}$ for every $t \in T$. In a GMG, the transitions can also satisfy the condition $G(t) = \{\{^\bullet t\}\}$. Such transitions will be called *simple* transitions, the rest of transitions will be called multi-guarded transitions. Simple transitions will be represented graphically as empty rectangles; multi-guarded transitions will be represented as rectangles with oblique lines inside (see Figure 2 for a multi-guarded transition).

Definition 2 (Firing semantics) *The dynamic behavior of a GMG system is determined by its firing rules. The execution of a transition t can be described as follows:*

- Guard selection. *A guard $g(t) \in G(t)$ for the next firing is selected nondeterministically. The guard selection is trivial for simple transitions, since they only have one guard. For multi-guarded transitions any guard in $G(t)$ can be selected. The selected guard of a transition t is* persistent, *i.e., it does not change until t fires.*
- Enabling. *If the guard $g(t) \in G(t)$ has been selected for the next firing of t, then the transition t becomes enabled when every place $p \in g(t)$ is positively marked, i.e., $\mathbf{m}(p) > 0$.*
- Firing. *A transition t enabled at marking $\mathbf{m}$ can fire leading to marking $\mathbf{m}'$ such that $\mathbf{m}' = \mathbf{m} + \mathbb{C}(P,t)$ where $\mathbb{C}(P,t)$ is the column of $\mathbb{C}$ corresponding to t.*

Notice that GMGs can reach markings with negative values. If $\mathbf{m}(p) \geq 0$ one says that place p has $\mathbf{m}(p)$ positive tokens. Otherwise, place p has $|\mathbf{m}(p)|$ negative tokens. Negative tokens account for the data that must be discarded when arriving at the input of the transition.

As in conventional MGs the state equation $\mathbf{m} = \mathbf{m}_0 + \mathbb{C} \cdot \sigma$ provides a necessary condition for the reachability of $\mathbf{m}$, where σ is the firing count vector of the transitions and $\mathbf{m}$ can contain negative elements. Then, vectors $\mathbf{y} \geq 0$, $\mathbf{y} \cdot \mathbb{C} = 0$ ($\mathbf{x} \geq 0$, $\mathbb{C} \cdot \mathbf{x} = 0$) represent P-semiflows or conservative components (T-semiflows or consistent components). A semiflow $\mathbf{v}$ is said to be *minimal* when its support, $\|\mathbf{v}\|$, is not a proper superset of the support of any other, and the greatest common divisor of its elements is one. As in conventional MGs [Mur89], every minimal P-semiflow of a GMG corresponds to a simple cycle, i.e., a cycle with no repeated vertices except for the start and end vertex.

Example 1 *Let us assume that the transition in Figure 2 has two guards $G(t) = \{\{p_a, p_b\}, \{p_a, p_c\}\}$, and that initially the places p_a and p_c have a token.*

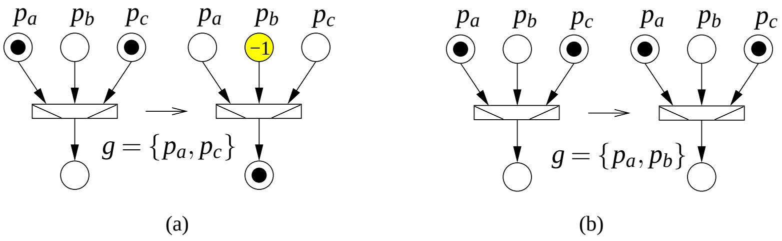

Figure 2: Multi-guarded transition: (a) early firing with guard $\{p_a, p_c\}$; (b) guard $\{p_a, p_b\}$ is selected, the transition will not fire until place p_b contains a token.

If the guard $\{p_a, p_c\}$ is selected (Figure 2(a)), the transition is enabled (p_a and p_c are marked) and can fire. The firing removes the tokens from p_a and p_c, puts one *positive token in the output place, and produces a negative token in p_b. If the guard $\{p_a, p_b\}$ is selected (Figure 2(b)), the transition is not enabled, and will not become enabled until a token is put in place p_b.*

The persistence of the guards is an accurate abstraction of the conditions to model early evaluations. For instance, once a multiplexer selects an input channel, it must wait for the selected channel to provide data before yielding its output and selecting another channel for the next firing.

In order to allow the system exhibit a cyclic and bounded behavior, we assume that the net is strongly connected. The subclass of GMG is sufficient for modeling a wide variety of systems, e.g., queueing networks, parallel processing systems, resource allocation schemes, etc. It also satisfies some properties that simplify their analysis.

2.2 Properties of GMGs

Two basic properties of GMGs are considered in this subsection: boundedness and liveness.

2.2.1 Boundedness

Definition 3 (Boundedness)

- A place, p, of a GMG is *upper-bounded* if there exists $u \in \mathbb{N}$ such that $\mathbf{m}(p) \leq u$ for every reachable marking $\mathbf{m}$. Place p can also be said *u-upperbounded*.
- A place, p, of a GMG is *lower-bounded* if there exists $l \in \mathbb{N}$ such that $\mathbf{m}(p) \geq l$ for every reachable marking $\mathbf{m}$. Place p can also be said *l-lowerbounded*.
- A place is *bounded* if it is upper-bounded and lower-bounded.

These definitions are trivially extended for nets: A net is said to be *upper-bounded* (*lower-bounded*, *bounded*) if every place is *upper-bounded* (*lower-bounded*, *bounded*). Since non-bounded systems cannot be implemented, boundedness is often a required system property. A GMG can be upper-bounded by adding complementary places in the same way it is done for conventional Petri nets. Moreover, the addition of places can also be used to lower-bound a GMG.

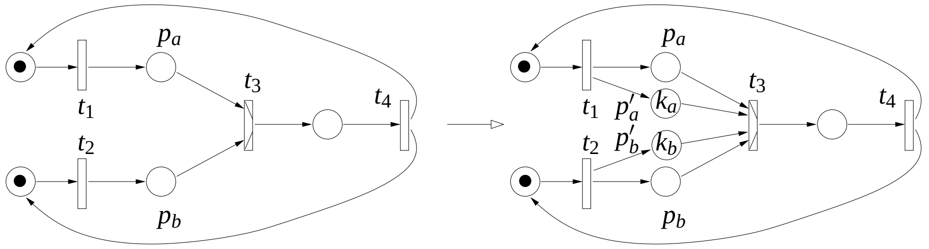

Figure 3: Transformation of a GMG with $G(t_3) = \{\{p_a\}, \{p'_b\}\}$ to lower-bound p_a and p_b. In the transformed net $G(t_3) = \{\{p_a, p'_a, p'_b\}, \{p_b, p'_a, p'_b\}\}$ and for every reachable marking it holds that: $\mathbf{m}(p_a) \geq -k_a$, $\mathbf{m}(p_b) \geq -k_b$.

Assume that the guards of t_3 in Figure 3(left) are $G(t_3) = \{\{p_a\}, \{p_b\}\}$. If the guard $\{p_a\}$ is selected indefinitely and t_2 is not fired, i.e., only transitions t_1, t_3 and t_4 are fired, then the marking of p_b tends to minus infinity. This can be avoided by adding places p'_a and p'_b and redefining the guards of t_3 as $G(t_3) = \{\{p_a, p'_a, p'_b\}, \{p_b, p'_a, p'_b\}\}$. This transformation still allows some flexibility, i.e., negative markings in p_a and p_b, but for every reachable marking it holds that: $\mathbf{m}(p_a) \geq -k_a$, $\mathbf{m}(p_b) \geq -k_b$.

2.2.2 Liveness

Definition 4 (Liveness) *A GMG is* live *if for every transition t and every reachable marking $\mathbf{m}$, there exists a marking $\mathbf{m}'$ reachable from $\mathbf{m}$ such that t is enabled at $\mathbf{m}'$.*

A conventional MG is live iff every cycle has a positive number of tokens. The same necessary and sufficient conditions applies to GMGs [Jul09].

Theorem 1 *A GMG is deadlock-free iff the sum of tokens in each cycle is positive.*

Thus, liveness of a GMG can be decided in polynomial time by checking that every cycle has a positive number of tokens [Sil93].

2.3 Timed Multi-Guarded Marked Graphs

In Petri nets, transitions usually represent the events or actions of the system. To model latencies or delays of the system events, a nonnegative real number $\delta(t)$ is associated with every transition t of the GMG. We will assume that delays are deterministic, i.e., a transition t that becomes enabled at time τ will fire at time $\tau + \delta(t)$. More general delay models are possible for GMGs, most notably it is possible to have probabilistic distributions of delays associated with every transition to model variable latency units or delay variations. This however goes beyond the scope of this paper.

The performance evaluation of the model requires some a priori information about probabilities for selecting guards. Such probabilities are usually known prior to system design or can be obtained after simulation by monitoring events and counting them. It is assumed here that selection of guards for different transitions are independent events and hence probabilities of the guards of different transitions are uncorrelated.

Definition 5 (TGMG) *A* Timed Multi-Guarded Marked Graph *(TGMG) is a tuple $N = \langle P, T, F, G, \mathbf{m}_0, \delta, \alpha \rangle$ where:*

- $\langle P, T, F, G, \mathbf{m}_0 \rangle$ *is a GMG.*
- $\delta : T \rightarrow \mathbb{R}^+ \cup \{0\}$ *assigns a nonnegative real delay to every transition.*
- $\alpha : G \rightarrow \mathbb{R}^+$ *assigns a strictly positive probability to each guard such that for every multi-guarded transition t: $\sum\limits_{g \in G(t)} \alpha(g) = 1$.*

It will be assumed that there exists at least one transition t such that $\delta(t) > 0$. For the firing of the transitions, the *single-server semantics* will be adopted. According to this semantics, no multiple-instances of the same transition can fire simultaneously. The single-server semantics is an abstraction for those systems that communicate through channels using FIFOs. The time evolution of a TGMG is derived from the previously described semantics of the GMG:

- After the firing of a multi-guarded transition t, a new guard for t is selected.
- Guard selection for every transition is still non-deterministic, but respects probabilities in the infinite executions.
- Firing of transition t takes $\delta(t)$ time units, from the time it becomes enabled until the firing is completed.

Definition 6 (Steady state throughput) *The steady state throughput of transition t, $Th(t)$, of a TGMG is defined as:*

$$Th(t) = \lim_{\tau\to\infty} \frac{\sigma(t,\tau)}{\tau} \tag{1}$$

where τ represents the time and $\sigma(t,\tau)$ is the firing count vector of t at time τ.

Given that the unique minimal T-semiflow of strongly connected MGs is a vector of ones, the following proposition holds:

Proposition 2 *Let $\mathcal{N}$ be a TGMG. For every couple of transitions $t_i, t_j \in T$ it holds that: $Th(t_i) = Th(t_j)$*

Definition 7 (Average marking) *The average marking of place p, $\overline{\mathbf{m}}(p)$, of a TGMG is defined as:*

$$\overline{\mathbf{m}}(p) = \lim_{\tau\to\infty} \frac{1}{\tau}\int_0^\tau \mathbf{m}(\xi)d\xi \tag{2}$$

where $\mathbf{m}(p,\xi)$ is the marking of p at time ξ.

Both limits, (1) for the steady state throughput and (2) for the average marking, exist for every TGMG [JCK06].

2.3.1 Reduction to singleton form

This subsection presents a technique to transform the guards of multi-guarded transitions into singleton guards. This transformation will very useful for the computation of upper bounds for the throughput.

Figure 4 shows a fragment of a TGMG with a multi-guarded transition t whose guards $G(t) = \{\{p_a, p_b\}, \{p_b, p_c\}\}$ are not singletons. An equivalent TGMG has a multi-guarded transition with singleton guards. Two new simple transitions, t_1 and t_2, with zero delay are introduced. They combine together the guards $\{p_a, p_b\}$ and $\{p_b, p_c\}$ of the original transition t. Note that place p_b is duplicated. This technique transforms any TGMG to a singleton form without changing its throughput. In general, for a given multi-guarded transition t with non-singleton guards, this technique creates $|g(t)|$ new transitions and $|g(t)| + \sum_{p\in{}^\bullet t} |\{g(t) \in G(t) \mid p \in g(t)\}| - |{}^\bullet t|$ new places. Notice that the number of new elements, i.e., places and transitions, is bounded by a polynomial in $|T|$, $|P|$, $|G|$.

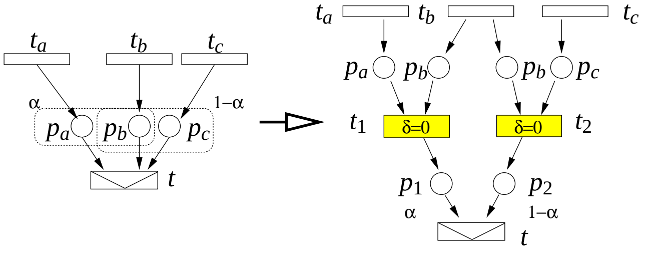

Figure 4: Reduction to singleton form.

In the rest of the paper, it is assumed that all non-simple transitions are transformed to the singleton form. Moreover, it is assume that every TGMG is bounded and live. These assumptions do not reduce in practice the generality of the results, since both properties are required for an adequate behavior of real systems.

3 TGMG as a semi-Markov process

This section shows how the exact throughput of a TGMG can be computed. The evolution of the state of a TGMG can be expressed as a graph in which each node represents a reachable state, and each arc represents a transition between two states (see Figure 1). A real number is associated to each arc indicating the probability of taking such arc. This way, the behavior of a TGMG is described as a semi-Markov [Wol89] process. Once the average sojourn time of each state is computed, the throughput of a transition, t, can be obtained by dividing the probability of being enabled, $prob(enab(t))$, by its delay, $\delta(t)$ [CAC+95]:

$$Th(t) = \frac{prob(enab(t))}{\delta(t)} \tag{3}$$

In the following example we will describe the state of a TGMG at a given instant as a tuple $\{\mathbf{m}, g, cks\}$, where $\mathbf{m}$ is the marking of the net, $g : T \rightarrow G$ is the set of guards selected for each multi-guarded transition, and cks is a function, $cks : T \rightarrow \mathbb{R}^+$ where $cks(t)$ is the remaining time to fire transition t (if t is not enabled at $\mathbf{m}$ then $cks(t) = \infty$).

Example 2 *Let us consider the system in Figure 5(a) with initial marking* $\mathbf{m}_0 = (1;0;1;0;1)$*, delays* $\delta = (3;1;1;3)$*, and* $\alpha(\{p_1\}) = \gamma$*,* $\alpha(\{p_3\}) = 1-\gamma$*. Assume that initially the guard selected for* t_1 *is* $\{p_1\}$*. At that initial state* t_1 *and* t_4 *are enabled, and will fire after 3 time units. Then, the initial state of the associated semi-Markov process is* $S1 = \{(1;0;1;0;1), \{p_1\}, (3;\infty;\infty;3)\}$*. After three time units* t_1 *and* t_4 *fire reaching marking* $(0;1;1;1;0)$*. Since the multi-guarded transition* t_1 *has fired, a new guard, either* $\{p_1\}$ *or* $\{p_3\}$*, must be selected. The new state of the systems depends on the selected guard.*

The set of reachable states for the system in Figure 5(a) is: $S1 = \{(1;\ 0;\ 1;\ 0;\ 1),\ \{p_1\},\ (3;\infty;\infty;3)\}$, $S2 = \{(0;\ 1;\ 1;\ 1;\ 0),\ \{p_1\},\ (\infty;1;1;\infty)\}$, $S3 = \{(0;\ 1;\ 1;\ 1;\ 0),\ \{p_3\},\ (3;1;1;\infty)\}$, $S4 = \{(1;\ 0;\ 1;\ 0;\ 1),\ \{p_3\},\ (2;\infty;\infty;3)\}$, $S5 = \{(0;\ 1;\ 0;\ 1;\ 1),\ \{p_1\},\ (\infty;1;1;1)\}$, $S6 = \{(0;\ 1;\ 0;\ 1;\ 1),\ \{p_3\},\ (\infty;1;1;1)\}$, $S7 = \{(1;\ 0;\ 1;\ 0;\ 1),\ \{p_3\},\ (3;\infty;\infty;3)\}$. *The graph in Figure 5(b) shows the transition probability among states.*

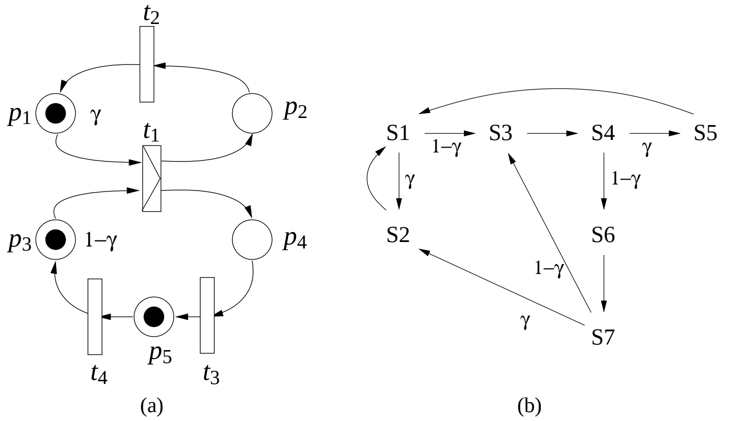

Figure 5: A TGMG (a) and its associated Markov process (b).

From the graph in Figure 5(b), one can obtain the transition probability matrix P*, where* $P(i,j)$ *equals the probability of taking the arc from* S_i *to* S_j*. As for discrete* *time Markov chains, one can solve the linear system:* $\Pi = \Pi \cdot P$*;* $\Pi \cdot \mathbf{1} = 1$*, to obtain the stationary distribution,* Π*, of the corresponding embedded Markov chain. Let* $\rho(s)$ *denote the sojourn time in state s, e.g.,* $\rho(S1) = 3$*. If the embedded Markov chain of the semi-Markov process is irreducible and recurrent then the fraction of time,* $\psi(s)$*, that the process spends in s is given by [Wol89]:*

$$\psi(s) = \frac{\rho(s) \cdot \Pi(s)}{\Sigma_{s_i \in \Theta} \rho(s_i) \cdot \Pi(s_i)}$$

where Θ *is the set of all states. For the system in Figure 5(a), we have:*

$$\psi = \frac{1}{7 - 3 \cdot \gamma}(6 \cdot \gamma - 3 \cdot \gamma^2;\ \gamma;\ 1 - \gamma;\ 2 - 2 \cdot \gamma;\ \gamma - \gamma^2;\ 1 - 2 \cdot \gamma + \gamma^2;\ 3 - 6 \cdot \gamma + 3 \cdot \gamma^2)$$

The throughput of a transition t is the probability of being enabled in the steady state divided by its delay $\delta(t)$*. Let us compute the throughput of* t_4*. Since* t_4 *is enabled at* $S1$*,* $S4$*,* $S5$*,* $S6$*, and* $S7$*, the probability of being enabled is* $prob(enab(t_4)) = \psi(S1) + \psi(S4) + \psi(S5) + \psi(S6) + \psi(S7)$*. According to the enabling operational law [CAC+95](see Equation* (3)*) we obtain:*

$$Th(t_4) = \frac{prob(enab(t_4))}{\delta(t_4)} = \frac{1}{3} \cdot \frac{6 - 3 \cdot \gamma}{7 - 3 \cdot \gamma} = \frac{2 - \gamma}{7 - 3 \cdot \gamma}$$

which by Proposition 2 is the throughput of all transitions.

4 Throughput bounds

This section shows how to formulate a linear programming problem to compute an upper bound for the steady state throughput of a TGMG.

4.1 Linear relationships

As stated in Section 2, it is assumed that every transition either is simple, i.e., it has only one guard, or has singleton guards, i.e., each guard contains only one place. We will discuss two kind of linear relationships: one for multi-guarded transitions, one for simple transitions.

Let t be a multi-guarded transition of a TGMG, $\delta(t)$ its delay, and $prob(enab(t))$ the probability of t to be enabled during the steady state. In other words, $prob(enab(t))$ is the time ratio during which t is enabled. Since transitions have deterministic delays and operate under the single server semantics, the enabling operational law [CAC+95] for t is:

$$\delta(t) \cdot Th(t) = prob(enab(t)) \qquad \text{for any } t \in T \tag{4}$$

After a number of algebraic manipulations, the value $prob(enab(t))$ can be expressed in terms of the marking of the input places of t. In particular, a useful expression is given by Theorem 3, which provides a linear relationship between the throughput of a multi-guarded transition and the average marking of its input places.

Theorem 3 *Let t be a transition with singleton guards, then:*

$$\delta(t) \cdot Th(t) = \sum_{p \in {}^\bullet t} \alpha(\{p\}) \cdot \left(\overline{\mathbf{m}}(p) - \sum_{i=2}^{\infty} (i-1) \cdot prob(\mathbf{m}(p) = i) \right)$$

Proof: See Appendix. □

Corollary 4 *Let t be a transition with singleton guards, then:*

$$\delta(t) \cdot Th(t) \leq \sum_{p \in {}^{\bullet}t} \alpha(\{p\}) \cdot \overline{\mathbf{m}}(p) \tag{5}$$

Moreover, if the input places of t are 1-upperbounded then:

$$\delta(t) \cdot Th(t) = \sum_{p \in {}^{\bullet}t} \alpha(\{p\}) \cdot \overline{\mathbf{m}}(p) \tag{6}$$

Notice that neither Theorem 3 nor Corollary 4 require the input places of t to be lowerbounded. Moreover, the conditions required to apply these results are purely local: It does not matter whether the rest of the places are bounded or not.

On the other hand, each pair $\{p,t\}$ where $p^{\bullet} = t$ is a simple transition can be seen as a simple queuing system for which Little's formula [Lit61] can be directly applied [CS92]:

$$\overline{R}(p) \cdot Th(t) = \overline{\mathbf{m}}(p) \tag{7}$$

where $\overline{R}(p)$ is the average residence time at place p, i.e., the average time spent by a token in p. The average residence time is the sum of the average waiting time due to a possible synchronization delay and the average service time which in our case is $\delta(t)$. Therefore the service time $\delta(t)$ is a lower bound for the average residence time. This leads to the inequality:

$$\delta(t) \cdot Th(t) \leq \overline{\mathbf{m}}(p) \qquad \text{for every input place } p \text{ of a simple transition} \tag{8}$$

4.2 Linear programming and throughput bounds

One can combine the above constraints on the throughput and on the average marking, (5) and (8), to build a Linear Programming Problem (LPP) that maximizes a parameter ϕ, corresponding to the TGMG throughput [JCK06]. One scalar variable suffices since the throughput of all transitions is the same (Proposition 2). Let T_1 be the set of multi-guarded transitions, and T_2 the set of simple transitions. Transitions with only one input place can be included either in T_1 or in T_2. The resulting LPP can be expressed as:

$$\begin{aligned}
&\textit{Maximize} \;\; \phi : \\
&\qquad \delta(t) \cdot \phi \leq \sum_{p \in {}^{\bullet}t} \alpha(\{p\}) \cdot \widehat{\mathbf{m}}(p) \quad \text{for every } t \in T_1 \\
&\qquad \delta(t) \cdot \phi \leq \widehat{\mathbf{m}}(p) \quad \text{for every } p \in {}^{\bullet}T_2 \\
&\qquad \widehat{\mathbf{m}} = \mathbf{m}_0 + \mathbb{C} \cdot \sigma \\
&\qquad \phi \geq 0, \;\; \sigma \geq 0
\end{aligned} \tag{9}$$

where σ represents the firing count vector that drives the system from the initial marking, $\mathbf{m}_0$, to the estimated average marking $\widehat{\mathbf{m}}$.

We will now perform some manipulations on (9) in order to obtain a more intuitive LPP. Let ${}^{\bullet}t_j = \{p_1, \ldots, p_n\}$ for a given $t_j \in T_1$. The term $\sum_{p \in {}^{\bullet}t} \alpha(\{p\}) \cdot \widehat{\mathbf{m}}(p)$ in (9) can be substituted by a single variable $\widehat{\mathbf{m}}_{tj}$ such that $\widehat{\mathbf{m}}_{tj} = \sum_{p \in {}^{\bullet}t} \alpha(\{p\}) \cdot \widehat{\mathbf{m}}(p)$. This is equivalent to substituting the rows of $\mathbf{m}_0$, *Pre* and *Post* corresponding to $\{p_1, \ldots, p_n\}$

by a single row that is a linear combination of them (in such a linear combination the weight of the row corresponding to p_i is $\alpha(\{p_i\})$). Let $\mathbf{m}_{0r}$, Pre_r, $Post_r$ and $\mathbb{C}_r = Post_r - Pre_r$ be the arrays obtained after performing such a substitution on the set of input places of every multi-guarded transition.

At net level such a substitution can be seen as a net transformation that produces "real weighted arcs": Figure 6 shows the graphical interpretation of the transformation. Notice that the obtained net is not a MG any more. We will just focus on the algebraic consequences of the transformation.

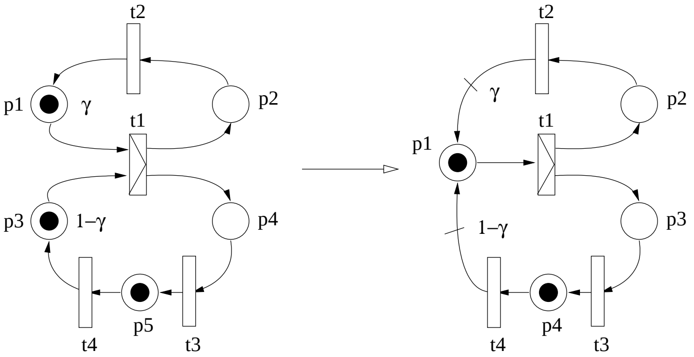

Figure 6: Transformation rule for the system in Figure 5. The multi-guarded transition t_1 has two guards with probabilities γ and $1-\gamma$. The weight of the arcs (t_2, p_1) and (t_4, p_1) of the transformed net are: γ and $1-\gamma$.

The LPP (9) can be expressed as:

$$max\{\ \phi \mid \phi \cdot D \leq \mathbf{m}_{0r} + \mathbb{C}_r \cdot \sigma,\ \phi \geq 0,\ \sigma \geq 0\ \} \tag{10}$$

where $D(p) = \delta(p^\bullet)$. Let us define $\rho = \dfrac{1}{\phi}$, and $\sigma' = \dfrac{\sigma}{\phi}$, then (10) becomes:

$$min\{\ \rho \mid D \leq \rho \cdot \mathbf{m}_{0r} + \mathbb{C}_r \cdot \sigma',\ \rho \geq 0,\ \sigma' \geq 0\ \} \tag{11}$$

The dual of (11) is:

$$max\{\ \mathbf{y} \cdot D \mid \mathbf{y} \cdot \mathbb{C}_r \leq 0,\ \mathbf{y} \cdot \mathbf{m}_{0r} \leq 1,\ \mathbf{y} \geq 0\ \} \tag{12}$$

One theorem of the alternatives [Mur83] states that $\exists\ \mathbf{x} > 0$ such that $\mathbb{C}_r \cdot \mathbf{x} \geq 0$ iff $\forall\ \mathbf{y} \geq 0$ such that $\mathbf{y} \cdot \mathbb{C}_r \leq 0$ then $\mathbf{y} \cdot \mathbb{C}_r = 0$. Since we are dealing with consistent systems, there exists $\mathbf{x} > 0$ such that $\mathbb{C}_r \cdot \mathbf{x} \geq 0$, therefore $\mathbf{y} \cdot \mathbb{C}_r \leq 0$ can be replaced by $\mathbf{y} \cdot \mathbb{C}_r = 0$. Furthermore, since the objective function $\mathbf{y} \cdot D$ is maximized, the solution must satisfy $\mathbf{y} \cdot \mathbf{m}_{0_r} = 1$ (otherwise a "more optimal" solution is possible with $\beta \cdot \mathbf{y}$, $\beta = 1/(\mathbf{y} \cdot \mathbf{m}_{0r})$).

Theorem 5 *Let $\mathcal{N}$ be a TGMG. Let $\mathbb{C}_r$, $D^{(1)}$ and $\mathbf{m}_{0r}$ be the matrices obtained after performing the above transformation rule. Let Γ be the solution of:*

$$\Gamma = max\{\ \mathbf{y} \cdot D \mid \mathbf{y} \cdot \mathbb{C}_r = 0,\ \mathbf{y} \cdot \mathbf{m}_{0r} = 1,\ \mathbf{y} \geq 0\ \} \tag{13}$$

Then, $Th(t) \leq \dfrac{1}{\Gamma}$ for any $t \in T$.

Proof: Let us first show that the LPP is feasible and bounded.

(Feasible) In [Sil93] it is shown that every place of a strongly connected MG belongs to a P-semiflow. The described net transformation substitutes sets of rows of $\mathbb{C}$

by a linear combination of them, then after the transformation every place still belongs to a P-semiflow. Let us take any P-semiflow, $\mathbf{y}$, of the transformed net and normalize it to satisfy $\mathbf{y} \cdot \mathbf{m}_{0r} = 1$. Then, $\mathbf{y}$ satisfies all constraints of the LPP. Hence, the LPP is feasible.

(Bounded) Given that we are considering live GMGs, every cycle, i.e., P-semiflow, contains at least one token (Theorem 1). Then, the normalization, $\mathbf{y} \cdot \mathbf{m}_{0r} = 1$, prevents any P-semiflow $\mathbf{y}$ from tending to infinity.

Given that at least one transition has strictly positive delay (see assumption below Definition 5), the value of Γ will be always strictly positive, that is $1/\Gamma$ is never infinity. The constraints in the LPP have been obtained from expressions that upper bound the throughput for each transition. Hence, the solution of the LPP produces an upper bound for the throughput. □

This way, the computation of an upper bound for the throughput can be computed in polynomial time by solving the LPP (13), which actually represents a search for the bottleneck P-semiflow of the transformed net.

Although, in general, the solution of (13) gives an upper bound for the throughput, there are two particular cases for which it gives the exact throughput.

Corollary 6 *Let $\mathcal{N}$ be a TGMG such that for every $t \in T$, t is simple and $|^{\bullet}t| = 1$ or t is multi-guarded and has singleton guards. If $\mathcal{N}$ is 1-upperbounded then $1/\Gamma$, where Γ is the solution of* (13)*, is the exact throughput of the TGMG.*

Proof: Assume that for every $t \in T$, t is simple and $|^{\bullet}t| = 1$ or t is multi-guarded and has singleton guards. Then the transformed net (see above transformation rule) has no synchronizations, i.e., for every $t \in T$ it holds that $|^{\bullet}t| = 1$. Then, the net has only one P-semiflow. In other words, there is a unique vector $\mathbf{y}$ that satisfies the constraints in the LPP. Given that those constraints must be satisfied by the system, the solution associated to such a vector must be the real throughput of the system. □

Corollary 7 *Let $\mathcal{N}$ be a 1-upperbounded TGMG with simple transitions only. Then, $1/\Gamma$, where Γ is the solution of* (13)*, is the exact throughput of TGMG.*

Proof: This is a well known result in marked graph theory [Ram74, RH80]. □

Example 3 *Let us consider again the 1-bounded TGMG in Figure 5 where the delays are $\delta = (3;1;1;3)$. The values of $\mathbf{m}_{0r}$ and $\mathbb{C}_r$ for the transformed net (see Figure 6) are: $\mathbf{m}_{0r} = (1;\ 0;\ 0;\ 1)$ and $\mathbb{C}_r = (\text{-}1\ \ \gamma\ \ 0\ \ 1\text{-}\gamma;\ 1\ \ \text{-}1\ \ 0\ \ 0;\ 1\ \ 0\ \ \text{-}1\ \ 0;\ 0\ \ 0\ \ 1\ \ \text{-}1)$. Then, the value of $1/\Gamma$ of the solution of the LPP* (13) *associated to this TGMG is: $\dfrac{1}{\Gamma} = \dfrac{2-\gamma}{7-3\cdot\gamma}$, which corresponds exactly to the solution obtained with the Markov chain analysis. The solution obtained by the LPP is necessarily the exact throughput since the condition of Corollary 6 is fulfilled.*

5 System bottlenecks

It has been shown that linear programming can be used to efficiently bound the steady state throughput of a TGMG. This section presents a method to estimate more accurately such throughput. The main idea of the method is to obtain a small subsystem, i.e., a system bottleneck, from a given TGMG that approximates the throughput of the TGMG. The method takes a subnet obtained from the solution of LPP (9) as the initial system bottleneck. Then an iterative process enlarges such initial bottleneck. The iterative process makes use of the concepts of basic behavior and tight marking. Subsection 5.1 introduces the concepts of basic behavior and tight marking. The iterative method to compute a system bottleneck is discussed in Subsection 5.2.

5.1 Basic behaviors and tight marking

Informally, a basic behavior of a TGMG is the behavior exhibited by the system when the probabilities of all the guards are set either to 0 or to 1 (in this section the probabilities of the guards are allowed to be 0). For instance, the system in Figure 5 has two basic behaviors: when $\gamma = 1$ the system behavior is equivalent to the loop $l_1 = \{t_1, t_2\}$, when $\gamma = 0$ it is equivalent to the loop $l_2 = \{t_1, t_3, t_4\}$. For our purposes, it is useful to consider basic behaviors of subnets. In these basic behaviors only the multi-guarded transitions in the subnet have probabilities 0 or 1 in their guards, the rest of multi-guarded transitions are taken as simple transitions.

Definition 8 ($\Omega(N_s)$) *Let N_s be a subnet of a given TGMG N. $\Omega(N_s)$ is the set of probability functions that assign to each guard in N_s either 0 or 1.*

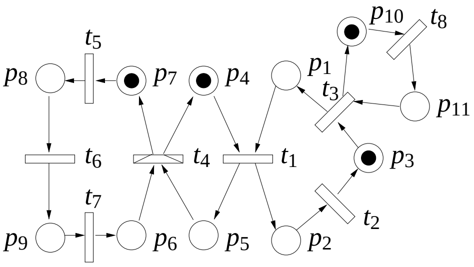

Figure 7: A TGMG with two basic behaviors.

In other words, each $\omega \in \Omega(N_s)$ represents a basic behavior of N_s. Consider the system in Figure 7 with all delays equal to 1. Let us assume that N_s is defined by the places $\{p_4, p_5, p_6, p_7, p_8, p_9\}$. Then, $\Omega(N_s)$ contains two elements, ω_1 and ω_2, such that $\omega_1(\{p_5\}) = 0$, $\omega_1(\{p_6\}) = 1$, $\omega_2(\{p_5\}) = 1$ and $\omega_2(\{p_6\}) = 0$.

Definition 9 ($N(N_s, \omega)$) *Let N_s be a subnet of a given TGMG N. For a given $\omega \in \Omega(N_s)$, $N(N_s, \omega)$ is the TGMG N in which $\alpha(\{p\}) = \omega(\{p\})$ for every $p \in N_s$, and every transition t not in N_s is taken as non guarded, i.e., t is a simple transition.*

Consider again the system in Figure 7 with ω_1 and ω_2 as defined above. Then, $N(N_s, \omega_1)$ is the whole net in Figure 7 with $\alpha(\{p_5\}) = 0$ and $\alpha(\{p_6\}) = 1$ ($N(N_s, \omega_2)$ is also the whole net with $\alpha(\{p_5\}) = 1$ and $\alpha(\{p_6\}) = 0$).

Given a basic behavior $\omega \in \Omega(N_s)$ of subnet N_s, the throughput of $N(N_s, \omega)$ is denoted as Th_ω. We say that the cycle Q of a $N(N_s, \omega)$ is *critical* if it fulfills the following equality:

$$Th_\omega = \frac{\sum\limits_{p \in Q} \mathbf{m}_0(p)}{\sum\limits_{t \in Q} \delta(t)} \tag{14}$$

Since $N(N_s, \omega)$ evolves as a conventional timed MG, critical cycles and Th_ω can be computed in an efficient way [Kar78,DG98]. The throughput for the system in Figure 7 for ω_1 is $Th_{\omega_1} = 0.25$.

Notice that if $\alpha(\{p\}) = 0$, p never constraints the firing of $p^\bullet$. In other words, if p is removed the evolution of the system keeps the same. If several places are removed, i.e., $\alpha(\{p\}) = 0$, the resulting net might consists of several non-connected components that would evolve independently. To avoid this phenomenon, it will be assumed that those places not reachable from the critical cycle are removed from $N(N_s, \omega)$, i.e., for every place p in $N(N_s, \omega)$ there is a directed path from the critical cycle to p.

Definition 10 (Tight marking) *A marking $\widetilde{\mathbf{m}}$ such that $\widetilde{\mathbf{m}} \in \mathbb{R}^{|P|}$ is called a* tight *marking of a given $N(N_s, \omega)$ if it satisfies:*

- $\widetilde{\mathbf{m}} = \mathbf{m}_0 + \mathbb{C} \cdot \sigma$
- $\delta(p^\bullet) \cdot Th_\omega \leq \alpha(\{p\}) \cdot \widetilde{\mathbf{m}}(p)$ *for every p*
- *for each t there exists $p \in {}^\bullet t$ such that $\widetilde{\mathbf{m}}(p) = \delta(p^\bullet) \cdot Th_\omega$*

For instance, a tight marking for $N(N_s, \omega_1)$ (where N is the net depicted in Figure 7 and the values for ω_1 are $\omega_1(\{p_5\}) = 0$ and $\omega_1(\{p_6\}) = 1$) is $\widetilde{\mathbf{m}} = (0.5; 0.25; 0.25; 0.25; 0.75; 0.25; 0.25; 0.25; 0.25; 0.25; 0.75)$.

Notice that the average marking, $\overline{\mathbf{m}}$, of $N(N_s, \omega)$ fulfills the first two constraints of Definition 10. Hence, it makes sense to take the tight marking as an estimate for the average marking. The places satisfying the last equation, $\widetilde{\mathbf{m}}(p) = \delta(p^\bullet) \cdot Th_\omega$, are called *tight* places, and can be seen as the actual throughput constraints. Tight places will be used by an iterative process to build a system bottleneck.

A tight marking of a given $N(N_s, \omega)$ can be obtained by means of a linear programming problem.

Proposition 8 *A tight marking, $\widetilde{\mathbf{m}}$, of a given $N(N_s, \omega)$ can be computed by solving the following LPP:*

$$\begin{aligned} &\textit{Maximize } \Sigma\sigma : \\ &\quad \widetilde{\mathbf{m}} = \mathbf{m}_0 + \mathbb{C} \cdot \sigma \\ &\quad \delta(p^\bullet) \cdot Th_\omega \leq \alpha(\{p\}) \cdot \widetilde{\mathbf{m}}(p) \quad \textit{for every } p \\ &\quad \sigma(t_a) = k \end{aligned} \tag{15}$$

where t_a is a transition that belongs to a critical cycle and k is any real constant number.

Proof: a) proves that the LPP is feasible and bounded. b) proves that the solution is a tight marking.

a) The set of solutions that satisfy the first two constraints is not empty, e.g., the average marking satisfies them. Since in $N(N_s, \omega)$ the vector $\mathbf{1}$ is a right annuller of $\mathbb{C}$, a marking $\widetilde{\mathbf{m}} = \mathbf{m}_0 + \mathbb{C} \cdot \sigma$ can be obtained with any firing sequence $\sigma' = \sigma + j \cdot \mathbf{1}$ where j is a real number. Hence, the third constraint $\sigma(t_a) = k$ can also be satisfied by the average marking. Therefore, the LPP is feasible.

The third constraint forces $\sigma(t_a) = k$. Let $p \in t_a^\bullet$ and $p^\bullet$ is a simple transition or $\alpha(\{p\}) > 0$. Then $\sigma(p^\bullet)$ cannot tend to infinity, otherwise $\widetilde{\mathbf{m}}(p)$ would tend to minus infinity and the second constraint $\delta(p^\bullet) \cdot Th_\omega \leq \alpha(\{p\}) \cdot \widetilde{\mathbf{m}}(p)$ would be violated. Since we assumed that in $N(N_s, \omega)$ every place is reachable from the critical cycle this reasoning can be extended to the rest of places in $N(N_s, \omega)$.

b) Since t_a is in a critical cycle, there exists a place $p \in {}^\bullet t_a$ such that p is in the same critical cycle. Hence, in order to satisfy (14) and the second constraint in (15), the solution of the LPP will fulfill that $\widetilde{\mathbf{m}}(p) = \delta(p^\bullet) \cdot Th_\omega$. Given that the objective function $\Sigma\sigma$ is maximized, for every transition t in $N(N_s, \omega)$ there will necessarily exist $p \in {}^\bullet t$ such that $\widetilde{\mathbf{m}}(p) = \delta(t) \cdot Th_\omega$ (otherwise $\sigma(t)$ could be greater). Hence, the obtained marking $\widetilde{\mathbf{m}}$ is a tight marking. □

5.2 Computing a system bottleneck

Subsection 4.2 showed that solving LPP (13) can be interpreted as a search for the bottleneck P-semiflow in the transformed net. Once the LPP is solved, the places of such bottleneck are given by the components of $\mathbf{y}$ that are positive. That is, a place p belongs to the bottleneck iff in the solution of the LPP it holds that $\mathbf{y}(p) > 0$; consequently a transition t belongs to the bottleneck iff the place ${}^\bullet t$ belongs to the bottleneck.

Recall that in the transformed net the input places of each multi-guarded transition were merged into a single one, see Figure 6. Hence, the bottleneck of the original net must include every place that after the transformation, resulted in a single place included in the bottleneck. For instance, if p_1 in Figure 6(right) is included in the bottleneck, then p_1 and p_3 of the original net, i.e., Figure 6(left), must be included in the bottleneck.

The subnet composed of the places and transitions in the bottleneck computed by LPP (13) will be denoted as N_b. We will now present a fix point algorithm in which N_b is taken as the initial bottleneck net of the system, and is enlarged in order to obtain a subnet, N_z, that approximates the throughput of the whole system.

The algorithm starts by exploring the different basic behaviors of N_b. For each basic behavior ω, a tight marking is computed. The tight places p, i.e., places fulfilling $\widetilde{\mathbf{m}}(p) = \delta(p^\bullet) \cdot th_\omega$, are considered as constraints for the throughput of ω and therefore they are included in the new bottleneck N_z. After exploring the basic behaviors, the strongly connected component (SCC) of N_z is taken as the seed for the next iteration.

Algorithm 9 (Computing a system bottleneck)

```
Input: $N_b$
Output: $N_z$
$N_z = N_b$
Repeat
    $N_{z0} = N_z$
    for each $\omega \in \Omega(N_{z0})$ of $N_{z0}$
        Compute $\widetilde{\mathbf{m}}_\omega$ and $Th_\omega$ of $N(N_{z0}, \omega)$
        $N_z = N_z \cup \{p | p$ is a tight place in $\widetilde{\mathbf{m}}_\omega\}$
    end_for
    $N_z = SCC(N_z)$
until $N_z = N_{z0}$
```

Example 4 *Let $\alpha(\{p_5\}) = 0.25$ for the system in Fig. 7, then the set of places in N_b is $\{p_4, p_5, p_6, p_7, p_8, p_9\}$. The first inner iteration of the algorithm computes the tight marking for ω_1 ($\omega_1(\{p_5\}) = 0$, $\omega_1(\{p_6\}) = 1$), whose tight places are $\{p_2, p_3, p_4, p_6, p_7, p_8, p_9, p_{10}\}$.*

The second inner iteration takes ω_2 ($\omega_2(\{p_5\}) = 1$, $\omega_1(\{p_6\}) = 0$), and obtains these tight places $\{p_1, p_2, p_3, p_5, p_7, p_8, p_9, p_{10}\}$. Hence, at the end of the inner loop, N_z contains all places but p_{11}. Therefore, p_{10} is removed from N_z when the strongly connected component is computed. Neither p_{10} nor p_{11} will be added to N_z in the next outer iteration, and the algorithm will finish. In fact, p_{11} does never constrain the firing of its output transitions. It can be checked that the same N_z is obtained if one takes $\alpha(\{p_5\}) > 0.5$, which implies that N_b is $\{p_1, p_2, p_3\}$.

The number of basic behaviors of a TGMG is $\prod_{t \in T} |G(t)|$ where $|G(t)|$ is the number of guards of t. Hence, the number of basic behaviors to explore might explode as the algorithm executes. Moreover, the subnet N_z might not be very useful if it is very large. It is, of course, possible to stop the execution of the algorithm when the size of N_z or the number of basic behaviors to explore reach a given limit. The subnet N_z can be seen as the part of the net in which design efforts must be focused. Table 2 shows, for a set of TGMGs, the size of the obtained bottlenecks and their associated steady state throughputs.

6 Experimental results

This section is divided into two subsections: the first one reports the results concerning the computation of throughput bounds via linear programming, see Section 4; the second one refers to the throughput of the system bottlenecks computed by the algorithm in Section 5.

The experiments have been performed on a set of circuits taken from the ISCAS-89 benchmarks [BBK89]. The largest strongly connected component of each circuit is taken and transformed into a TGMG as follows: a) each node is taken as a transition and each arc as a place; b) a place is initially marked with one token with a probability of 0.25; b) the deterministic delay of each transition is assigned uniformly random from the interval $[0,1]$; c) a transition is taken as multi-guarded, i.e., as a multiplexer, with probability 0.25; d) each input place of a multi-guarded transition is taken as a guard; e) the probability of each guard is assigned uniformly random from the interval $(0,1)$; f) the TGMG is made 1-upperbounded as discussed in Section 2. The simulations were carried out using the independent replication method [Kob78, Law07]. The precision of the computed throughput was set to 1% with a confidence level of 95%. All the experiments were performed in Matlab 7.3 environment running on Linux in a 2.0 GHz processor.

6.1 Throughput bounds

Section 4 showed that an upper bound for the throughput of a TGMG can be computed in polynomial time by solving the LPP (13). One can also obtain a lower throughput bound for a TGMG just by considering all transitions as non-guarded, and then computing the throughput of the resulting conventional marked graph. This computation can be efficiently carried out [DG98]. Table 1 reports the obtained throughput bounds, the committed errors with respect to the real throughput of the system and the required CPU times.

Circuit			Throughput					CPU time	
Name	$\|P\|$	$\|T\|$	LowLP	UpLP	MeanLP	Sim	Err	LPcpu	Simcpu
s27	136	58	0.2252	0.2847	0.2549	0.2699	5.55%	0.13 s	163.32 s
s208	66	30	0.4182	0.5714	0.4948	0.5465	9.46%	0.05 s	156.61 s
s349	442	180	0.2637	0.2637	0.2637	0.2637	0.00%	0.21 s	562.72 s
s382	172	68	0.1389	0.1523	0.1456	0.1522	4.32%	0.26 s	114.13 s
s386	778	306	0.1436	0.2518	0.1977	0.2364	16.37%	0.52 s	956.51 s
s400	180	70	0.4663	0.5224	0.4944	0.4949	0.11%	0.34 s	386.43 s
s444	204	78	0.4125	0.4337	0.4231	0.4274	1.02%	0.39 s	378.15 s
s510	2416	904	0.2335	0.3004	0.2670	0.2691	0.79%	8.03 s	4664.03 s
s526	278	118	0.2001	0.3500	0.2751	0.3429	19.78%	0.13 s	447.40 s
s713	654	264	0.3292	0.3476	0.3384	0.3379	0.15%	0.42 s	1129.42 s
s820	2740	1056	0.2489	0.3510	0.3000	0.3105	3.40%	9.32 s	6640.29 s
s832	3062	1186	0.1329	0.1329	0.1329	0.1329	0.00%	9.78 s	3411.44 s
s953	982	388	0.3202	0.3412	0.3307	0.3246	1.88%	0.84 s	1751.89 s
s1423	2582	976	0.1842	0.2106	0.1974	0.2085	5.34%	1.81 s	4063.12 s
s1488	3718	1420	0.2340	0.2759	0.2549	0.2615	2.49%	20.28 s	8818.12 s
s1494	3712	1420	0.1420	0.2632	0.2026	0.2168	6.55%	15.28 s	7341.48 s

Table 1: Throughput bounds obtained by linear programming.

The columns $|P|$ and $|T|$ are the number of places and transitions of the TGMGs. The value of the obtained lower and upper bounds are shown in columns LowLP and UpLP. The column MeanLP stands for the average $(LowLP+UpLP)/2$ which is taken as the estimation for the system throughput. The column Sim is the throughput of the system obtained by simulation. The relative error yielded by MeanLP with respect to the middle point of the confidence interval given by Sim is reported in Err. The columns LPcpu and Simcpu express CPU times, in seconds, required to compute MeanLP and Sim.

One of the main advantages of using an LPP to obtain a throughput bound is that its complexity is polynomial, and, in general, provides a good approximation to the real throughput. Thus, it is a suitable method for fast system evaluation.

6.2 System bottlenecks

Table 2 illustrates the results obtained after applying Algorithm 9 on the circuits of the previous subsection. The following heuristics has been followed: The maximum number of basic behaviors considered is limited to the exploration of 5 multiplexers in N_z. When there are more than 5 multiplexers in N_b only those with probabilities closer to 0.5 are explored, the rest are handled in the same way as in LPP (9).

Circuit	System bottleneck				Throughput		CPU time	
Name	$\|sbP\|$	$\|sbT\|$	Reduc	sbcpu	sbSim	Err	sbSim	Sim
s27	30	23	27.3%	2.4 s	0.2762	2.33%	33.8 s	163.3 s
s208	31	22	55.2%	0.9 s	0.5491	0.46%	66.7 s	156.6 s
s349	6	6	1.9%	0.8 s	0.2637	0.00%	7.1 s	562.7 s
s382	12	11	9.6%	0.7 s	0.1522	0.00%	8.0 s	114.1 s
s386	66	48	10.5%	54.2 s	0.2377	0.54%	63.1 s	956.5 s
s400	61	43	41.6%	9.9 s	0.4953	0.08%	120.0 s	386.4 s
s444	41	30	25.2%	5.5 s	0.4292	0.40%	70.0 s	378.1 s
s510	122	102	6.7%	593.2 s	0.2647	1.66%	147.8 s	4767.5 s
s526	13	11	6.1%	0.7 s	0.3456	0.78%	19.0 s	447.4 s
s713	194	134	35.7%	308.9 s	0.3476	2.87%	287.1 s	1129.4 s
s820	133	112	6.5%	506.8 s	0.3058	1.54%	174.1 s	6640.2 s
s832	12	12	0.6%	19.3 s	0.1329	0.00%	7.2 s	3411.4 s
s953	113	85	14.5%	56.0 s	0.3246	0.01%	155.8 s	1751.8 s
s1423	120	89	5.9%	1128.1 s	0.2062	1.10%	104.8 s	4063.1 s
s1488	190	167	6.9%	765.6 s	0.2507	4.30%	228.5 s	8818.1 s
s1494	368	293	12.9%	916.5 s	0.2167	0.04%	398.3 s	7341.4 s

Table 2: Throughput approximations obtained with system bottlenecks.

The columns $|sbP|$ and $|sbT|$ are the number of places and transitions of the system bottleneck. The column Reduc expresses the reduction ratio, it is computed as $100 \cdot (|sbP| + |sbT|)/(|P| + |T|)$. The column sbcpu stands for the CPU time spent to compute the system bottleneck. The throughputs, computed by simulation, of the system bottlenecks are presented in sbSim (the throughputs of the original systems are shown in Table 1). The column Err is the relative error of the sbSim with respect to Sim in Table 1. The CPU times to simulate the system bottleneck and the original system are shown under the title 'CPU time' (sbSim for the bottleneck system, Sim for the original one). All CPU times are expressed in seconds.

The exploration of at most 5 multiplexers limits the time execution of the algorithm and the size of the system bottleneck N_z. This way, some "relevant" places might not be included in N_z, causing N_z to have a similar, not equal, throughput to the original system throughput. However, from a practical point of view, it can be worth dealing with a much smaller, yet representative, net at the cost of losing some precision.

7 Conclusions

In conventional marked graphs a transition cannot fire until all its input places are marked. This constraint turns out to be excessively restrictive when modeling the behavior of actions that only require a subset of input elements to start performing. Typical examples of this behavior are found in electronic circuits, e.g., a multiplexer can yield an output as soon as the selected input arrives. The class of multi-guarded marked graphs extends the modeling power of marked graphs to capture these behaviors.

A timed multi-guarded marked graph evolves as a semi-Markov process. Thus, it is theoretically possible to compute the exact throughput of a system by solving its embedded Markov chain. Unfortunately, the state explosion problem prevents, in practice, the use of this approach. Two different methods have been developed that avoid the state explosion problem and estimate the system throughput. The first method makes use of an algebraic expression that relates the throughput of a transition and the average marking of its input places. This expression allows one to easily establish a linear programming problem whose solution is an upper bound for the throughput.

The second method is a heuristics that extracts from a timed multi-guarded marked graph a bottleneck with similar throughput. The heuristics is inspired by conventional marked graphs, in which the throughput of a critical cycle is equal to the throughput of the whole system. The method is based on the solution of the linear programming problem to compute throughput bounds and on the concept of tight marking.

References

[BBK89] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. *Circuits and Systems, 1989., IEEE International Symposium on*, pages 1929–1934 vol.3, May 1989.

[BC99] D. Bertsimas and T. Chryssikou. Bounds and policies for dynamic routing in loss networks. *Oper. Res.*, 47(3):379–394, 1999.

[CAC$^+$95] G. Chiola, C. Anglano, J. Campos, J. M. Colom, and M. Silva. Operational Analysis of Timed Petri Nets and Application to the Computation of Performance Bounds. In F. Baccelli, A. Jean-Marie, and I. Mitrani, editors, *Quantitative Methods in Parallel Systems*, pages 161–174. Springer, 1995. Also appears in Procs. PNPM93.

[CS92] J. Campos and M. Silva. Structural Techniques and Performance Bounds of Stochastic Petri Net Models. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 352–391. Springer, 1992.

[DG98] A. Dasdan and R. K. Gupta. Faster maximum and minimum mean cycle algorithms for system performance analysis. *IEEE Transactions on Computer-Aided Design*, 17(10):889–899, 1998.

[DK98] I. Demongodin and N. T. Koussoulas. Differential Petri nets: representing continuous systems in a discrete-event world. *IEEE Transactions on Automatic Control*, 43(4):573–579, April 1998.

[JCK06] J. Júlvez, J. Cortadella, and M. Kishinevsky. Performance analysis of concurrent systems with early evaluation. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, November 2006.

[Jul09] Jorge Julvez. Basic qualitative properties of petri nets with multi-guarded transitions. In *American Control Conference, 2009. ACC '09.*, pages 5026–5031, June 2009.

[Kar78] R. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[KK94] S. Kumar and P.R. Kumar. Performance bounds for queueing networks and scheduling policies. *Automatic Control, IEEE Transactions on*, 39(8):1600–1611, Aug 1994.

[Kob78] H. Kobayashi. *Modeling and Analysis. An Introduction to System Performance Evaluation Methodology*. Addison Wesley, 1978.

[Law07] A. M. Law. *Simulation Modeling and Analysis*. McGraw-Hill, 2007.

[Lit61] J. D. C. Little. A proof of the queueing formula $L = \lambda\, W$. *Operations Research*, 9:383–387, 1961.

[Liu98] Z. Liu. Performance Analysis of Stochastic Timed Petri Nets using Linear Programming Approach. *IEEE Transactions on Software Engineering*, 24:1014–1030, 1998.

[LP05] T.E. Lee and S.H. Park. An extended event graph with negative places and tokens for time window constraints. *IEEE Transactions on Automation Science and Engineering*, 2(4):319–332, Oct. 2005.

[Mur83] K. G. Murty. *Linear Programming*. Wiley and Sons, 1983.

[Mur89] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[MY90] T. Murata and H. Yamaguchi. A Petri net with negative tokens and its application automated reasoning. In *Proceedings of the 33rd Midwest Symposium on Circuits and Systems*, volume 2, pages 762–765, Aug 1990.

[OM05] N. G. Odrey and G. Mejia. An augmented Petri Net approach for error recovery in manufacturing systems control. In *Robotics and Computer-Integrated Manufacturing. 14th International Conference on Flexible Automation and Intelligent Manufacturing.*, volume 21, pages 346–354, August-October 2005.

[Ram74] C. Ramchandani. Analysis of asynchronous concurrent systems by timed Petri nets. Technical Report Project MAC Tech. Rep. 120, Massachusetts Inst. of Tech., February 1974.

[RH80] C. V. Ramamoorthy and G. S. Ho. Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets. *IEEE Trans. on Software Engineering*, 6(5):440–449, 1980.

[Sil93] M. Silva. Introducing Petri Nets. In *Practice of Petri Nets in Manufacturing*, pages 1–62. Chapman & Hall, 1993.

[Wol89] R. W. Wolff. *Stochastic modeling and the theory of queues*. Prentice Hall, 1989.

[YCT84] D. D. Yao, M.L. Chaudry, and J.G.C. Templeton. On bounds for bulk arrival queues. *European Journal of Operational Research*, 15(2):237–243, 1984.

8 Appendix (proof of Theorem 3)

The proof of Theorem 3 is structured in four parts: first, some auxiliary definitions are presented; then, some technical lemmas are introduced; afterwards, an expression for the enabling degree is developed; finally, the result is proved. As stated in Section 2, every TGMG is assumed to be bounded and live, and every guard is assumed to be a singleton.

8.1 Auxiliary Definitions

Each input place p of a multi-guarded transition t can be seen as a queue where tokens are arranged in cells. At a given marking each cell is either empty or stores a positive token or stores a negative token. Figure 8(b) shows the distribution in cells of the tokens in places p_1 and p_2 of Figure 8(a).

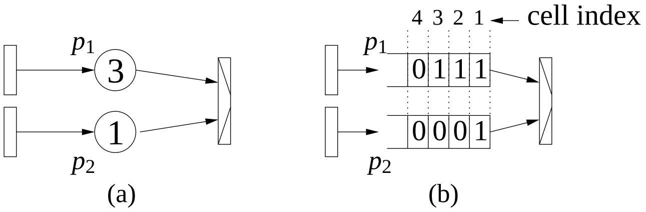

Figure 8: Each place can be seen as a queue where tokens are arranged in cells.

Cells are indexed according to their proximity to the multi-guarded transition, see Figure 8(b). When a token arrives into a place, it is stored in the cell with lowest index that is empty or has a negative token. Each cell corresponds to a firing instance of the transition. When the transition fires, one token is removed from the corresponding cell of every input place, and a new guard is selected for the next cell, i.e., for the next fire.

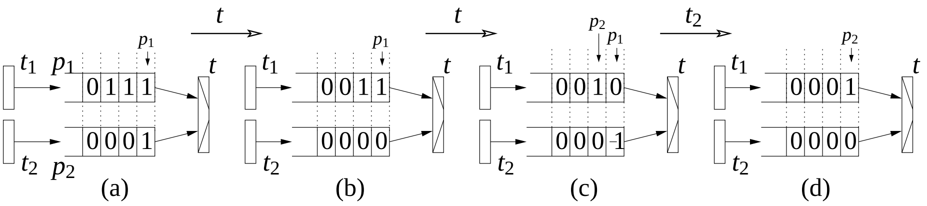

Figure 9: Each pair of cells with the same index corresponds to a firing instance.

Consider the net in queue-like form in Figure 9 with $G(t) = \{\{p_1\},\{p_2\}\}$ to show how the marking of the different cells evolves. Assume that the guard for the first firing of t is $\{p_1\}$ (see above each cell in Figure 9). Thus, the first firing requires one token in p_1. Hence, t is enabled in Figure 9(a), and will fire after $\delta(t)$ time units. The firing of t removes one token from p_1 and one token from p_2. After the firing of t, the first cell of each place becomes empty, and hence we can shift every cell to the right (this is equivalent to remove the first cell and decrease the index of the rest), see Figure 9(b). Assume that the guard for the second firing of t is again $\{p_1\}$, then, t is enabled in Figure 9(b). The second firing of t produces a negative token in p_2, Figure 9(c). Assume that the guard for the third firing of t is again $\{p_2\}$, then, t is not enabled in Figure 9(c), and will not be enabled until a positive token is stored in the corresponding cell of p_2. If t_2 fires from the marking in Figure 9(c) it will cancel out the existing negative token and the cells can be shifted to the right again.

The marking of the l^{th} cell of p_i is denoted as $\mathbf{m}(p_i,l)$, and the guard for the l^{th} cell is denoted as $g(t,l)$. The fact that a guard for a given cell l has not been selected yet is denoted by $g(t,l)=\{\}$.

For each input place p_i of a multi-guarded transition and each cell l, we define three boolean functions, $p^+_{i,l}$, $p^0_{i,l}$, and $p^-_{i,l}$, that take a marking $\mathbf{m}$ as input argument:

- $p^+_{i,l}(\mathbf{m}) \leftrightarrow\ (\mathbf{m}(p_i,l)=1 \wedge g(t,l)\neq\{\})$
- $p^0_{i,l}(\mathbf{m}) \leftrightarrow\ (\mathbf{m}(p_i,l)=0 \wedge g(t,l)\neq\{\})$
- $p^-_{i,l}(\mathbf{m}) \leftrightarrow\ (\mathbf{m}(p_i,l)=-1 \wedge g(t,l)\neq\{\})$

For clarity, the argument $\mathbf{m}$ of the above functions will not be shown if it is evident from the context. We will use the shorthand α_i to denote $\alpha(\{p_i\})$, and the boolean variable $g_{i,l}$ that is true iff $g(t,l)=\{p_i\}$. The fact that $g(t,l)\neq\{p_i\}$ is denoted as $not(g_{i,l})$. Consider the next statement to illustrate the use of the introduced notation: transition t is enabled at $\mathbf{m}'$ iff there exists $p_i \in {}^\bullet t$ and a cell l such that $(p^+_{i,l}(\mathbf{m}')\wedge g_{i,l})$ (or simply $(p^+_{i,l}\wedge g_{i,l})$ given that the argument of $p^+_{i,l}$ is clearly $\mathbf{m}'$).

By $prob(statement)$ we denote the fraction of time *statement* holds in the steady state. For instance, $prob(p^+_{j,l}\wedge p^0_{k,l}\wedge g_{k,l})$ is the fraction of time at which the system marking satisfies that $\mathbf{m}(p_j,l)=1$, $\mathbf{m}(p_k,l)=0$, and $g(t,l)=\{p_k\}$.

8.2 Preliminary Lemmas

The developments in this section focus on cells that have the same index l. Then, for clarity, $p^+_{i,l}$, $p^0_{i,l}$, $p^-_{i,l}$, $g_{i,l}$ are shortened to p^+_i, p^0_i, p^-_i, g_i.

Lemma 10 *Let $\langle N,\mathbf{m}_0\rangle$ be a TGMG system. Let t be a multi-guarded transition of the TGMG and $p_j,p_k,p_q\in{}^\bullet t$, then it holds that:*

$$prob(p^+_j\wedge g_q)=prob(p^+_j\wedge p^0_k\wedge g_q)+prob(p^+_j\wedge p^+_k\wedge g_q) \tag{16}$$

Proof: If p^+_j holds then the transition has not fired yet. This implies that $p^0_k\vee p^+_k$ holds for every $k\in\{1,\ldots,n\}$, what leads directly to the result. □

From (16), the following equations are trivially obtained for ever couple of input places of the multi-guarded transition t:

$$prob(p^+_j\wedge g_k)=prob(p^+_j\wedge p^0_k\wedge g_k)+prob(p^+_j\wedge p^+_k\wedge g_k)\ \ \forall p_j,p_k\in{}^\bullet t \tag{17}$$

$$prob(p^+_j\wedge g_j)=prob(p^+_j\wedge p^0_k\wedge g_j)+prob(p^+_j\wedge p^+_k\wedge g_j)\ \ \forall p_j,p_k\in{}^\bullet t \tag{18}$$

Lemma 11 *Let $\langle N,\mathbf{m}_0\rangle$ be a TGMG system. Let t be a multi-guarded transition of the TGMG and $p_j,p_k\in{}^\bullet t$, then it holds that:*

$$\frac{\alpha_k}{\alpha_j}=\frac{prob(p^+_j\wedge p^0_k\wedge g_k)}{prob(((p^+_j\wedge p^0_k)\vee p^-_k)\wedge g_j)} \tag{19}$$

Proof: The proof is based on the semi-Markov process associated with the TGMG system. Let $H(t)$ denote the set of states of the semi-Markov process from which a guard for t is selected. For instance $H(t_1) = \{S_1, S_4, S_7\}$ in Figure 5(a). Then, among the successors of $h \in H(t)$, e.g., state S_1, there are $|^\bullet t| = n$ states, $h_1, \ldots, h_n$ that only differ in the guard selected, e.g., states S_2 and S_3, and consequently they can also differ in the time to fire of transitions.

If the statement $(p_j^+ \wedge p_k^0)$ does not hold for any successor of $H(t)$, then $p_j^+ \to p_k^+$ is true in any successor of $H(t)$, consequently $prob(p_k^- \wedge g_j) = 0$ and the lemma trivially holds. Assume there exists $h \in H(t,l)$, such that a successor h_j of h exists at which $(p_j^+ \wedge p_k^0 \wedge g_j)$ holds, e.g., $(p_{ae}^+ \wedge p_{de}^0 \wedge g_{ae})$ holds in S_3. Then, a successor h_k, e.g., S_2, of h that only differs from h_j in the guard must exists, namely at h_k the statement $(p_j^+ \wedge p_k^0 \wedge g_k)$ holds.

Given that the guard selected for cell l does not affect the system evolution neither from h to h_j nor from h to h_k, $\dfrac{\alpha_k}{\alpha_j}$ is the ratio of visits between states h_k and h_j. The lemma is true if the time fraction during which $(p_j^+ \wedge p_k^0 \wedge g_k)$ holds from h_k is not less than the time fraction during which $(((p_j^+ \wedge p_k^0) \vee p_k^-) \wedge g_j)$ holds from h_j. The statement $(p_j^+ \wedge p_k^0)$ will hold from h_k until one token is put in cell l of p_k. On the other hand, the statement $(((p_j^+ \wedge p_k^0) \vee p_k^-) \wedge g_j)$ will hold from h_j until one token is put in cell l of p_k.

States h_j, e.g., S_3, and h_k, e.g., S_2, only differ in the guards, and therefore, they can also differ in the fact that t is enabled in h_j, and will fire after $\delta(t)$ time units, but it is not enabled in h_k. Given that every TGMG is assumed to be live, every P-semiflow (or cycle) has a positive number of tokens (Theorem 1). Hence, at h_j and h_k the overall number of tokens in any directed path from t to $^\bullet p_k$ is at least 1. Then, the first firing of $^\bullet p_k$ from h_j and h_k do not require the token that t will produce. In other words, the time elapsed until one token is put in cell l of p_k does not depend on the guard selected at h. Therefore, $prob(p_j^+ \wedge p_k^0 \wedge g_k)$ and $prob(((p_j^+ \wedge p_k^0) \vee p_k^-) \wedge g_j)$ are just related by the probabilities of selecting $\{p_k\}$ or $\{p_j\}$ as guard from state h. □

In the rest of the section we will assume, without loss of generality, that the input places of t are indexed from 1 to n, i.e., $^\bullet t = \{p_1, \ldots, p_n\}$.

Lemma 12 *Let $\langle N, \mathbf{m}_0 \rangle$ be a TGMG system. Let t be a multi-guarded transition of the TGMG and $p_j \in {}^\bullet t$, then it holds that:*

$$\begin{aligned}\alpha_j \cdot prob(p_j^+ \wedge \ not(g_j)) = \sum_{\substack{k=1\\k\neq j}}^{n} \alpha_j \cdot prob(p_j^+ \wedge p_k^+ \wedge g_k) + \sum_{\substack{k=1\\k\neq j}}^{n} \alpha_k \cdot prob(p_j^+ \wedge p_k^0 \wedge g_j)\\ + \sum_{\substack{k=1\\k\neq j}}^{n} \alpha_k \cdot prob(p_k^- \wedge g_j)\end{aligned} \tag{20}$$

Proof: Equation (17) allows us to expand the term $\alpha_j \cdot prob(p_j^+ \wedge \ not(g_j))$ as follows:

$$\begin{aligned}\alpha_j \cdot prob(p_j^+ \wedge not(g_j)) &= \alpha_j \cdot \sum_{\substack{k=1\\k\neq j}}^{n} prob(p_j^+ \wedge g_k)\\ &= \alpha_j \cdot \sum_{\substack{k=1\\k\neq j}}^{n} \big(prob(p_j^+ \wedge p_k^0 \wedge g_k) + prob(p_j^+ \wedge p_k^+ \wedge g_k)\big)\end{aligned}$$

$$= \sum_{\substack{k=1\\k\neq j}}^{n} \alpha_j \cdot prob(p_j^+ \wedge p_k^+ \wedge g_k) + \sum_{\substack{k=1\\k\neq j}}^{n} \alpha_j \cdot prob(p_j^+ \wedge p_k^0 \wedge g_k) \qquad (21)$$

The equality (19) provided by Lemma 11 can be written as:

$$\alpha_j \cdot prob(p_j^+ \wedge p_k^0 \wedge g_k) \geq \alpha_k \cdot prob(p_j^+ \wedge p_k^0 \wedge g_j) + \alpha_k \cdot prob(p_k^- \wedge g_j) \qquad (22)$$

The substitution of $\alpha_j \cdot prob(p_j^+ \wedge p_k^0 \wedge g_k)$ in (21) by the expression on the right of (22) yields:

$$\alpha_j \cdot prob(p_j^+ \wedge\ not(g_j)) = \sum_{\substack{k=1\\k\neq j}}^{n} \alpha_j \cdot prob(p_j^+ \wedge p_k^+ \wedge g_k) + \sum_{\substack{k=1\\k\neq j}}^{n} \alpha_k \cdot prob(p_j^+ \wedge p_k^0 \wedge g_j) + \sum_{\substack{k=1\\k\neq j}}^{n} \alpha_k \cdot prob(p_k^- \wedge g_j)$$

□

8.3 Enabling of a Cell

In this subsection, we will obtain an expression for the probability that *a cell is enabled*. A cell l of input places of t is said to be enabled if $p_{j,l}^+ \wedge\ g_{j,l}$ holds for any input place, p_j, of t. The boolean function $enab(t,l)$ is defined to be true iff cell l is enabled. Then, the term $prob(enab(t,l))$ is the fraction of time that cell l is enabled. More formally:

$$prob(enab(t,l)) = \sum_{p_j \in {}^\bullet t} prob(p_{j,l}^+ \wedge\ g_{j,l})$$

As in the previous developments, we will focus only one cell l, then for clarity the subindex l is omitted.

Lemma 13 *Let $\langle N, \mathbf{m}_0 \rangle$ be a TGMG system. Let t be a multi-guarded transition of the TGMG and $p_j \in {}^\bullet t$, then:*

$$prob(enab(t,l)) = \sum_{p_j \in {}^\bullet t} \alpha_j \cdot \left(prob(p_j^+) - prob(p_j^-)\right) \qquad (23)$$

Proof: Let us develop $prob(enab(t,l))$ as follows:

$$prob(enab(t,l)) = \sum_{p_j \in {}^\bullet t} prob(p_j^+ \wedge\ g_j) = \sum_{p_j \in {}^\bullet t} \Big(prob(p_j^+) - prob(p_j^+ \wedge\ not(g_j))\Big)$$

$$= \sum_{p_j \in {}^\bullet t} \Big(prob(p_j^+) - \big((1-\alpha_j) \cdot prob(p_j^+ \wedge not(g_j)) + \alpha_j \cdot prob(p_j^+ \wedge not(g_j))\big)\Big) \qquad (24)$$

By Lemma 12, Equation (24) becomes:

$$\begin{aligned} prob(enab(t,l)) = \sum_{p_j \in {}^{\bullet}t} \Big(& prob(p_j^+) - \big((1-\alpha_j) \cdot prob(p_j^+ \wedge\ not(g_j)) \\ & + \sum_{\substack{k=1 \\ k \neq j}}^{n} \alpha_k \cdot prob(p_j^+ \wedge p_k^0 \wedge g_j) + \sum_{\substack{k=1 \\ k \neq j}}^{n} \alpha_j \cdot prob(p_j^+ \wedge p_k^+ \wedge g_k) \\ & + \sum_{\substack{k=1 \\ k \neq j}}^{n} \alpha_k \cdot prob(p_k^- \wedge g_j)\big)\Big) \end{aligned} \tag{25}$$

In (25) let us swap $\alpha_j \cdot prob(p_j^+ \wedge p_k^+ \wedge g_k)$ with $\alpha_k \cdot prob(p_j^+ \wedge p_k^+ \wedge g_j)$, and let us swap $\alpha_k \cdot prob(p_k^- \wedge g_j)$ with $\alpha_j \cdot prob(p_j^- \wedge g_k)$. This rearrangement of terms allows us to express (25) as:

$$\begin{aligned} prob(enab(t,l)) = \sum_{p_j \in {}^{\bullet}t} \Big(& prob(p_j^+) - \big((1-\alpha_j) \cdot prob(p_j^+ \wedge\ not(g_j)) \\ & + \sum_{\substack{k=1 \\ k \neq j}}^{n} \alpha_k \cdot prob(p_j^+ \wedge p_k^0 \wedge g_j) + \sum_{\substack{k=1 \\ k \neq j}}^{n} \alpha_k \cdot prob(p_j^+ \wedge p_k^+ \wedge g_j) \\ & + \sum_{\substack{k=1 \\ k \neq j}}^{n} \alpha_j \cdot prob(p_j^- \wedge g_k)\big)\Big) \end{aligned} \tag{26}$$

Notice that if p_j is selected as guard it cannot become negatively marked after the firing, i.e., $prob(p_j^- \wedge g_j) = 0$, therefore:

$$\sum_{\substack{k=1 \\ k \neq j}}^{n} prob(p_j^- \wedge g_k) = prob(p_j^-) \tag{27}$$

By (27) and (18), Equation (26) becomes:

$$\begin{aligned} prob(enab(t,l)) &= \sum_{p_j \in {}^{\bullet}t} \Big(prob(p_j^+) - \big((1-\alpha_j) \cdot prob(p_j^+ \wedge\ not(g_j)) \\ &\qquad + \sum_{\substack{k=1 \\ k \neq j}}^{n} \alpha_k \cdot prob(p_j^+ \wedge g_j) + \alpha_j \cdot prob(p_j^-)\big)\Big) \\ &= \sum_{p_j \in {}^{\bullet}t} \Big(prob(p_j^+) - \big((1-\alpha_j) \cdot prob(p_j^+ \wedge\ not(g_j)) \\ &\qquad + (1-\alpha_j) \cdot prob(p_j^+ \wedge g_j) + \alpha_j \cdot prob(p_j^-)\big)\Big) \\ &= \sum_{p_j \in {}^{\bullet}t} \Big(prob(p_j^+) - \big((1-\alpha_j) \cdot prob(p_j^+) + \alpha_j \cdot prob(p_j^-)\big)\Big) \\ &= \sum_{p_j \in {}^{\bullet}t} \alpha_j \cdot \big(prob(p_j^+) - prob(p_j^-)\big) \end{aligned}$$

□

8.4 Proof of Theorem 3

Theorem 3: Let t be a transition with singleton guards, then:

$$\delta(t) \cdot Th(t) = \sum_{p \in {}^\bullet t} \alpha(\{p\}) \cdot \Big(\overline{\mathbf{m}}(p) - \sum_{i=2}^{\infty} (i-1) \cdot prob(\mathbf{m}(p) = i)\Big)$$

Proof: Given that transition t is enabled when any of its input cells is enabled, the probability that t is enabled is:

$$prob(enab(t)) = \sum_{l=1}^{\infty} prob(enab(t,l))$$

By Lemma 13, $prob(enab(t,l)) = \sum_{p_j \in {}^\bullet t} \alpha_j \cdot \big(prob(p^+_{j,l}) - prob(p^-_{j,l})\big)$, then:

$$\begin{aligned} prob(enab(t)) &= \sum_{l=1}^{\infty} \sum_{p_j \in {}^\bullet t} \alpha_j \cdot \big(prob(p^+_{j,l}) - prob(p^-_{j,l})\big) \\ &= \sum_{p_j \in {}^\bullet t} \alpha_j \cdot \big(\sum_{l=1}^{\infty} prob(p^+_{j,l}) - \sum_{l=1}^{\infty} prob(p^-_{j,l})\big) \end{aligned}$$

Notice that if the marking of p_j is greater than 0, then at least one cell has a positive token, moreover, a guard must be present for the cell with lowest index containing a token. More formally, $\mathbf{m}(p_j) \geq 1 \rightarrow \exists\, l$ such that $p^+_{j,l}$. The reverse is also true: if a cell has one token and a guard is selected for it then, the marking of p_j is necessarily greater than 0, i.e., $\mathbf{m}(p_j) \geq 1 \leftrightarrow \exists\, l$ such that $p^+_{j,l}$. In terms of probabilities this can be expressed as: $\sum_{l=1}^{\infty} prob(p^+_{j,l}) = prob(\mathbf{m}(p_j) \geq 1)$. On the other hand, the term $\sum_{l=1}^{\infty} prob(p^-_{j,l})$ accounts for every negative token in p_j and can be expressed as: $\sum_{l=1}^{\infty} prob(p^-_{j,l}) = \sum_{i=1}^{\infty} i \cdot prob(\mathbf{m}(p_j) = -i)$. Then:

$$\begin{aligned} prob(enab(t)) &= \sum_{p_j \in {}^\bullet t} \alpha_j \cdot \big(prob(\mathbf{m}(p_j) \geq 1) - \sum_{l=1}^{\infty} prob(p^-_{j,l})\big) \\ &= \sum_{p_j \in {}^\bullet t} \alpha_j \cdot \big(\sum_{i=1}^{\infty} prob(\mathbf{m}(p_j) = i) - \sum_{i=1}^{\infty} i \cdot prob(\mathbf{m}(p_j) = -i)\big) \\ &= \sum_{p_j \in {}^\bullet t} \alpha_j \cdot \big(\sum_{i=1}^{\infty} i \cdot prob(\mathbf{m}(p_j) = i) - \sum_{i=2}^{\infty} (i-1) \cdot prob(\mathbf{m}(p_j) = i) \\ &\qquad - \sum_{i=1}^{\infty} i \cdot prob(\mathbf{m}(p_j) = -i)\big) \\ &= \sum_{p_j \in {}^\bullet t} \alpha_j \cdot \Big(\overline{\mathbf{m}}(p_j) - \sum_{i=2}^{\infty} (i-1) \cdot prob(\mathbf{m}(p_j) = i)\Big) \end{aligned}$$

Thus, by Equation (4):

$$\delta(t) \cdot Th(t) = \sum_{p_j \in {}^\bullet t} \alpha_j \cdot \Big(\overline{\mathbf{m}}(p_j) - \sum_{i=2}^{\infty} (i-1) \cdot prob(\mathbf{m}(p_j) = i)\Big)$$

□