



Periodic scheduling of marked graphs using balanced binary words

Jean-Vivien Millo, Robert de Simone

**RESEARCH
REPORT**

N° 7891

February 2012

Project-Teams AOSTE



Periodic scheduling of marked graphs using balanced binary words

Jean-Vivien Millo, Robert de Simone

Project-Teams AOSTE

Research Report n° 7891 — February 2012 — 30 pages

Abstract: This report presents an algorithm to statically schedule live and strongly connected Marked Graphs (MG). The proposed algorithm computes the best execution where the execution rate is maximal and place sizes are minimal. The proposed algorithm provides transition schedules represented as binary words. These words are chosen to be balanced. The contributions of this paper is the proposed algorithm itself along with the characterization of the best execution of any MG.

Key-words: Marked Graph, Scheduling, Balanced binary word

**RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE**

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Ordonnancement périodique de graphes marqués en utilisant les mots binaires balancés

Résumé : Ce rapport présente un algorithme pour ordonnancer statiquement un graphe marqué fortement connexe et vivant. L'algorithme proposé calcule la meilleure exécution pour laquelle le rendement effectif est maximal et la taille des places est minimale. L'algorithme proposé fournit les ordonnancements de chacun des noeuds de calcul sous la forme de mots binaires. Ces mots sont choisis balancés. Les contributions du rapport sont à la fois l'algorithme proposé lui-même et la caractérisation de la meilleure exécution d'un graphe marqué.

Mots-clés : graphe marqué, ordonnancement, mot binaire balancé

Contents

1	Introduction	3
2	Algorithm overview	4
3	Marked graph	7
3.1	Semantics of execution of an MG	8
3.2	Classical results	9
3.3	Execution rate	9
3.4	Size of places and boundedness	10
3.5	Delay	11
3.6	Latencies	12
3.7	TN-equalization	13
4	Balanced binary words	14
4.1	Finite and infinite binary words	15
4.2	Rotation and transposition	15
4.3	Balanced binary words	16
4.4	Transposition on balanced binary words	16
4.5	Equivalence between rotation and transposition on balanced binary words	17
4.6	From word to schedule	17
5	Balanced scheduling of MG	18
5.1	Algorithm details	19
5.1.1	Step 1: compute k and p	19
5.1.2	Step 2: compute the latest delays position D	19
5.1.3	Steps 3: compute $Exec_{periodic}$	19
5.1.4	Step 4: compute $M_{periodic}$	22
5.1.5	Step 5: compute $Exec_{initial}$	22
5.1.6	Step 6: compute $Exec$	24
5.1.7	Step 7: compute C_{Exec}	24
5.2	Correctness of the step 4	25
5.2.1	Reachability of $M_{periodic}$ from M_0	25
5.2.2	Validity of $Exec_{periodic}$ from $M_{periodic}$	27
5.3	Extension to the simply connected case	27
6	Results and discussion	28

1 Introduction

In the System-on-Chip design domain, the trend is component based design. A new design is assembled from IP components which are interconnected through a network of point-to-point communication channels. In this area, the problem of long wire communication latency has emerged as a limitation [20]. A channel is not able to forward a datum in a single step but requires many.

To solve this problem, a component based design has to be provided with its scheduling to take care of the latency issues. Luca Carloni *et al.* have proposed the theory of Latency Insensitive Design (LID) [14] as a dynamic scheduling solution but LID is greedy in buffering

element. From our initial tentative to improve the LID [12], we have established that a component based design along with its latency issues can be modeled using Marked/Event graph (MG) [17]. Consequently, from the challenge of scheduling a System-on-Chip design, we arrive to the more general and abstract challenge of scheduling an MG with respect to communication and computation latencies.

To enter this challenge, we have developed the proposed algorithm which provide a statically computed execution to any live and strongly connected MG. The proposed algorithm can eventually be applied to any system (software, hardware, production chain) which can be abstracted as an MG with fixed communication and computation latencies.

It is clear from historical results [6, 13, 22] that a live MG always admits an execution irrespectively of the communication or computation latencies. The proposed algorithm consists in computing the *best* ASAP execution where execution rate is maximal and place sizes are minimal. These properties match with the requirements encountered in the domain of System-on-Chip design [14]. Lastly, the proposed algorithm is extended to simply connected MGs. However, the validity of the computed execution relies on the on-demand availability of tokens on global inputs.

Except the proposed algorithm itself, the main contribution of the paper is the characterization of this *best* ASAP execution. From the initial marking, an guided execution shall lead to different markings. From each of these markings, the ASAP execution will be different and token accumulation in the places may vary. For example, in a given ASAP execution, a transition may fire all its tokens in sequence and then stall for the rest of the period, while in another ASAP execution, the same transition is fired every two instants. The first example promotes tokens accumulation. Within this set of ASAP executions, the one with the smallest tokens accumulation is called the *balanced ASAP execution*. This execution always exists and can be analytically computed for any MG. In a balanced ASAP execution, the binary words that represent the activities of the transitions through time (**1** for activity, **0** for inactivity) are all balanced.

Related works Marked graphs is a well studied domain for more than forty years and many works are closely related to ours. [18] state the notions but also some results used in this paper. [13] and [6] are the bases of our scheduling theory.

Historically, some works related to the notion of balancedness can be found in a publication of Jean Bernoulli in the 18th century [7]. Then they appeared as Christoffel words in the 19th century [15]. More recently Christoffel words appear again in [19], and as Sturm words in [8, 3], or as mechanical words in [5]. [9] records the history of balanced binary words.

In [1, 2], balanced binary words are used to balance load of Erlang network. In [4], the authors try to minimize the data lose in a graph with fixed storage capacities by optimally routing data trough communication channels using balanced binary words.

Outline Section 2 runs the proposed algorithm on an example. Section 3 presents the MG definition followed by all the required results about static analysis of MG. Balanced binary words are presented and studied in Section 4. The proposed algorithm is presented in Section 5, followed by the proofs of correctness and then Section 6 discusses our results.

2 Algorithm overview

This section gives an informal overview of the major steps of the proposed algorithm. The vocabulary used is formally defined below. However it mostly refers to the usual and accepted definitions of the same in literature.

Algorithm inputs and outputs The proposed algorithm inputs are the live and strongly connected MG and its initial marking (initial token positions). The proposed algorithm outputs are the computed execution and the size of the places for this execution.

Latency expansion and N-equalization In the MG presented in Figure 1-a, the transition (rectangle) on the top has a computation latency of 1. The right-most place (oval) has a communication latency of 3. Usually, a token goes through a transition instantaneously and through a place in one step. When the computation latency is different from 0, the tokens are kept for some time in the transition. Similarly, the tokens are kept longer in a place when its communication latency is more than 1.

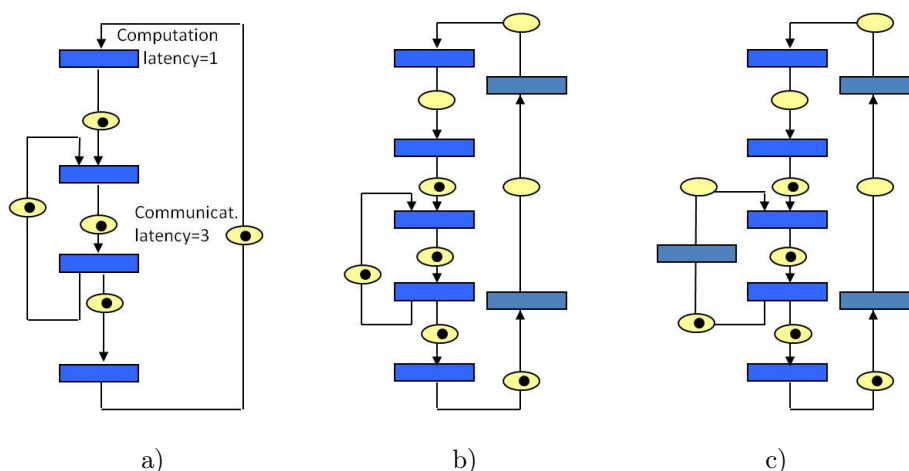


Figure 1: a) an MG with a computation latency on the top-most transition and a communication latency on the right-most place. Its expansion gives the plain MG in b). The MG in c) is the N-equalized version of b).

In this representation, tokens evolution during the MG execution is not obvious. For example, in a place with a communication latency of 3, some tokens could have been there for 1 instant while others have been there for more than 3 instants. The duration of their stay is not explicit.

To avoid this problem, the vertices with latencies are expanded in sequences of plain vertices such that the “semantics of the latency” remains. A place with a communication latency n is replaced by n successive places while a transition with a computation latency m is replaced by $m + 1$ transitions interleaved with m places. Thanks to this transformation, the exact location of tokens is known. Figure 1-b) is the expansion of Figure 1-a).

In every ASAP executions reachable from the initial marking (after guided initialization), token accumulation mostly occurs in the same places. In the MG in Figure 1-b), token accumulation occurs in the left-most place. When the accumulation is such that every token is kept at least 2 instants in the place, the behavior of the place is similar to one with a communication latency of 2. Thus it can be expanded. The N-equalization [12] detects these places analytically and increases their latencies accordingly. In Figure 1-c), the MG is the N-equalized version of the one presented in Figure 1-b).

Running the proposed algorithm on an example The proposed algorithm is defined for an \mathbb{N} -equalized MG where the latencies has been expanded. These steps are considered to be the preliminary steps of the proposed algorithm.

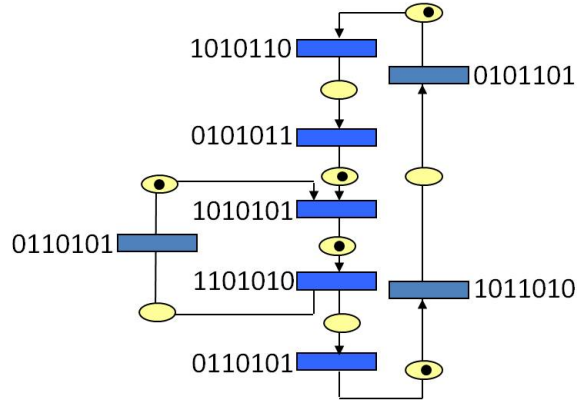


Figure 2: The binary words associated to the transitions express the balanced ASAP execution of the MG introduced in Figure 1-c.

Even in a \mathbb{N} -equalized MG, token accumulation occurs. In some of the ASAP executions (reached after guided initialization), the accumulation is very limited while in others, many tokens can be regrouped in the same place. The balanced ASAP execution ($Exec_{periodic}$) has the lowest accumulation. Figure 2 presents the schedule of every transitions according to $Exec_{periodic}$ (**0** means inactivity, **1** means activity). The first main step of the proposed algorithm computes $Exec_{periodic}$ analytically.

In Figure 2, the marking from which $Exec_{periodic}$ occurs is called $M_{periodic}$. It is different from the initial marking (M_0) (Figure 1-c). The second main step of the proposed algorithm computes $M_{periodic}$.

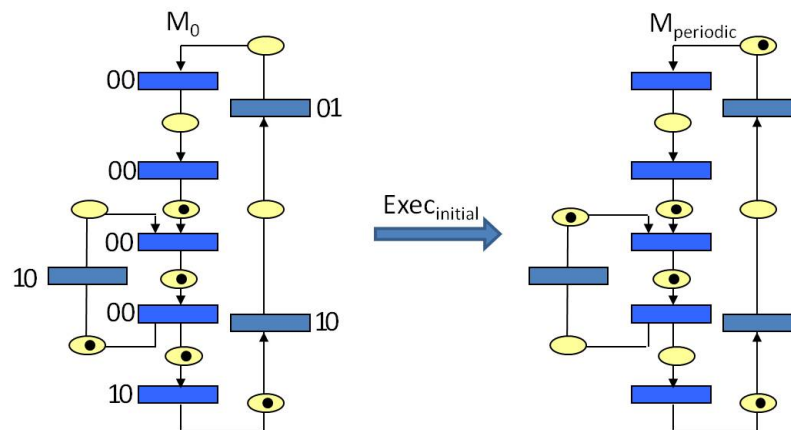


Figure 3: From M_0 on the left to $M_{periodic}$ on the right following the initial guided execution.

The third main step of the proposed algorithm consists in finding the guided initialization

($Exec_{initial}$) leading to $M_{periodic}$ from M_0 . In Figure 3, The 2-bits-length schedules attached to each transition is $Exec_{initial}$.

As one can see in Figure 4, the computed execution is $Exec_{initial}$ followed by the infinite repetition (ω) of $Exec_{periodic}$. It guarantees a maximal execution rate and a minimal accumulation of tokens. The proposed algorithm guarantees that place sizes are either 1 or 2. In the running example, every place size is 1.

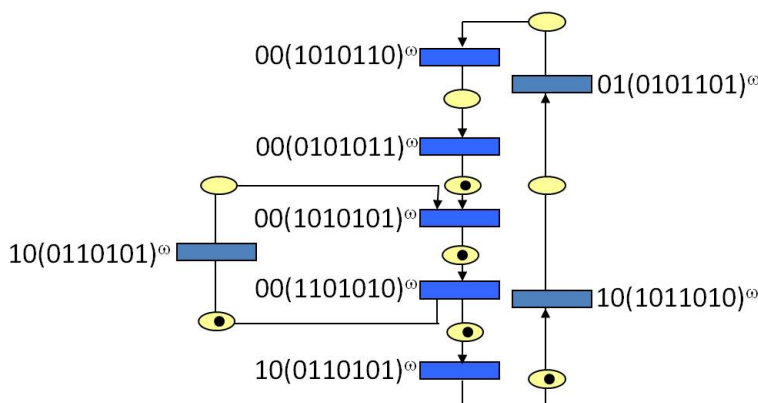


Figure 4: The MG in M_0 and the execution computed by the proposed algorithm.

3 Marked graph

This section presents the *Marked Graph* (MG) model also known as *Event Graph* along with classical definitions and results that will be used in the sequel. Our contributions in this section are the notion of delay presented in Section 3.5 and Theorem 24.

An MG is a graph where vertices can have two types: transitions and places. A place can stock tokens. The edges of a MG are called arcs. They cannot connect two vertices of the same type. A source is a transition without incoming arc. A sink is a transition without outgoing arc.

Definition 1 (Marked Graph). *A marked graph is a structure $G = \langle T, P, F, M_0 \rangle$ where*

- T is a set of transitions.
- P is a set of places.
- $F \subseteq (T \times P) \cup (P \times T)$ is a set of arcs. If $t \in T$ and $p \in P$, (t, p) and (p, t) are two arcs resp. from t to p and from p to t .
- $M : P \rightarrow \mathbb{N}$ is a marking. M_0 is its initial marking.
- Each place has exactly one incoming and one outgoing arcs: $\forall p \in P, |\{(t, p) \mid \forall t \in T\}| = |\{(p, t) \mid \forall t \in T\}| = 1$.

The constraint on the number of place inputs and outputs guarantees that a token can be used by only one transition. Consequently, the MG is said conflict free or deterministic. Figure 1-b presents an MG with 7 transitions (rectangles) and 8 places (ovals). 5 of these places contain one token (black dots).

Notation 2 (Predecessor, successor). *Let G be an MG, $t \in T$ and $p \in P$. We note :*

- $\bullet t$ is the preset of t , $\bullet t = \{p \mid (p, t) \in F\}$.
- t^\bullet is the postset of t , $t^\bullet = \{p \mid (t, p) \in F\}$.
- $\bullet p$ is the transition which precedes p , $\bullet p = t$ such that $(t, p) \in F$.
- p^\bullet is the transition which succeeds p , $p^\bullet = t$ such that $(p, t) \in F$.

Definition 3 (Throughput of an MG, critical element). *Let G be an MG and p be a place of G . A cycle c is a path from p to p . It is called elementary if all the transitions of the cycle are different. The marking of c is $M(c) = \sum_{p \in c} M(p)$ and the latency of c , denoted $L(c)$, is the number of place on c . The value $M(c)/L(c)$ is the throughput of c . The cycle(s) with the lowest throughput is (are) said critical and the throughput of the MG is the one of the critical cycle(s). The transitions, arcs and places are said critical if they belong to a critical cycle.*

An MG is closed if it has neither source nor sink and it is connected if there exists a path, in the underlying undirected graph, relating any pair of vertices. It is strongly connected if there exists a path, in the MG itself, relating any pair of vertices. A strongly connected component (SCC) of an MG is a subgraph that is strongly connected (a subgraph of an MG is an MG composed of a subset of T , a subset of P , and a subset of F); it is said critical (CSCC) if all its elements are critical. A direct acyclic component (DAC) is a subgraph that does not contain any cycle. In general, a connected MG is composed of DACs relating SCCs together. A strongly connected MG is ever closed.

3.1 Semantics of execution of an MG

We define an execution semantics of an MG based on a logical time with a synchronous semantics. At the instant 0, the MG is in its initial marking. Then, an execution step leads to another marking at instant 1 and so on. During a single execution step, many fireable transitions can be fired simultaneously (synchronously) but a single transition can be fired only once.

Definition 4 (Fireable transition at a marking M in an MG). *In an MG G , a transition $t \in T$ is fireable at a marking M if $\forall p \in \bullet t, M(p) > 0$. A source is always fireable. F_M is the set of fireable transitions at a marking M .*

Definition 5 (MG execution model). *Let G be an MG and M its current marking. An execution step is a transition relation from M to M' denoted $M \xrightarrow{FT} M'$ with $FT \subseteq F_M, \forall p \in P, M'(p) = M(p) + FT(\bullet p) - FT(p^\bullet)$. ($FT(t) = 1$ if and only if $t \in FT$. $FT(t) = 0$ otherwise).*

An execution ($Exec$) of an MG is a finite or infinite sequence of execution steps: $Exec = M_0 \xrightarrow{FT_1} M_1 \xrightarrow{FT_2} M_2 \xrightarrow{FT_3} \dots \xrightarrow{FT_i} M_i \xrightarrow{FT_{i+1}} \dots$ where $FT_i \subseteq F_{M_{i-1}}$.

Notation 6 (Concatenation of execution). *Let G be an MG. Let $Exec_0$ be a finite execution of G from the marking M_0 to the marking M_1 and $Exec_1$ be a finite or infinite execution of G from the marking M_1 .*

$Exec_0.Exec_1$ is the execution of G formed by $Exec_0$ followed by $Exec_1$.

Notation 7 (ASAP and guided executions). *Let G be an MG. An execution of G is said As Soon As Possible (ASAP) if and only if $\forall i, FT_i = F_{M_{i-1}}$ (all fireable transitions are ever fired). An execution of G is said guided if and only if $FT_i \subseteq F_{M_{i-1}}$. In a guided execution, one has to decide which fireable transitions are fired at every step.*

Definition 8 (Scheduling and schedule). *Let G be an MG with an execution $Exec$. Let $t \in T$ be a transition of G . The schedule of t is the binary word relating the activity of t : $Sched(t) = FT_1(t).FT_2(t) \cdots FT_i(t) \cdots$. The scheduling of G for an execution $Exec$ is the mapping $t \rightarrow Sched(t) \mid \forall t \in T$.*

Remark 9 (Scheduling and execution). *The successive steps of an execution can be deduced from its scheduling. Consequently, a scheduling defines an execution and vice versa.*

As we have seen in Section 2, the proposed algorithm computes an ASAP execution by computing the schedule of every transition.

3.2 Classical results

Definition 10 (Liveness). *An MG is live if there exists an execution where every transition is fired infinitely often.*

In [17], the authors show that the number of tokens on a cycle remains constant through execution. They deduce an MG is live iff all its cycles contain at least one token.

Definition 11 (Mutually reachable marking). *Let G be a strongly connected MG, M and M' two markings of G . M and M' are mutually reachable if there exists an execution sequence from M to M' and another from M' to M .*

In [18], the authors prove that two live markings, M and M' , of the same strongly connected MG, G , are mutually reachable (through a guided execution) if for every cycle c of G , $M(c) = M'(c)$.

As we have seen in Section 2, the proposed algorithm computes an execution formed by an initial part followed by a steady part. The steady part is not reachable from the initial marking through an ASAP execution. Thus the initial part is a finite guided execution from the initial marking to the first marking of the steady part. This operation is possible because the two markings are mutually reachable.

3.3 Execution rate

In [13], the authors prove that the ASAP execution of a live and strongly connected MG is ultimately repetitive following an execution pattern. Equation 1 shows the evolution of the marking of a live and strongly connected MG. M_0 is the initial marking and the arrows are ASAP execution steps.

$$M_0 \rightarrow M_1 \rightarrow \dots \rightarrow M_i \rightarrow \dots \rightarrow M_{i+p} \quad (1)$$

The period of the pattern is p and the number of firings of every transition within a period is k (the periodicity). We say that, the execution of the MG is k -periodic with a period p . In other words, the *execution rate* is k/p . In [6], the authors give a formula to calculate the exact value of the periodicity (k) and the period (p) of the ASAP execution of a closed MG. According to this formula, the execution rate (k/p) equals the value of the throughput given in Definition 3. Thus the throughput is the maximal execution rate of the MG since it is the one of the ASAP executions. This is the one guaranteed by the proposed algorithm.

Proposition 12 (Maximal execution rate of an MG). *The maximal reachable execution rate is achieved by the ASAP execution of the MG.*

Remark 13 (Execution rate in an open MG). *The result on the maximal execution rate is valid for strongly connected MGs (and thus closed). In a simply connected open MG, the execution rate depends upon the execution rate of the source(s) but the maximal execution rate is bounded by the worst throughput of its SCCs and can be calculated using the same formula given in [6]. Consequently, if the source(s) fire(s) on demand, the MG can be considered closed.*

In Section 5.3, the proposed algorithm is extended to simply connected MG. In such a case, the proposed algorithm returns the schedules of the sources and sinks. The schedule of a source says when a token has to be generated by the source in order to feed the next transition and ensure the overall consistency of the execution.

3.4 Size of places and boundedness

As we have seen in Section 2, the proposed algorithm computes an execution which implies a minimal size of places. Let us now define this notion.

Definition 14 (Size of places). *Let G be an MG, $Exec$ an execution and p a place of G . The size of p on $Exec$ denoted $C_{Exec}(p)$ is the highest marking of p during the entire execution.*

From the initial marking of a strongly connected MG, a guided execution can lead to any of the reachable markings. From each of these markings, there exists a bounded ASAP execution. The size of the places for these executions may vary. An execution has a minimal size of places if every places has a minimal size compared to the other ASAP executions.

Definition 15 (Minimal size of places). *Let G be an MG and M_0 its initial marking. Let \mathbb{M} be the set of markings reachable from M_0 . Let \mathbb{E} be the set of ASAP executions from the markings of \mathbb{M} .*

$Exec \in \mathbb{E}$ has a minimal size of places iff $\forall Exec' \in \mathbb{E}, \forall p \in G$, we have $C_{Exec}(p) \leq C_{Exec'}(p)$.

As we have seen in Section 2, the proposed algorithm computes an ASAP execution where place sizes are minimal. The extension of the proposed algorithm to simply connected MGs is discussed in Section 5.3 but it suffers from limitations since the execution of a simply connected MG may not be bounded.

Definition 16 (Boundedness). *An execution is bounded if the size of every place is bounded. An MG is bounded if every execution is bounded.*

Whereas any SCC is bounded, the sizes of the places of a DAC are not. Let us assume an MG composed of two SCCs connected through a DAC. If the throughput of the upper SCC is superior to the throughput of the underneath SCC, the ASAP execution of the MG will lead to an infinite accumulation of tokens in the DAC. On contrary, If the throughput of the upper SCC is inferior to the throughput of the underneath SCC, the upper SCC will limit the execution rate of the underneath SCC and the behavior of the MG during an ASAP execution will be ultimately repetitive.

Proposition 17 (ASAP execution of a connected MG). *The ASAP execution of a connected MG may be unbounded.*

In every bounded execution of a connected MG, all SCCs have the same execution rate. Consequently, the highest reachable execution rate is equals to the worst throughput among the SCCs. An execution at this rate will be ASAP for the SCCs with the worst throughput. The execution will also be ASAP for the underneath SCCs. However, the upper SCCs will be slow down to avoid accumulation and thus will not have an ASAP behavior.

Proposition 18 (Bounded execution of a connected MG). *The bounded execution of a connected MG may not be ASAP.*

The propositions 17 and 18 explain why the proposed algorithm is restricted to strongly connected MG. However, this restriction can be abolished as discussed in section 5.3. A simply connected MG can be transformed in a strongly connected one (by relating SCCs together) so that the proposed algorithm is applicable.

3.5 Delay

During the execution of an MG, at a given instant, if a token reaches a place p and is not consumed by p^\bullet at the next instant, then the token is said delayed. This can happen in two cases: 1) when p^\bullet is not fired; all the tokens in p are delayed. 2) When p^\bullet is fired and p contains many tokens; all tokens in p excepted the used one are delayed. Globally, delays can be seen as a way for the MG to synchronize its branches together. A non critical cycle leans to take advance over critical cycles (it is faster) but eventually, the execution rate is the same for every one. So the delays reduce the execution rate of fast cycles to the execution rate of the slowest dynamically. The value $M_{c_2} * L_{c_1} - M_{c_1} * L_{c_2}$ represents the number of delay required during a period of execution to synchronize the cycle c_1 with the cycle c_2 .

Definition 19 (Delay). *Let G be an MG and $p \in P$ be a place of G . Let $Exec$ be an execution of G . Following the notation of Definition 5, $Delay(p, i)$ is the number of delays occurring in p at the i^{th} step of $Exec$. $Delay(p, i) = M_{i-1}(p) - FT_i(p^\bullet)$.*

After the initial part, the sum of delays in the places of a cycle during a period of execution reflects the difference of rate between a critical cycle and the current cycle.

Theorem 20 (Delay in a cycle during a period of execution). *Let G be an MG and c a cycle of G . Let $Exec$ be an [ultimately] k -periodic execution of G with a period \mathfrak{p} . Let j_0 be an upper bound of the length of the initialization.*

$$\sum_{p \in c} \sum_{i=1}^{\mathfrak{p}} Delay(p, j_0 + i) = M(c) * \mathfrak{p} - L(c) * k$$

Proof. In c , at each instant, $M(c)$ tokens are present. This means $M(c) * \mathfrak{p}$ transitions could be fired over a period of execution. However, in a period of execution, every transition of G are fired k times. This means $L(c) * k$ transitions are effectively fired on c during a period of execution. The difference between the amount of possible fired transition and the amount of effective fired transition is the number of delays in c over a period. \square

The spatial distribution of delays is the exact location where the delays occur during a period of execution.

Definition 21 (Spatial distribution of delays). *Let G be an MG. Let $Exec$ be an [ultimately] k -periodic execution of G with a period \mathfrak{p} . $D : P \rightarrow \mathbb{N}$ is called a spatial distribution of delays if $\forall c$, the cycles of G , $\sum_{p \in c} D(p) = M(c) * \mathfrak{p} - L(c) * k$.*

Exec is said to be based on D if after the initial part, the delays in exec during a period of execution occur as expressed in D .

In the specific case of an ASAP execution, the delays occur as late as possible in the MG. This makes the corresponding spatial distribution of the delays unique for a given strongly connected MG. This spatial distribution is called the "latest delays position". The theorems 23 and 24 prove these claims.

Definition 22 (Latest delays position). *Let G be a strongly connected MG. Let D be a spatial distribution of the delays in G . D is the latest delays position if for all transition t of G , there exists at least one place p in $\bullet t$ such that $D(p) = 0$.*

Theorem 23 (Existence of the latest delay position). *Let G be a strongly connected MG with a throughput inferior or equal to 1. The latest delay position ever exists for G .*

Proof. A spatial distribution of delays D can be deduced from a period of the ASAP execution of G . $\forall p \in P$, $D(p) = \sum_{i=1}^p \text{Delay}(p, j_0 + i)$ where j_0 is the length of the initial part.

Either D is the latest delay position or there exists at least a transition t for which every places in the preset of t has at least n delays (with $n \geq 1$). In the second case, n delays can be removed to every place in the preset of t and added to every place in the postset of t . This transformation gives another (valid) spatial distribution of delays for which t has at least one place in its preset without delay.

The iteration of this transformation reaches a fix point because no delay depends on the critical cycle. The fix point is the latest delay position. One should note the similarity of this argument to the liveness condition. \square

Theorem 24 (Latest delay position and ASAP execution). *Let G be a strongly connected MG with an execution $Exec$. $Exec$ is based on the spatial distribution of delays D . i) If D is the latest delays position, then $Exec$ is ASAP. ii) Let $Exec'$ be another ASAP execution of G from another initial marking M'_0 . $Exec'$ is based on the spatial distribution of delays D' . If M_0 and M'_0 are mutually reachable, then $D = D'$.*

Proof. i) If $\forall t \in T$, $\exists p \in \{\bullet t\}$ such that $D(p) = 0$, as soon as $M(p) > 0$, t fires. This is a ASAP execution.

ii) If M_0 and M'_0 are mutually reachable, the number of tokens per cycle is the same in M_0 and M'_0 for every cycle of G [18]. Consequently, the number of delays per cycle is the same in D and D' .

Now let us assume there exists a place p such that $D(p) \neq D'(p)$. Let $path_1$ and $path_2$ be two paths in the graphs. $path_1$ goes from a transition of a critical cycle to $\bullet p$ and $path_2$ goes from p^\bullet to a transition of a critical cycle. We assume without lost of generality that the number of delays on $path_1$ is the same according to D and D' . $path_1$ followed by p followed by $path_2$ followed by a section of a critical cycle forms a cycle for which the number of delays is the same according to D and D' . Since $D(p) \neq D'(p)$, the number of delays on $path_2$ is different on D and D' .

Since D and D' are the latest delays position, there exists a path $path_0$ from a transition of a critical cycle to p^\bullet which do not contains any delay (the construction of this path can be done by backtracking from p : while reaching a transition, the input place without delay is selected, a critical cycle will ultimately be reached). But $path_0$ followed by $path_2$ followed by a section of the same critical cycle forms a cycle where the number of delays is different according to D and D' . M_0 and M'_0 are not mutually reachable. \square

As we have seen in Section 2, the proposed algorithm computes an ASAP execution. This ASAP execution is based on the latest delays position of G . In some sense, the proposed algorithm proves that there ever exists an ASAP execution based on the latest delay position.

3.6 Latencies

The preliminary step of the proposed algorithm is the expansion of the vertices with latency in plain vertices.

Definition 25 (MG with communication/ computation latencies).

Let G be an MG. A marked graph with latency G' is a tuple $\langle G, L_{com}, L_{cal} \rangle$:

- The mapping $L_{com}:P \rightarrow \mathbb{N} \setminus \{0\}$ gives the communication latencies of places.
- The mapping $L_{cal}:T \rightarrow \mathbb{N}$ gives the computation latency of transitions (cal stands for calculation).

A place with a communication latency of n keeps every token at least n instants. A transition with a computation latency of m keeps every token exactly m instants. According to Definition 5, the latency of a transition in a plain MG is 0 and the latency of a place is 1. The tokens go through transitions instantaneously but stay at least one instant in a place. The transformation from an MG with latencies to an MG without latency has been introduced by Chander Ramchandani in [22]. This transformation preserves the semantics of a latency.

Figure 1-a presents an MG with computation latencies on the top transition and communication latencies on the right-most place. Figure 1-b is the expansion of Figure 1-a. The top-most transition is replaced by two transitions with a place in between which represents the computation latency. The right-most place is replaced by three places interlaced by two transitions. Each of the three places represents a communication latency.

Liveness, closedness, (strongly) connection, throughput, execution rate, number of cycles, and number of tokens per cycle remain constant through the latency expansion process.

3.7 \mathbb{N} -equalization

In an MG where a cycle c is largely faster than the critical cycle, any ASAP execution will lead to a situation where a place of c will keep every token at least two instants. In consequence, the behavior of this place is exactly the same as two places in sequence with a dummy transition in-between. The \mathbb{N} -equalization performs this transformation wherever it is required. The MG in Figure 1-c is the \mathbb{N} -equalized version of the MG in Figure 1-b.

The resulting \mathbb{N} -equalized MG has the same behavior as the original one but the throughput of c has changed. It has been reduced to approach the critical one but cannot become less. It may append that some non-critical cycles can become critical and the value of k and p can change but the ratio k/p remains constant. The major expected change is that for every places in the resulting MG, the number of delays over a period becomes bounded by k . More details about \mathbb{N} -equalization is available in [11, 12].

Definition 26 (\mathbb{N} -equalized MG). An MG G is said \mathbb{N} -equalized if and only if every transition belonging to a strongly connected component of G belongs to a cycle c such that:

$$M(c)/(L(c) + 1) < \text{throughput}(G) \leq \text{throughput}(c)$$

Lemma 27 (Delay in a \mathbb{N} -equalized MG). Let G be a \mathbb{N} -equalized MG. Let p be a place of G . Let D be a spatial repartition of delays. $0 \leq D(p) < k$ holds (where k is the periodicity of G).

Proof. For all places p in G , there is a cycle c such that $\sum_{p \in c} D(p) = M(c) * p - L(c) * k$.

Moreover, if G is \mathbb{N} -equalized,

$$M(c)/(L(c) + 1) < \text{throughput}(G) \leq \text{throughput}(c)$$

$\Leftrightarrow M(c)/(L(c) + 1) < k/p \leq M(c)/L(c)$. The two inequations hold:

- $M(c)/(L(c) + 1) < k/p \Leftrightarrow M(c) * p < (L(c) + 1) * k \Leftrightarrow M(c) * p - L(c) * k < k$.
- $k/p \leq M(c)/L(c) \Leftrightarrow M(c) * p - k * L(c) \geq 0$.

The two inequations can be merged in $0 \leq M(c) * p - L(c) * k < k \Leftrightarrow 0 \leq \sum_{p \in c} D(p) < k$. Even if all the delays of the cycle are merged in one place, $D(p) < k$. \square

The major complexity of the \mathbb{N} -equalization comes from the interleaving of cycles in the MG. The addition of an extra place on a path may increase the latency of many cycles and some of them can become slower than a critical cycle while some others still require extra places. Consequently, all the cycles have to be considered simultaneously to find the correct location of the additional places. In [11, 12], integer linear programming is used to specify all the \mathbb{N} -equalization constraints. A more elegant solution can be built based on (max,plus) algebra [6] by considering the incidence matrix of the MG and its evolution over a period.

In Figure 1, the \mathbb{N} -equalization may appear trivial because many places belong to only one cycle. The left cycle in Figure 1-b is faster than the right cycle, so an extra place can be added after the leftmost place. The critical (right) cycle has a throughput of $4/7$. The left cycle has a throughput $2/3$. The inequation $2/(3 + 1) < 4/7 \leq 2/3$ holds.

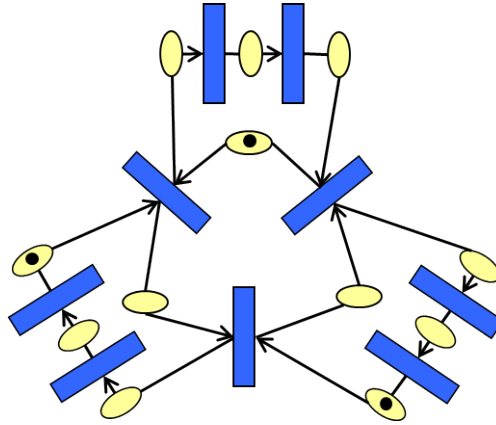


Figure 5: This MG is already \mathbb{N} -equalized.

Figure 5 presents a non-trivial example of \mathbb{N} -equalization. The outer cycle is critical with a throughput $2/9$. There is three cycles with a throughput $1/4$. Since $1/(4 + 1) < 2/9 \leq 1/4$, the \mathbb{N} -equalization condition of Definition 26 hold. The inner cycle has a throughput $1/3$ so it seems that an extra place could be added to equalize it ($1/(3 + 1) > 2/9$) but every place of this cycle also belongs to another cycle with a throughput $1/4$. Consequently, the MG is already \mathbb{N} -equalized.

As we have seen in Section 2, the \mathbb{N} -equalization of an MG is the preliminary step of the proposed algorithm.

4 Balanced binary words

This section presents the basic definitions and well-known results on balanced binary words ([9]). Up to our knowledge, Theorem 44, that presents the relation between the operation of rotation and transposition, is original. The goal of this section is to present all these results in a way that eases the comprehension of the proposed algorithm.

4.1 Finite and infinite binary words

As usual the set of binary values is noted $\mathbb{B} = \{\mathbf{0}, \mathbf{1}\}$, \mathbb{B}^* the set of finite binary words, \mathbb{B}^n the set of binary words of length n , \mathbb{B}^+ the set of non-empty finite binary words, \mathbb{B}^ω the set of infinite binary words, and ε the empty word. We note $\mathbb{B}^\infty = \mathbb{B}^* \cup \mathbb{B}^\omega$, the set of finite or infinite binary words.

For $u \in \mathbb{B}^\infty$, we note $|u|$ the length of u (with $|u| = \infty$ whenever $u \in \mathbb{B}^\omega$). Similarly we note $|u|_1$ and $|u|_0$ the number of occurrences of letters $\mathbf{1}$ and $\mathbf{0}$ in u respectively. Also, for $u \in \mathbb{B}^+$ we note $\text{slope}(u)$ the ratio $|u|_1/|u|$. $\mathbb{B}_k^p = \{u \mid u \in \mathbb{B}^p \text{ and } |u|_1 = k\}$. For $i \leq |u|$ we note $u(i)$ the i^{th} letter of u .

The lexicographic ordering on words is defined as: for $u, v \in \mathbb{B}^\infty$, $u < v$ iff $\exists i \in \mathbb{N}$, $\forall j < i$, $u(j) = v(j)$ and either $u(i) = \mathbf{0}$ and $v(i) = \mathbf{1}$ or $|u| = i - 1$ and $|v| \geq i$. This order is total. For any finite subset V of \mathbb{B}^∞ , $\text{inf}(V)$ and $\text{sup}(V)$ are respectively its lowest and highest elements for this ordering. Finally, for $u \in \mathbb{B}^*$ and $v \in \mathbb{B}^\infty$, u is a factor of v if $\exists u_1 \in \mathbb{B}^*$, $u_2 \in \mathbb{B}^\infty$ such that $v = u_1.u.u_2$.

Definition 28 (Ultimately k-periodic binary word). *An infinite binary word is called ultimately k-periodic if it is of the form $u.v^\omega$, with $u \in \mathbb{B}^*$ and $v \in \mathbb{B}^+$ with $|v|_1 = k > 0$.*

It is called simply *k-periodic* if in addition $u = \varepsilon$. It is called *ultimately periodic* if $k = 1$. It is called only *periodic* if both conditions occur. For an ultimately k-periodic word, u is called the *initial part*, v , the *steady part*, $k = |v|_1$ is the *periodicity*, and $p = |v|$ is the *period*. By definition $\text{slope}(u.v^\omega) = \text{slope}(v)$. \mathbb{P} is the set of ultimately periodic infinite binary words and \mathbb{P}_k^p is the set of such word of periodicity k and period p .

Example 29. $11.(0110101)^\omega$ is 4-periodic with period 7, and so is in \mathbb{P}_4^7 .

Because the ASAP execution of an MG is ultimately periodic, the proposed algorithm mainly focus on a single period of execution that aim to be indefinitely repeated. Thus, the following results concern finite binary words. In the proposed algorithm, for each transition t of an MG, the appropriate words v_t and u_t are found and the ultimately k-periodic word $u_t.(v_t)^\omega$ is built to represent the schedule of t .

4.2 Rotation and transposition

As we have seen in Section 2, the proposed algorithm computes the schedule of every transition of the MG. To do so, the schedule of a transition is deduced from the schedule of one of its predecessors ($\{\bullet p \mid p \in \bullet t\}$) using the transposition and rotation. In Section 4.6, we illustrate the link between the rotation and the effect of a latency on a schedule as well as the link between the transposition and the effect of a delay on a schedule.

Definition 30 (Unitary forward rotation). *The unitary forward rotation is defined as $\rho: \mathbb{B}^* \rightarrow \mathbb{B}^*$, $\rho(\varepsilon) = \varepsilon$, and $\forall u \in \mathbb{B}^*$, $\forall b \in \mathbb{B}$, $\rho(u.b) = b.u$.*

Definition 31 (Rotation). *Let $u \in \mathbb{B}_k^p$. we note $\rho^n(u)$ the n successive unitary forward rotation of u . $\rho^0(u) = u$, $\rho^1(u) = \rho(u)$, $\rho^n(u) = \rho^{n-1} \circ \rho(u)$ and, $\rho^{-n}(u) = v$ when $u = \rho^n(v)$. The parameter n is called the spin of the rotation.*

Example 32. $\rho^3(1101010) = 0101101$, $\rho^{-3}(1101010) = 1010110$ and, $\rho^p(u) = \rho^0(u) = u$

Definition 33 (Orbit). *Let $u \in \mathbb{B}^*$, the set of all rotations of u is called the orbit of u and is noted $O(u)$.*

Example 34. For $u = 0110101$, $O(u) = \{u, \rho^1(u), \dots, \rho^6(u)\} = \{0110101, 1011010, 0101101, 1010110, 0101011, 1010110\}$.

Definition 35 (Transposition). Let $u, v \in \mathbb{B}^\infty$. v is called the unitary forward transpose of u (or simply transpose for short) and noted $v = \tau(u, \Delta)$, iff $\exists u_1 \in \mathbb{B}^*$ and $\exists u_2 \in \mathbb{B}^\infty$, $u = u_1.1.0.u_2$, $v = u_1.0.1.u_2$, and $\Delta = |u_1| + 1$. Δ is called the location of the transposition. By definition, if $u = 0.u_1.1$, $\tau(u, |u|) = 1.u_1.0$ where u is finite.

Example 36. $\tau(1010101, 3) = 1001101$, $\tau(1101010, 3)$ is not defined, $\tau(011, 3) = 110$, $\tau((10101)^\omega, 3) = 10011.(10101)^\omega$ and, $(\tau(10101, 3))^\omega = (10011)^\omega$.

4.3 Balanced binary words

The proposed algorithm computes an execution where all schedules are ultimately k -periodic balanced binary words with a period p .

Definition 37 (Balanced binary word). A finite binary word $u \in \mathbb{B}^+$ is said balanced if $\forall v, t$, two factors of u^ω such that $|v| = |t|$, the following property holds: $-1 \leq |v|_1 - |t|_1 \leq 1$.

The set of finite balanced binary words with length p and containing k occurrences of $\mathbf{1}$ is denoted by \mathbb{S}_k^p . Also, $u \in \mathbb{S}_k^p$ is said primitive when k and p are mutually prime. By extension an ultimately periodic word is called *balanced* if its steady part is. We have chosen the letter \mathbb{S} for *Smooth*.

In [9], the authors prove that i) in a balanced binary word u , the number of $\mathbf{1}$ in every factor of u^ω with a length l is either $\lfloor l * |u|_1 / |u| \rfloor$ or $\lceil l * |u|_1 / |u| \rceil$, ii) all the balanced binary words with the same slope are equivalent by rotation (let $u, v \in \mathbb{S}_k^p$, $O(u) = O(v) = \mathbb{S}_k^p$), iii) $\inf(\mathbb{S}_k^p) = 0.u.1$ and $\sup(\mathbb{S}_k^p) = 1.u.0$ ($u \in \mathbb{B}^{p-2}$), and lastly iv) whenever k and p are not mutually prime, every balanced binary word in \mathbb{S}_k^p (called in this case non-primitive) is the repetition of a smaller primitive balanced binary word: let $0 < k \leq p$ and $GCD(k, p) = x$, $\forall u \in \mathbb{S}_k^p$, $\exists v \in \mathbb{S}_{k/x}^{p/x}$ such that $u = v^x$.

When the proposed algorithm meets none-primitive balanced binary word, it considers the primitive balanced binary word imprinted into it. The execution is correct because when $u = v^x$, we have $u^\omega = v^{x^\omega} = v^\omega$.

4.4 Transposition on balanced binary words

Definition 40 defines a bijective function of transposition from \mathbb{S}_k^p to \mathbb{S}_k^p . It requires some intermediate results.

Lemma 38 (Transposition in \mathbb{S}_k^p). $\forall u \in \mathbb{S}_k^p$ with k and p relatively prime, There exists a unique Δ such that $\tau(u, \Delta) \in \mathbb{S}_k^p$.

Proof. If the transposition is applied to any $\mathbf{1}$ of $\inf(\mathbb{S}_k^p)$, the transpose is a lower word which is consequently not balanced except for the last bit of $\inf(\mathbb{S}_k^p)$, in this case, the transpose is $\sup(\mathbb{S}_k^p)$. This result is consistent modulo rotation. \square

If k and p are not relatively prime, $GCD(k, p) = x$. $\forall u \in \mathbb{S}_k^p$, $u = v^x$. We define $\Delta = \Delta'$ such that Δ' is the unique location where $\tau(v, \Delta') \in \mathbb{S}_{k/x}^{p/x}$.

Lemma 38 shows that Δ is the last position of $\inf(\mathbb{S}_k^p)$. Starting from this location, Δ can be found in every word of \mathbb{S}_k^p .

Corollary 39. In $\rho^n(\inf(\mathbb{S}_k^p))$, $\Delta = p + n \equiv n \pmod{p}$.

We define the transposition function as the transposition applied on the bit Δ of a balanced binary word.

Definition 40 (The transposition function on balanced binary words).

We define the transposition function applied on balanced binary words as: $\tau^n: \mathbb{S}_k^p \rightarrow \mathbb{S}_k^p$. $\tau^0(u) = u$, $\tau(u) = \tau^1(u) = \tau(u, \Delta)$ where Δ is the same as in Lemma 38, $\tau^n = \tau^{n-1} \circ \tau$, and $\tau^{-n}(u) = v$ if and only if $\tau^n(v) = u$. If k and p are not relatively prime, $GCD(k, p) = x$. $\forall u \in \mathbb{S}_k^p$, $u = v^x$. $\tau^n(u) = (\tau^n(v))^x$.

Example 41. $\tau^1(1101010) = 1011010$, $\tau^2(1010101) = 0101101$, $\tau^p(w) = w$, and $\tau(110110) = 101101$.

Lemma 42. The function τ^n is bijective.

Proof. Since $\tau^1(\rho^n(\inf(\mathbb{S}_k^p))) = \rho^n(\sup(\mathbb{S}_k^p))$, there is a one to one correspondence between the elements and the images through the τ function. \square

4.5 Equivalence between rotation and transposition on balanced binary words

Theorem 44 presents our original result on balanced binary word. It states that for any given balanced binary word u , the transpose of u is equivalent to the rotation of u with a spin $-\alpha$. Let us first define α .

Definition 43 (The alpha coefficient). Let k, p be two relatively prime integers, $0 < k < p$. α is the inverse of $-k \pmod p$. So we have $-k * \alpha \equiv 1 \pmod p$ and α relatively prime with p .

Theorem 44. $\forall u \in \mathbb{S}_k^p$, $\rho^{-\alpha}(\tau(u)) = u$.

Proof. We are going to prove that $u = u_1.0.1.u_2$ and $\rho^\alpha(u) = u_1.1.0.u_2$ ($u_1, u_2 \in \mathbb{B}^*$). This means that u is the transpose of $\rho^\alpha(u)$. So we compare u and $\rho^\alpha(u)$ bit-wise for $i \in \llbracket 1, p \rrbracket$. $\rho^\alpha(u)(i) = u(i - \alpha) = \lfloor (i - \alpha) * k / p \rfloor - \lfloor (i - 1 - \alpha) * k / p \rfloor$.

α in $u(i - \alpha)$ is replaced by its value and the equation is simplified in:

$$u(i - \alpha) = \lfloor \frac{i * k + 1}{p} \rfloor - \lfloor \frac{(i-1) * k + 1}{p} \rfloor. \text{ Otherwise, } u(i) = \lfloor \frac{i * k}{p} \rfloor - \lfloor \frac{(i-1) * k}{p} \rfloor.$$

For $i * k \neq k - 1$ and $i * k \neq p - 1$ modulo p , $u(i - \alpha) = u(i)$ and

for $i * k = p - 1$ modulo p , $u(i - \alpha) = 1$, $u(i) = 0$, moreover,

$(i + 1) * k = p - 1 + k = k + 1$ modulo p , and $u(i + 1 - \alpha) = 0$, $u(i + 1) = 1$

\square

The proposed algorithm computes the schedules of the transitions from the schedules of its parent transitions. These schedules are equivalent by rotation because they are all balanced. Thanks to Theorem 44, the rotation is used instead of transposition in the schedule computation formulas. This simplification lightens the formulas and allows correctness checking of the proposed algorithm.

4.6 From word to schedule

The unitary forward rotation represents the effect of a latency on a transition schedule while the unitary forward transposition represents the effect of a delay. Figure 6 focuses on two transitions of a 4-periodic MG with a period 7. The schedules of A and B are binary words with length 7 containing 4 bits with the value 1. In Figure 6-a, the schedule of B is the unitary rotation of the schedule of A because no delay is affected to the place in-between. The arrows illustrate

this rotation ($B(i + 1) = A(i), \forall i \pmod{7}$). In Figure 6-b, two delays are affected to the place in-between. B does not compute all the tokens generated by A as soon as they are available any more. Two of them are delayed. The schedule of B is the double transposition of the rotation of the schedule of A . The first arrow in diagonal illustrates the rotation, the two next, the transpositions. In Figure 6-c, thanks to Theorem 44, the succession of operations presented in Figure 6-b is replaced by the equivalent rotation of value: $1 - 2 * \alpha$ where 1 is the original rotation, $-2 * \alpha$ represents the two transpositions. For $(k, p) = (4, 7)$, we have $\alpha = 5$ (Definition 43). So the spin of the rotation is 5 ($1 - 2 * \alpha \equiv 1 - 2 * 5 \equiv -9 \equiv 5 \pmod{7}$).

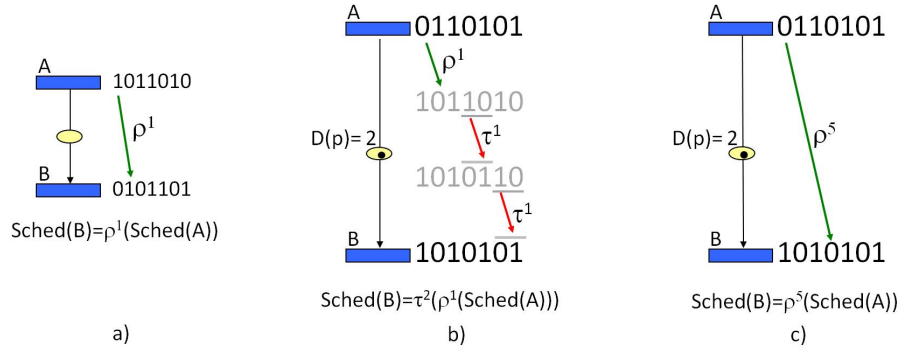


Figure 6: a) The schedule of B is the unitary rotation of the schedule of A . b) The schedule of B is the double transposition of the rotation of the schedule of A because the place in between is a 2-delays place. c) Thanks to Theorem 44, the schedule of B in b) is the rotation of spin 5 of the schedule of A .

5 Balanced scheduling of MG

This section details the proposed algorithm that computes an execution which is characterized by the following properties: i) the execution rate is maximal, ii) place sizes are minimal, and iii) after a guided initialization, the execution is ASAP.

Input: the proposed algorithm, presented in Algorithm 1, takes as input a live and strongly connected MG with a throughput inferior or equals to 1. Section 5.3 discusses the application of the proposed algorithm on a simply connected MG.

Output: Algorithm 1 returns the computed execution along with the size of the places required for this execution.

The following notation are used in Algorithm 1:

- G is the MG in input and M_0 is its initial marking.
- D is the latest delays position (Definition 22).
- $Exec_{initial}$ is the initial guided execution of G from its initial marking to $M_{periodic}$.
- $M_{periodic}$ is the marking of G from which $Exec_{periodic}$ starts.
- $Exec_{periodic}$ is an balanced ASAP execution of G from the marking $M_{periodic}$.
- $Sched(t)$ is the schedule of the transition t in $Exec_{periodic}$.

- The execution $Exec = Exec_{initial}.Exec_{periodic}$ is the output of the proposed algorithm.
- C_{Exec} gives place sizes according to $Exec$ (Definition 14).

We consider that the preliminary step of the proposed algorithm is the \mathbb{N} -equalization of the MG followed by the expansion of its latencies. \mathbb{N} -equalization is discussed in Section 3.7 and expansion of latencies is discussed in Section 3.6.

Algorithm 1 The proposed algorithm

Input : G with its initial marking M_0 .
Output : The execution $Exec$ and the place sizes C_{Exec} .

1. $(k, p) \leftarrow compute_k_p(G)$
2. $D \leftarrow compute_D(G)$
3. $Exec_{periodic} \leftarrow compute_Exec_{periodic}(G, D, k, p)$
4. $M_{periodic} \leftarrow compute_M_{periodic}(G, Exec_{periodic}, p)$
5. $Exec_{initial} \leftarrow compute_Exec_{initial}(G, M_0, M_{periodic})$
6. $Exec \leftarrow Exec_{initial}.Exec_{periodic}$
7. $C_{Exec} \leftarrow compute_C_{Exec}(G, D, k, p)$.

return $(Exec, C_{Exec})$

5.1 Algorithm details

5.1.1 Step 1: compute k and p

The formula is given in [6]. $k = GCD(M_0(c))$ and $p = GCD(L(c))$, for all cycle c of the CSCCs. Step 1 requires the enumeration of all the elementary cycles. This enumeration has an exponential complexity with respect to the number of transitions. It binds the overall complexity of the proposed algorithm.

5.1.2 Step 2: compute the latest delays position D

D has to be the latest delays position (Definition 22) in order to build the ASAP execution $Exec_{periodic}$. Theorem 23 shows that the latest delays position can be deduced from any ASAP execution of G . Thus, Step 2 computes D from the ASAP execution of G . Step 2 has a polynomial complexity according to the number of transitions. Algorithm 2 details Step 2.

Figure 7 presents D on the running example. The right-most cycle, c_1 , is critical, it does not contain any delay. The left-most cycle, c_2 , is not. The difference of firing over a period is $|c_1| * |c_2|_1 - |c_1|_1 * |c_2| = 7 * 2 - 4 * 3 = 2$. The places of c_2 that do not belong to c_1 should share 2 delays. The left-most and top-most place contains all these delays because in the latest delays position, the delays have to occur as late as possible.

5.1.3 Steps 3: compute $Exec_{periodic}$

Step 3 affects a schedule to every transition with respect to D . Algorithm 3 details Step 3. It has a linear complexity according to the number of transitions.

In Figure 8, Step 3 generates a balanced binary word $1101010 \in \mathbb{S}_4^7$ because the MG is 4 periodic with a period 7. Step 3 affects this word to a transition and it computes the schedule of the other transitions using the rotation. The schedule of the 2-inputs transition (1010101) can be found from its right predecessor $\rho^1(0101011)$ or from its left predecessor $\rho^5(0110101)$. The

Algorithm 2 *compute_D*

Input : G .
Output : D .
 Run the ASAP execution of G .
for all $p \in P$ **do**
 $D(p) = \sum_{i=1}^p \text{Delay}(p, j_0 + i)$ where j_0 is the length of the initial part.
end for
while D is not the latest delay position **do**
for all $t \in T$ **do**
 $\text{forwarded_delay} = \min(D(p) \mid \forall p \in \bullet t)$
 for all $p \in \bullet t$ **do**
 $D(p)- = \text{forwarded_delay}$
 end for
 for all $p \in t \bullet$ **do**
 $D(p)+ = \text{forwarded_delay}$
 end for
end for
end while
return D

Algorithm 3 *compute_Execperiodic*

Input : G , D , k , and p .
Output : $\text{Exec}_{\text{periodic}}$.
 Let $t \in T$, $\text{Sched}(t) \leftarrow \text{get_a_word_in}(\mathbb{S}_{k/r}^{p/t})$ {with $r = \text{GCD}(k, p)$.}
 $\text{current_transition} \leftarrow t$
while $\exists t' \in T$ such that $\text{Sched}(t')$ is not defined **do**
for all $t' \in \{(\text{current_transition} \bullet) \bullet\}$ **do**
 $\text{Sched}(t') \leftarrow \rho^{1-D(\bullet t') * \alpha}(\text{Sched}(\text{current_transition}))$
end for
 $\text{current_transition} \leftarrow t'$
end while
return $\text{Exec}_{\text{periodic}}$

spin of this last rotation is $5 \equiv 1 - 2 * \alpha \pmod{p}$. The place in-between the transitions contains 2 delays. Since $\alpha = 5$, $1 - 2 * \alpha = 1 - 2 * 5 = -9 \equiv 5 \pmod{7}$.

The consistency of this method is guaranteed because the number of delay for each cycle is conformed to Theorem 20. The lemma 45 formalizes this result.

Lemma 45 (Creation of $\text{Exec}_{\text{periodic}}$). *Step 3 is consistent.*

Proof. Let $u \in \mathbb{S}_k^p$ be a balanced binary word. The number of delays occurring on a cycle c during a period of execution is $n = M_0(c) * p - L(c) * k$. The latency on this same cycle is $L(c)$.

If we impose the schedule of a transition t on c to $\text{Sched}(t) = u$ and we propagate this schedule to the successors according to Step 3, then t will be ultimately reached again. The updated schedule of the t will be $\rho^{L(c) - n * \alpha} u$. We know from Definition 43 that $\alpha * k \equiv -1 \pmod{p}$ so if we focus on the quantity $L(c) - n * \alpha$:

$$\begin{aligned} L(c) - n * \alpha &= L(c) - \alpha * (M_0(c) * p - L(c) * k) \equiv L(c) - \alpha * M_0(c) * p + \alpha * L(c) * k \\ &\equiv L(c) - \alpha * M_0(c) * p - L(c) \pmod{p} \equiv -\alpha * M_0(c) * p \pmod{p} \equiv 0 \pmod{p}, \end{aligned}$$

it is equivalent to 0 modulo p .

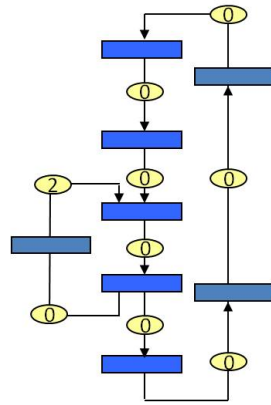


Figure 7: D : The amount of delay is written within the place.

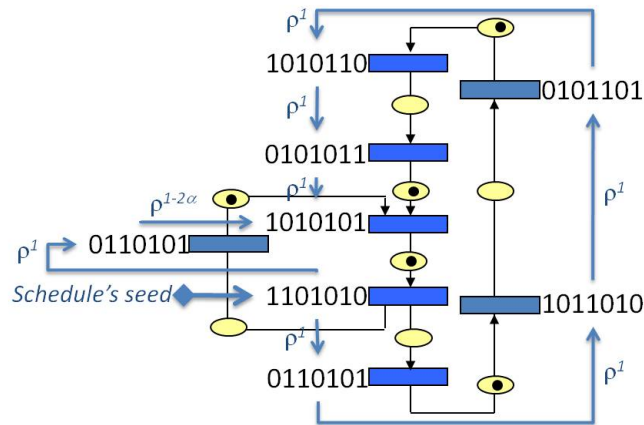


Figure 8: From the schedule's seed, Step 3 generates all other schedules through rotation. The schedule of the 2-inputs transition can be found from both its predecessor.

Consequently, the schedule of t remains the same, the method is consistent. \square

5.1.4 Step 4: compute $M_{periodic}$

Step 4 deduces $M_{periodic}$ from $Exec_{periodic}$. $M_{periodic}$ is not only the marking from which $Exec_{periodic}$ runs but also the marking generated by $Exec_{periodic}$ after a period of execution. Consequently, the last step of a period reaches $M_{periodic}$. The last bit of $Sched(\bullet p)$ represents the activity of $\bullet p$ at the last instant of the period. If it has been active, it has produced a token in p . Algorithm 4 details Step 4. It has a linear complexity according to the number of places.

Algorithm 4 *compute_* $M_{periodic}$

Input : G , $Exec_{periodic}$, and \mathbf{p} .
Output : $M_{periodic}$.
 $p \in P$, $Sched(\bullet p) = u^\omega$ and $Sched(p^\bullet) = v^\omega$
for all $p \in P$ **do**
 $M_{periodic}(p) \leftarrow u(\mathbf{p}) + [\rho(u) < v]$
 $\{[\rho(u) < v] = 1 \text{ if } \rho(u) < v \text{ and } 0 \text{ otherwise.}\}$
end for
return $M_{periodic}$

$[\rho(u) < v] = 1$ means that one token is being delayed in the place at the current instant. $[\rho(u) < v]$ is always equal to 0 when $D(p) = 0$ because $v = \rho(u)$. When $D(p) > 0$, v is the transpose of $\rho(u)$. In the usual case, $\rho(u) > v$ because transposition shifts 1s to the right. But when the transposition occurs on the last bit of the word, the transpose gets a bit on its first position and becomes higher than the original word. Thus, if a transposition occurs on the last bit, it means that a token is currently delayed in the place. Lemma 46 formalizes this intuition.

Lemma 46 (Presence of tokens in delayed places). *Let p be a place of G such that $D(p) = n > 0$. Let $u = Sched(\bullet p)$ and $v = Sched(p^\bullet)$. If $\rho(u) < v$, p is delaying a token in the marking $M_{periodic}$.*

Proof. $v = \rho^{1-n*\alpha}(u) = \tau^n(\rho(u))$. By definition, the transpose of a word is lower than the original word except when the last bit is transposed. In this last case, the transpose is higher than the original word. If, $v > \rho(u)$ (but $v = \tau^n(\rho(u))$), at least one of the transpositions occurs on the last bit. The interpretation of this statement is that the firing of p^\bullet was supposed to occur at the last instant of the period but has been delayed to the next one. The token related to this execution is currently in p . \square

Figure 9 illustrates Step 4. The last bit of the schedule of a transition determines whether a token is present in its output place(s). The place with delays contains a regular token because the schedule of the predecessor finishes by 1 but it does not contain an extra token because $1010101 < \rho(0110101) = 1011010$.

The correctness of Step 4 is presented in Section 5.2. First, Lemma 51 proves that the marking $M_{periodic}$ is reachable from M_0 . Then, Theorem 56 shows that the ASAP execution of G from $M_{periodic}$ is $Exec_{periodic}$.

5.1.5 Step 5: compute $Exec_{initial}$

Algorithm 5 computes $Exec_{initial}$ based on integer linear programming solving. The optimization criterion is the minimization of the number of firing because one cannot express linearly the

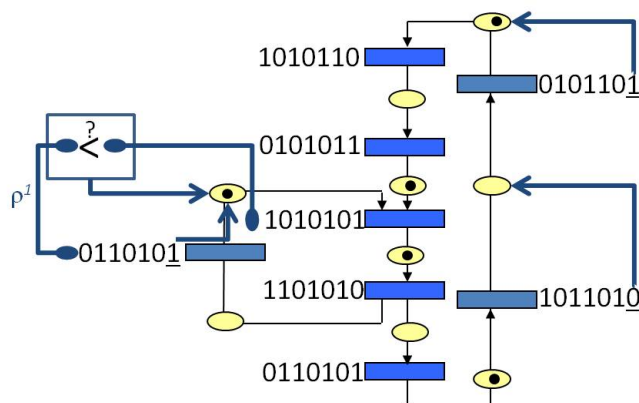


Figure 9: The step 4 generates $M_{periodic}$ from $Exec_{periodic}$. The presence of an additional token in the delayed place is found using the function $[v > \rho(u)]$.

minimization of the number of steps required to run $Exec_{initial}$. The mapping F_{init} associates to each transition the number of firing required to reach $M_{periodic}$. The function $build_execution$ builds $Exec_{initial}$ by simulating an ASAP execution of G where each transition t cannot be fired more than $F_{init}(t)$. The complexity of Step 5 depends upon the algorithm used to solve the linear system of inequation. Lemma 47 shows the correctness of Step 5.

Algorithm 5 *compute_Exec_initial*

Input : G , M_0 , and $M_{periodic}$.

Output : $Exec_{initial}$.

$Cst = \{Cst \text{ is the set of linear constraints}\}$

for all $t \in T$ **do**

$Cst+ = \{F_{init}(t) \geq 0\}$

for all $p \in t^\bullet$ **do**

$Cst+ = \{F_{init}(\bullet p) = F_{init}(p^\bullet) + M_{periodic}(p) - M_0(p)\}$

end for

end for

$F_{init} \leftarrow lp_solve(Cst, Min(\sum_{t \in T} F_{init}(t)))$

$Exec_{initial} \leftarrow build_execution(F_{init})$

return $Exec_{initial}$

In Figure 3, M_0 is on the left. The 2-bits-length schedules attached to each transition is $Exec_{initial}$ leading to $M_{periodic}$ on the right.

Lemma 47 (Correctness of Step 5). *Algorithm 5 computes a valid execution $Exec_{initial}$ reaching $M_{periodic}$.*

Proof. Let us call M_1 the marking at the end of $Exec_{initial}$. $\forall p \in P$, $M_1(p) = M_0(p) - F_{init}(p^\bullet) + F_{init}(\bullet p) = M_0(p) - F_{init}(p^\bullet) + F_{init}(p^\bullet) + M_{periodic}(p) - M_0(p) = M_{periodic}(p)$. \square

According to [18], the maximum number of firings between two markings (M_0 and $M_{periodic}$ in our case) is in $O(n^3)$ where n is the number of transitions in the MG. We assume that the length of $Exec_{initial}$ is convenient because: i) the bound $O(n^3)$ is given in terms of number of

firings. $Exec_{initial}$ allows parallel firing of transitions. ii) the periodic execution $Exec_{periodic}$ covers a set of \mathbf{p} markings. The initial part can reach any of these marking. So the problem is equivalent to: reaching the closest marking of $Exec_{periodic}$ instead of only $M_{periodic}$. iii) the cases where the upper bound is reached are extreme cases where all tokens have to shift to another place far from the initial one or because the shift of one token implies the shift of all others. In $M_{periodic}$, the tokens are "spread equally" in the MG. $M_{periodic}$ might be the easiest reachable marking.

5.1.6 Step 6: compute $Exec$

$Exec$ is composed of $Exec_{initial}$ followed by $Exec_{periodic}$. After the guided initialization, the execution is ASAP and repetitive. In Figure 4, the MG is in its initial marking. The execution, $Exec$, is represented by the ultimately k -periodic schedules attached to each transition.

5.1.7 Step 7: compute C_{Exec}

If a place does not contain delay, every token reaching the place leaves it at the next instant. As long as a place contains at most one token in M_0 , its size is 1. Lemma 48 demonstrates that if a place contains delays, tokens are never delayed more that one consecutive instant because the MG is \mathbb{N} -equalized and the schedules are balanced. In consequence a place cannot accumulate more than two tokens.

Lemma 48 (Delayed place size is bounded by 2). *According to $Exec$, place size where delays occur is bounded by 2.*

Proof. First, G is \mathbb{N} -equalized, so the number of delay per place is bounded by k . Secondly, since the execution is balanced, a token can be delayed only once in a row. Lastly, since the execution is k -periodic, there is (at most) k different tokens to delay. These conditions guarantee that a token cannot stay more than 2 instants in the place. Consequently, no accumulation of more than 2 tokens can occur. \square

Even for delayed places, a size of two is required only if a token is delayed while another reaches the place. Theorem 49 shows that a delayed place has a size of one when $D(p) < \mathbf{p} - k$ because delays occur first on the $\mathbf{1}$ which are followed by a $\mathbf{0}$. In Figure 7, all the places have a size of 1. In the delayed place p , $D(p) = 2 < 7 - 4$.

Theorem 49 (Exact delayed place size). *Let p be a place,*

$$C_{Exec}(p) = 1 \Leftrightarrow D(p) \leq \mathbf{p} - k$$

Proof. First, if a place p with $D(p) = n$ has a size one, every other place p' with $D(p') \leq n$ also has a size one. If a place p with $D(p) = m$ has a size two, every other place p' with $D(p') \geq m$ also has a size two. This property is guaranteed by the Lemma 38. In two different delayed places, the delayed tokens are the same modulo rotation. So the problem of calibrating the size of a place only depends upon the amount of delays in that place and not at all about the location of these delays.

Let $u = Sched(\bullet p)$ and $v = Sched(p\bullet)$. A place p requires a size two when a token is used after the next one has reached the place. Formally, there exists n such that $[v]_n > [u]_n$ (where $[u]_n$ is the position of the n^{th} $\mathbf{1}$ in u). v says when the current token is used, u says when a new token reaches p .

Let us assume that $D(p) = \mathbf{p} - k$. We have $v = \rho^{1-(\mathbf{p}-k)*\alpha} u = \rho^{1-\mathbf{p}*\alpha+k*\alpha} u = u$ so $[u]_n > [u]_n$ never holds.

Let us assume that $D(p) = p - k + 1$. We have $v = \rho^{1-(p-k+1)*\alpha}u = \rho^{1-p*\alpha+k*\alpha-\alpha}u = \tau(u)$ so $[\tau(u)]_n > [u]_n$ holds when n is the index of the delayed token. \square

The following theorem proves that the proposed algorithm computes an execution which has a minimal size of the places as claimed earlier.

Theorem 50 (Minimal size of the places). C_{Exec} gives the minimal size of places.

Proof. When $D(p) \leq p - k$, $C_{Exec}(p) = 1$ so it is minimal.

Let us now assume an ASAP execution $Exec'$ from the marking M' reachable from M_0 . Let assume a place p' such that $D(p') = p - k + 1$. At most $p - k$ tokens within a period can be delayed while no token follows. It remains at least 1 token that has to be delayed but that is followed by another token. In this last configuration, p contains two tokens and thus the size of p is at least 2. Consequently, $C_{Exec}(p)$ is also minimal when $D(p') > p - k$. \square

5.2 Correctness of the step 4

Let us first prove the reachability of $M_{periodic}$ from M_0 then we prove that the ASAP execution from $M_{periodic}$ is $Exec_{periodic}$.

5.2.1 Reachability of $M_{periodic}$ from M_0

Lemma 51 (Reachability of $M_{periodic}$ from M_0). $M_{periodic}$, as computed in the step 4, is reachable from M_0 .

Proof. According to [18], both markings are mutually reachable if and only if for each cycle of the MG, the two markings have the same number of tokens. Now, let us prove that $M_{periodic}$ and M_0 respect this condition.

First, Lemma 53 considers that all the delays of a cycle are assembled in the same place and proves that the condition holds. Lemma 53 requires the Lemma 52. Then Lemma 54 generalizes Lemma 53 to any allocation of delays in a cycle. \square

If all the delays are assembled in the same place p , $M_{periodic}(c)$ is equals to the number of 1s in the suffix of length $L(c)$ of $Sched(p^\bullet)$ because the schedules are, in such a case, elementary rotations of the previous ones and the bit of index p says whether a token is there in the output place. We have seen in Section 4.3 that the number of 1s in a factor of a balanced binary word of length $L(c)$ is either $\lfloor L(c) * |u|_1 / |u| \rfloor$ or $\lceil L(c) * |u|_1 / |u| \rceil$. Lemma 52 proves that if the suffix of length $L(c)$ has $\lfloor L(c) * |u|_1 / |u| \rfloor$ 1s, p is currently delaying a token. Otherwise, p is not. Consequently, the number of tokens in c is always $\lceil L(c) * |u|_1 / |u| \rceil$. Lemma 53 concludes that if the MG is equalized, $M_0(c) = \lceil L(c) * |u|_1 / |u| \rceil$ also.

Lemma 52 (Suffixes and lexicographic order in \mathbb{S}_k^p). Let $u \in \mathbb{S}_k^p$ and $j, l \in \mathbb{N}$ such that $0 < j \leq l$ and $k > j * p - k * l \geq 0$. We note $n = j * p - k * l$.

There exists n balanced binary words $v \in O(u)$ such that $|suffix(v, l)|_1 = \lfloor l * k / p \rfloor$ ($suffix(v, l)$ is the suffix of v of length l). Moreover, these n words are the highest according to the lexicographic order.

Proof. Consider the word u^l . By definition $slope(u^l) = slope(u) = k/p$. u^l can be sliced in p factors of length l . Each factor is different from the others and matches with a suffix of length l of $v \in O(u)$. If the number of factors containing $\lfloor l * k / p \rfloor$ 1s is different from n , $slope(u^l)$ cannot be k/p .

Moreover, if $|suffix(v, l)|_1 = \lfloor l * k / p \rfloor$, $|prefix(v, p - l)|_1 = k - \lfloor l * k / p \rfloor$. So if $|suffix(v, l)|_1 = \lceil l * k / p \rceil$, $|prefix(v, p - l)|_1 = k - \lceil l * k / p \rceil$. A word with more **1**s in its prefix is higher than another with less **1**s according to the lexicographic order. \square

Lemma 53 (Reachability of $M_{periodic}$ from M_0 in the simple case). *Let c be a cycle of G such that all the delays occurring in c are assembled in the place p . We have $M_{periodic}(c) = M_0(c)$.*

Proof. Let us call u the schedule of p^\bullet . The number of token in c is $M_{periodic}(c) = \sum_{i=0}^{L(c)-1} u(p - i) + [u > \rho^{D(p)*\alpha} u]$.

$\sum_{i=0}^{L(c)-1} u(p - i) = |suffix(u, L(c))|_1$. Since u is balanced, $\lfloor L(c) * k / p \rfloor \leq |suffix(u, L(c))|_1 \leq \lceil L(c) * k / p \rceil$.

Case 1: if $|suffix(u, L(c))|_1 = \lfloor L(c) * k / p \rfloor$, u is one of the $D(p)$ highest word of $O(u)$ (Lemma 52). Consequently, $u > \rho^{D(p)*\alpha} u$ because a rotation of α increases the value of the word according to the lexicographic order but if the highest is reached, another rotation of α gives the lowest. So $[u > \rho^{D(p)*\alpha} u] = 1$ and $M_{periodic}(c) = \lfloor L(c) * k / p \rfloor + 1 = \lceil L(c) * k / p \rceil$ (In the case $D(p) \neq 0$, p does not divide $k * L(c)$).

Case 2: if $|suffix(u, L(c))|_1 = \lceil L(c) * k / p \rceil$, u is not one of the $D(p)$ highest word of $O(u)$ (Lemma 52). Consequently, $[u > \rho^{D(p)*\alpha} u] = 0$, and $M_{periodic}(c) = \lceil L(c) * k / p \rceil$ also.

Conclusion: since G is \mathbb{N} -equalized, $M_0(c)/L(c) \geq k/p > M_0(c)/(L(c) + 1)$. So $(k * L(c) + k)/p > M_0(c) \geq k * l/p$. By definition of the \mathbb{N} -equalization, the solution always exists and is unique: $\lceil L(c) * k / p \rceil$. \square

In Lemma 53, a delay can occurs only in one place but in Lemma 54, every place can contain delays and they might be delaying a token in $M_{periodic}$. In this Lemma, we give the formula to compute $M_{periodic}$ from a place p_0 that we are going to consider as the first place of the cycle, then we prove that if a delay is shifted to the last place of the cycle, the number of tokens in the cycle will be the same. Thanks to this result, we can shift all the delays into the last place and conclude that the number of tokens found in Lemma 53 is applicable to the general case. The inertia of the shift operation on the number of tokens is proven by considering the last places of the cycle such that the first and the last of this sequence of places contain delays but none of the other in-between does. In such a case, the effect of the shift operation on the formula to compute $M_{periodic}$ can be analyzed locally.

Lemma 54 (Reachability of $M_{periodic}$ from M_0 in the general case). *For all cycle c , $M_{periodic}(c) = M_0(c)$.*

Proof. Let c be a cycle of G . The places of c are $\{p_0, p_1, \dots, p_{L(c)-1}\}$. We note u the schedule of the transition $\bullet p_0$.

$$M_{periodic}(c) = \sum_{i=0}^{L(c)-1} \left(u(p - (i - (D(p_0) + \dots + D(p_i)) * \alpha)) + [\rho^{i+1-(D(p_0)+\dots+D(p_{i+1})) * \alpha} u > \rho^{i+1-(D(p_0)+\dots+D(p_i)) * \alpha} u] \right).$$

Let i_0 be such that $\forall i \in]i_0, L(c) - 1]$, $D(p_i) = 0$ and let us focus on the few last terms of this sum such that $i_0 < i \leq L(c) - 1$ (In the worst case, $i_0 = L(c) - 2$ and only the last term of the sum is there). The following equality is going to be proved for these terms only:

$$[\rho^{i_0-(D(p_0)+\dots+D(p_{i_0})) * \alpha} u > \rho^{i_0-(D(p_0)+\dots+D(p_{i_0-1})) * \alpha} u] \quad (A)$$

$$+ \sum_{i=i_0}^{L(c)-1} u(p - (i - (D(p_0) + \dots + D(p_i)) * \alpha)) \quad (B)$$

$$+ [u > \rho^{D(p_{L(c)-1}) * \alpha} u] \quad (C)$$

=

$$[\rho^{i_0-(D(p_0)+\dots+D(p_{i_0-1})) * \alpha} u > \rho^{i_0-(D(p_0)+\dots+D(p_{i_0-1})-1) * \alpha} u] \quad (A')$$

$$\begin{aligned}
 & + \sum_{i=i_0}^{L(c)-1} u(p - (i - (D(p_0) + \dots + D(p_i) - 1) * \alpha)) \text{ (B')} \\
 & + [u > \rho^{(D(p_{L(c)-1})+1)*\alpha} u] \text{ (C')}.
 \end{aligned}$$

There is only three cases to study to prove this property:

- When (A) is equals to 1 but (A') is equals to 0, then the first term of (B) is equals to 0 and the first term of (B') is equals to 1. If the first place delays a token (A)=1 but not any more after the shift (A')=0, the token has been computed instead of being delayed and then it appears in the next place (B')=1. All the other term of the sum are the same.
- When (C) is equals to 0 but (C') is equals to 1, the last term of (B) is equals to 1 and the last term of (B') is equals to 0. If the last place does not delay any token (C)=0 but does after the shift (C')=1, this token was in the last but one place (B)=1 and is now in the last one (B')=0. All the other term of the sum are the same.
- In every other possible cases, (A) equals (A'), (B) equals (B'), (C) equals (C').

Thanks to this property, we know that the number of tokens in c is the same wherever are the delays in the cycle. So the result found in lemma 53 is applicable to the general case. \square

5.2.2 Validity of $Exec_{periodic}$ from $M_{periodic}$

Lemma 55 (A step of execution from $M_{periodic}$). *Let M_1 be the marking resulting from a step of ASAP execution from $M_{periodic}$, M'_1 is the marking resulting from a step of $Exec_{periodic}$ from $M_{periodic}$.*

Then, $M_1 = M'_1$

Proof. In an ASAP execution, a transition t executes if and only if all the incoming places contains a token. In $M_{periodic}$, the place $\bullet t$ contains a token if and only if $Sched(\bullet\bullet t)(p) = 1$ or $[\rho^1(Sched(\bullet\bullet t)) < Sched(t)]$. In the first step of $Exec_{periodic}$, a transition t executes if and only if $Sched(t)(1) = 1 \Leftrightarrow \rho^{-1}(Sched(t))(p) = 1 \Leftrightarrow Sched(\bullet\bullet t)(p) = 1$ or that $[\rho^1(Sched(\bullet\bullet t)) < Sched(t)]$. The condition of execution are the same. If the same transitions are fired according to an ASAP execution or $Exec_{periodic}$, then the resulting markings are the same. \square

Theorem 56 (Validity of $Exec_{periodic}$). *The ASAP execution of G from the marking $M_{periodic}$ is $Exec_{periodic}$.*

Proof. Step 3 is based on the affectation of a schedule by a random balanced binary word from \mathbb{S}_k^p . The lemmas 45, 51 and Lemma 55 also hold for any other balanced binary word from \mathbb{S}_k^p . Since all the words of \mathbb{S}_k^p are equivalent by rotation, Step 4 gives all the successive markings of $Exec_{periodic}$ when the Step 3 is initiated with, successively, all the words of \mathbb{S}_k^p . For each of these marking, Lemma 55 proves that the next marking is reachable through ASAP execution. Consequently, from $M_{periodic}$, and after p steps of execution, $Exec_{periodic}$ reaches $M_{periodic}$. \square

5.3 Extension to the simply connected case

As we have seen in proposition 18, one cannot guaranty that an ASAP and bounded execution exists for a given simply connected MG. Since a System-on-Chip cannot be designed with unbounded memories, the extension of the proposed algorithm to simply connected case preserves the bounded property at the expense of the ASAP property. The maximum execution rate is still preserved but the minimality of the size of places is altered.

A simply connected MG can be transformed into a strongly connected one by adding feedback paths. Thus, the proposed algorithm can be applied. To do so, we add to the MG some feedback

paths which bind all the components together. The functional behavior of the system will be preserved but its scheduling will be over-constrained by the added feedback paths i.e. adding different feedback paths imply a different execution computed by the proposed algorithm. These feedback paths act as synchronization barriers.

There is different algorithmic solution to realize the transformation; however, the added feedback paths should not create a cycle with a throughput inferior to the critical one in the original MG. Otherwise, the maximal execution rate will not be achieved. It is easy to prove that the marking and the latency of the added feedback paths can always be adjusted so that the created cycles have a non-critical throughput.

The minimality of the size of the places is guaranteed for the original SCCs, but the size of the places on the original DAC depends upon the added feedback paths. One may find another set of feedback paths such that the size of places on the original DAC is less. We have not yet studied this optimization.

Open MG If a simply connected MG is open, one can consider that the system has global input(s) and output(s). In order to schedule the MG, it is transformed in a strongly connected one. Consequently, the MG becomes closed. The run of the proposed algorithm shall return a schedule for every source and sink. The schedule of a sink says when the system produces an output token and the schedule of a source says when the system consumes an input token. Thus, the concerned input token has to be present when required. In [10], we state that the execution rates of the feeder and eater have to be the same in order to calibrate the capacity of the "interconnection" place with a finite value and thus ensure on-demand token availability. In [16], the authors study thoroughly the sizing of buffer between clocked systems.

The AES example Figure 10 presents an implementation of the AES encryption standard. The MG has been represented using K-Passa (K-Periodic Asap Static Schedule Analyser) [21]. K-Passa implements the proposed algorithm but also the \mathbb{N} -equalization. The circles represent the transitions of the system. The arrows represent the sequences ($arc \rightarrow place \rightarrow arc$) in-between two transitions. The two left most transitions called *key* and *word* are sources (the local loop has been added for simulation purpose). The central transition called *output word* is a sink. The schedule attached to each transition is the one computed by the proposed algorithm. The guided initialization has a length 1, then the behavior is 1-periodic with a period 6. Every place has a size one. The only place where one delay occurs is the one between *word* and *mux* (where a small square appears), however a size one is enough.

As one can see, the AES example is a simply connected graph. In order to run the proposed algorithm, two paths from the sink to each of the sources have been added to the system.

6 Results and discussion

This paper proposes an algorithm to statically schedule any live and strongly connected MG with a throughput inferior or equals to one. The proposed algorithm computes the balanced ASAP execution where the execution rate is maximal and place sizes are minimal. Moreover, a transformation has been proposed to change a simply connected MG in a strongly connected MG such that the proposed algorithm can be applied.

In the domain to the System-on-Chip design, the proposed algorithm is used to schedule applications which are subject to the problem of long wire latency. If we compare our approach to the latency insensitive design, this last is not as strict as our approach about the constraint on availability of data on global inputs. It is a purely dynamic solution but the cost for this

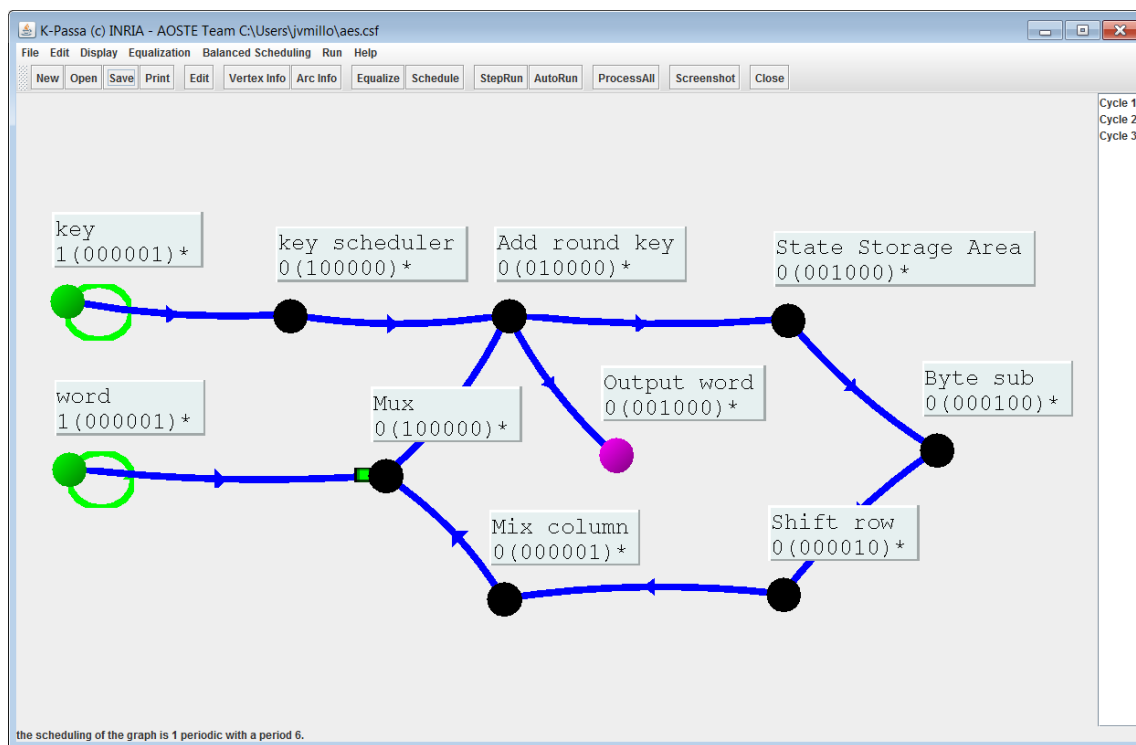


Figure 10: The MG presents an implementation of the AES encryption standard.

dynamicity is the duplication of every data path in the circuit and the replacement of every simple register by a two-sized-register to manage the dynamic communication and computation protocol. This difference makes our approach better for pure data flow system.

Acknowledgment

This work has been supported by CIMPACA/SYS2RTL. The authors would like to thanks Benoit Ferrero for his help with the proofs, Anthony Coadou for his constructive remarks, and the anonymous reviewers for their suggestions who have led us in the right direction.

References

- [1] M. Alanyali and B. Hajek. Analysis of simple algorithms for dynamic load balancing. In *Mathematics of Operations Research*, pages 230–238, 1995.
- [2] M. Alanyali and B. Hajek. On load balancing in erlang networks. *Stochastic Networks: Theory and Applications Oxford University Press*, -:215–230, 1996.
- [3] C. Allauzen. Une caractérisation simple des nombres de sturm. *Journal de la théorie des nombres de Bordeaux*, 10.2:237–241, 1998.

- [4] E. Altman, B. Gaujal, and A. Hordijk. Balanced sequences and optimal routing. *Journal of the ACM*, 47(4):752–775, 2000.
- [5] E. Hyon B. Gaujal. A new factorization of mechanical words. *INRIA/RR 5175*, 2004.
- [6] F. Baccelli, G. Cohen, G. J. Olsder, and J-P Quadrat. *Synchronization and Linearity: an algebra for discrete event systems*. John Wiley & Sons, 1992.
- [7] J Bernoulli. Recueil pour les astronomes. *A Berlin*, 1:255–284, 1772.
- [8] J. Berstel and A. Luca. Sturmian words, Lyndon words and trees. *Theoretical Computer Science*, 178:171–203, 1997.
- [9] J. Berstel and P. Séébold. *Sturmian Words. In: Lothaire, M. (Ed.): Algebraic Combinatorics on Words. Chap. 2*. Cambridge University Press, 2001.
- [10] J. Boucaron and J-V Millo. Compositionality of statically scheduled IP. *Electronic Notes in Theoretical Computer Science*, 200(1):71–87, 2008.
- [11] J Boucaron, J-V Millo, and R de Simone. Latency-insensitive design and central repetitive scheduling. In *MEMOCODE '06. Proceedings. Fourth ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2006.*, pages 175– 183, Piscataway, NJ, USA, 2006. IEEE Press.
- [12] J Boucaron, J-V Millo, and R de Simone. Formal methods for scheduling of latency-insensitive designs. *EURASIP journal on embedded system*, 2007.
- [13] J. Carlier and P. Chrétienne. *Problème d'ordonnement: modélisation, complexité, algorithmes*. Masson, Paris, 1988.
- [14] L. Carloni, K. McMillan, and A. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20(no. 9):pp. 1059–1076, 2001.
- [15] E. B. Christoffel. Observatio arithmetica. *Ann. Mat. Pura Appl*, 6:148–152, 1875.
- [16] Albert Cohen, Louis Mandel, Florence Plateau, and Marc Pouzet. Abstraction of clocks in synchronous data-flow systems. In *The Sixth ASIAN Symposium on Programming Languages and Systems (APLAS 2008)*, Bangalore, India, December 2008.
- [17] F. Commoner, A. W.Holt, S. Even, and A. Pnueli. Marked directed graphs. *Journal of Computer and System Sciences*, 5:511–523, October 1971.
- [18] J. Desel and J. Esparza. *Free choice Petri nets*. Cambridge University Press, New York, NY, USA, 1995.
- [19] E. Laurier. Opérations sur les mots de Christoffel. *Journal de la théorie des nombres de Bordeaux*, 11.1:111–132, 1999.
- [20] D. Matzke. Will physical scalability sabotage performance gains? *Computer*, 30(9):37 –39, sep 1997.
- [21] Jean-Vivien Millo. <http://www-sop.inria.fr/members/jean-vivien.millo/#tools>, February 2012.
- [22] C. Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. Cambridge, Massachusetts.: MIT, Dept. Electrical Engineering, PhD Thesis, 1974.



**RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE**

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399