QUASI-STATIC SCHEDULING OF FREE-CHOICE PETRI NETS

by

Marco Sgroi, Luciano Lavagno, and
Alberto Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M98/9

13 March 1998

QUASI-STATIC SCHEDULING OF FREE-CHOICE PETRI NETS

by

Marco Sgroi, Luciano Lavagno, and
Alberto Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M98/9

13 March 1998

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

QUASI-STATIC SCHEDULING OF FREE-CHOICE PETRI NETS

by

Marco Sgroi, Luciano Lavagno, and
Alberto Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M98/9

13 March 1998

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Quasi-Static Scheduling of Free-Choice Petri Nets

Marco Sgroi† Luciano Lavagno‡ Alberto Sangiovanni-Vincentelli†
†University of California at Berkeley,
‡Politecnico di Torino, Cadence Berkeley Labs

Abstract

Efficient scheduling of concurrent specifications is a key problem in embedded system design. Static scheduling algorithms decide the schedule at compile time, dynamic scheduling algorithms make some or all decisions at run-time. The choice of the scheduling policy mainly depends on the specification of the system. For specifications containing data-dependent control structures, like the if-then-else or while-do constructs, the dynamic behaviour cannot be fully predicted at compile time and some scheduling decisions are to be made at run-time. The quasi-static scheduling approach for Data Flow networks proposed in [7] makes most of the scheduling decisions at compile time, thus maximizing the predictability and run-time efficiency of the schedule. However, the problem of finding a schedule with bounded memory is undecidable for DF networks, and thus can be solved only in special cases. In our approach, we abstract Data Flow networks as Petri Nets (PNs), a model for which most properties are decidable, and define the quasi-static scheduling (QSS) problem for PNs. We solve QSS for a sub-class of PNs known as *free-choice* nets (FCPNs) [8, 1], by reducing it to a decomposition of the net into statically schedulable components. The proposed algorithm is complete, in that it can solve QSS for any FCPN that is statically schedulable. It also allows one to explore different schedulings, in terms of schedule and buffer size (trading off code and data size) .

1 Introduction

Efficient scheduling of concurrent specifications is a key problem in embedded system design. The problem has been historically tackled for two different classes of applications, and related specification models:

1. Data Flow (DF) specifications, in which little or no runtime decisions are required and I/O timing is regular and known in advance. For such specifications, quasi-static scheduling [7] makes most of the scheduling decisions at compile time, leaving at run-time only choices that depend on the value of data (and not on their timing). This maximizes the predictability and run-time efficiency of the schedule, and allows one to optimize and trade off smoothly code size and data memory size. However, the problem of quasi-static scheduling is decidable only for DF networks that make no run-time decisions (Static Data Flow, SDF), while real complex systems rarely fall entirely into this class.

2. Real Time (RT) specifications, in which the run-time behavior heavily depends on the relative time of occurrence of internal and external events. For such specification, the most commonly used models [6] ignore or abstract inter-process dependence, and may thus yield grossly pessimistic schedules. In this case, run-time choice between alternative behaviors is modeled only as independent tasks, thus forbidding partially static schedules.

In this paper we focus our attention on the scheduling problem for non-static Data Flow specifications, that include data-dependent control structures and therefore require some run-time scheduling decisions. In particular we address the scheduling problem for Petri Nets (PN) [11], a well-known model of computation widely used to model distributed Discrete Event Dynamic Systems. PNs are able to model concurrency, choice, synchronization and causality, and hence seem to be appropriate for embedded systems with non-static Data Flow specifications. The most significant difference between DF networks and PNs is that the former model also the *values* that are communicated by *tokens* between computational units (also called *actors*), while the latter describe only the causality, concurrency and choice relations between those units (also called *transitions*). Moreover, the semantics of communication in DF is FIFO, while PNs do not impose any order on tokens. In this paper we show that a scheduling algorithm for PNs can find a larger class of applications then the well-behaved nets [5] that are handled successfully by the algorithm for DF networks proposed in [7].

We restrict our analysis to a sub-class of PNs called *Free-Choice* (FCPNs), because they do not exhibit any confusion between the notions of concurrency and choice. Hence they seem appropriate to model computations in which the outcome of a choice depends on the value of a token (and hence is abstracted as non-deterministic in PNs), rather than on the time of arrival of a token. However, we hope that the techniques that we develop can be extended to PN classes that include modeling exceptions, such as [9].

In the following Sections we provide a definition of quasi-static schedulability for FCPNs and propose an algorithm that finds a schedule with bounded memory, if the net is schedulable.

The problem of quasi-static scheduling FCPNs, simultaneously determining tight upper bounds on code and data memory requirements, can be solved by finding a finite set of finite firing sequences that allow, when repeated one after the other, to execute a PN forever without unbounded accumulation of tokens. A Free Choice Petri Net has a valid schedule if there exists a valid firing sequence, that is a sequence returning the PN to its initial state, for every possible way to solve the non-deterministic choices.

We show that a useful tool to solve the problem of finding a valid schedule is the Marked Graph decomposition algorithm proposed by Hack [8] to check boundedness of strongly connected ordinary nets. However, in the domain of embedded reactive systems most applications have lots of interactions with the environment, that are naturally modelled as source and sink transitions. As a result, nets modelling embedded systems are usually not strongly connected. Therefore, we extend Hack's approach and use it to decompose *non-strongly connected PNs* into Conflict Free (CF) components. Then, if the net is allocatable, i.e. decomposable into as many components as the number of possible choices, we statically schedule each component separately and obtain a valid schedule.

This paper is organized as follows. Section 2 gives a general description of the Petri Net model and provides definitions of properties and subclasses. Section 3 relates Petri Nets to Dataflow Networks. Section 4 presents our strategy to find a quasi-static schedule for Free-Choice Petri Nets.

2 Petri Nets

2.1 Definition

A **Petri Net** is a triple (P, T, F), where P is a non-empty finite set of places, T a non-empty finite set of transitions and $F : (T \times P) \cup (P \times T) \to \mathbb{N}$ the weighted flow relation between transitions and places. If $F(x, y) > 0$, there is a multiple arc of weight F from x to y. If $F : (T \times P) \cup (P \times T) \to \{0, 1\}$ the net is **ordinary**. [11]

Given a node x, either a place or a transition, we define its **preset** as ${}^\bullet x = \{y | (y, x) \in F\}$ and

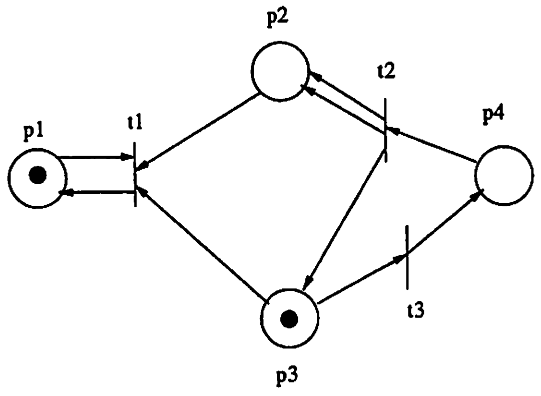


Figure 1: A Petri Net

its **postset** as $x^\bullet = \{y|(x,y) \in F\}$. $Pre[x,y]$ is equal to the weighted flow relation $F(x,y)$ between node x and node y. $Post[x,y] = F(y,x)$. A transition (place) whose preset is empty is called **source transition** (place), a transition (place) whose **postset** is empty is called **sink transition** (place). A place p such that $|p^\bullet| > 1$ is called **choice** or **conflict**. If $|^\bullet p| > 1$, p is called **attribution** or **merge**. A **marking** μ is an n-vector $\mu = (\mu_1, \mu_2, ..., \mu_n)$ where $n = |P|$ and μ_i is the non-negative number of tokens in place p_i.

A transition whose input places have enough tokens is enabled and may fire. When it fires, it removes tokens from the input places and produces tokens in the output places. The **firing rules** are the following [11]:

1. A transition t is enabled if each input place p of t is marked with at least $w(p,t)$ tokens, where $w(p,t)$ is the weight of the arc from p to t.
2. An enabled transition may or may not fire.
3. A firing of an enabled transition t removes $w(p,t)$ tokens from each input place p of t and produces $w(t,p)$ tokens to each output place p of t, where $w(t,p)$ is the weight of the arc from t to p.

In the Petri Net shown in figure 1 [10], only transition t_3 is enabled. When t_3 fires, the token in place p_3 is removed and one token is produced in place p_4 . Then, only transition t_2 is enabled. Transition firings can continue as long as there is at least one enabled transition.

2.2 Properties

The properties of Petri Nets can be distinguished in structural and behavioural [1]. Structural properties are concerned only with the structure of the graph, behavioural properties depend also on the initial marking and therefore are related to the dynamic occurrence of transitions.

The following PN properties are relevant in our discussion:

- Reachability. A marking μ' is reachable from a marking μ if there exists a firing sequence σ starting at state μ and finishing at μ'.
- Safeness. A place is safe if during any possible execution the number of tokens in that place never exceeds one. A Petri Net is safe if all the places in the net are safe. It is possible to force a Petri Net to be safe by adding acknowledgment arcs.

- Boundedness. A Petri Net is said to be k-bounded if the number of tokens in every place of a rechable marking does not exceed a finite number k. A safe Petri Net is said to be 1-bounded. If places of Petri Nets are used to represent buffers, in a bounded net buffer overflow doesn't occur whatever firing sequence is taken.
- Deadlock-freedom. A Petri Net is deadlock-free if, no matter what marking has been reached, it is possible to fire at least one transition of the net.
- Liveness. A Petri Net is live if for every reachable marking and every transition t it is possible to reach a marking that enables t.
- Coverability. A marking μ in a Petri Net is said to be coverable if there exists a marking μ' in $R(\mu_0)$ such that $\mu'(p) \geq \mu(p)$ for each p in the net.

All such properties are decidable for any Petri Net.

2.3 Analysis techniques

The most important analysis tools are the coverability tree and the incidence matrix [11]. In this section we introduce the incidence matrix. We define two matrices D^+ and D^- as follows: $D^-[j,i] = w(i,j)$ contains the number of tokens that transition j consumes from place i and $D^+[j,i] = w(j,i)$ contains the number of tokens that transition j produces at place i. Every matrix has dimension $m \times n$, where m is the number of transitions and n the number of places.

Therefore transition j is enabled if $\mu \geq e[j] \cdot D^-$, where $e[j] = (0,0,..0,1,0,....)$ is the unit vector, where only the j-th component equal to 1. Firing transition t_j from marking μ we reach marking $\mu' = \mu - e[j] \cdot D^- + e[j] \cdot D^+ = \mu + e[j] \cdot D$. The matrix $D = D^+ - D^-$ is called incidence matrix.

The D matrices for the net shown in figure 1 are the following:

$$D^- = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$D^+ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & -1 & -1 & 0 \\ 0 & 2 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

For a sequence of transition firings $\sigma = t_1 t_2 t_3$ ... the state equations are

$$\mu' = \mu + f(\sigma) \cdot D$$

where f(σ) is called the firing vector whose j-th component is the number of times that transition t_j fires in sequence σ.

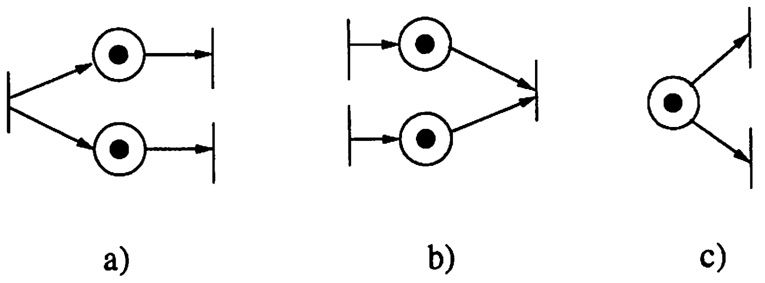


Figure 2: Concurrency (a), Synchronization (b) and Conflict (c)

2.4 Subclasses of Petri Nets

When modeling dynamic systems, it is often necessary to use a restricted class of Petri Nets, where some constraints are imposed on the graph. In this way the analysis of the system is simplified at the expense of a reduction in the expressive power of the model. Some of the most common subclasses of Petri Nets are:

- State Machine: a Petri Net such that each transition t has exactly one input place and exactly one output place.
$$\forall t \in T, |{}^{\bullet}t| = |t^{\bullet}| = 1.$$
- Marked Graph: a Petri Net such that each place p has exactly one input transition and exactly one output transition.
$$\forall p \in P, |{}^{\bullet}p| = |p^{\bullet}| = 1.$$
- Conflict Free Net: a Petri Net such that each place p has at most one output transition.
$$\forall p \in P, |p^{\bullet}| \leq 1.$$
- Free Choice Net: a Petri Net such that every arc from a place is either a unique outgoing arc or a unique incoming arc to a transition.
$$\forall p_1, p_2 \in P, p_1^{\bullet} \cap p_2^{\bullet} \neq 0 \Rightarrow |p_1^{\bullet}| = |p_2^{\bullet}| = 1.$$
- Extended Free Choice Net: a Petri Net such that any two transitions sharing some predecessor places have exactly the same set of predecessor places.
$$p_1^{\bullet} \cap p_2^{\bullet} \neq 0 \Rightarrow |p_1^{\bullet}| = |p_2^{\bullet}| \; \forall p_1, p_2 \in P.$$

Marked Graphs can represent concurrency (fig. 2a) and synchronization (fig. 2b) but not conflict (fig. 2c); on the contrary, State Machines can represent conflict but not synchronization. Free Choice Nets allow one to model both conflict and synchronization, under the condition that every transition that is successor of a choice has exactly one predecessor place. This implies that whenever an output transition of a place is enabled, all the output transitions of that place are enabled. Therefore the choice is "local" and, since it does not depend on the rest of the system, is said to be **free**. The Petri Net shown in figure 3b) is not a Free Choice Net, because transition t_3 is enabled and transition t_2 is not. Extended Free-Choice Petri Nets (EFCPN) allow a more general structure: conflicting transitions can have more than one predecessor places as long as their preset is the same. An EFCPN is shown in figure 3a).

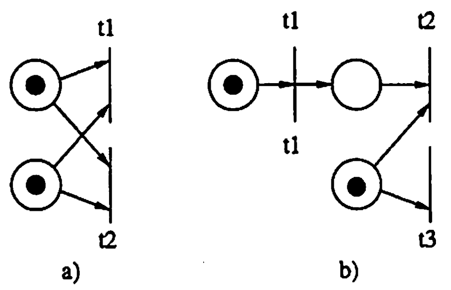


Figure 3: Extended Free Choice Net(a),not Free Choice Net(b)

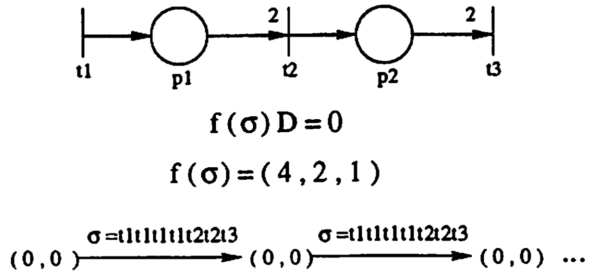


Figure 4: Cyclic schedule

Definition 2.4.1 *Two transitions t and t' are said to be in* **Equal Conflict Relation** *if* $Pre[P,t] = Pre[P,t']$.

This is an equivalence relation that partitions the set of transitions of the net into a set of equivalence classes called Equal Conflict Sets [4].

3 Dataflow and Petri Net

3.1 Cyclic schedules and reachability

Given a Petri Net and an initial marking, a **finite complete cycle** is a sequence of transition firings that returns the net to its initial state. Since both the transition firings in a finite complete cycle and the tokens produced by each firing are finite, the number of tokens that can accumulate in any place of the net during the execution is bounded. Therefore, if such a finite complete cycle exists, the net can be executed forever with bounded memory by repeating infinitely many times this sequence of transition firings (figure 4). [7]

The problem of finding a finite complete cycle for a net can be reduced to the well-known reachability problem, that is known to be decidable [11]. In terms of reachability the goal is to find a sequence of transitions σ that starting from the marking μ returns the net to the same marking μ. In this case, where $\mu' = \mu$, the state equations become $f(\sigma) \cdot D = 0$ and the solution $f(\sigma)$, when it exists, is called **T-invariant** [1].

Definition 3.1.1 *A Petri Net is consistent iff* $\exists T > 0$ *s.t.* $T \cdot D = 0$.

The existence of a solution of the state equations is a necessary, but not sufficient condition for μ' to be reachable from μ. In fact, even if there exists a solution of the state equations, the net may deadlock during execution because there are not enough tokens to fire any transition. An example

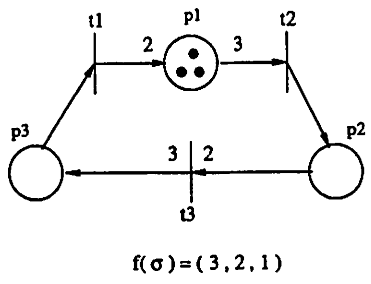


Figure 5: Simulation detects deadlock

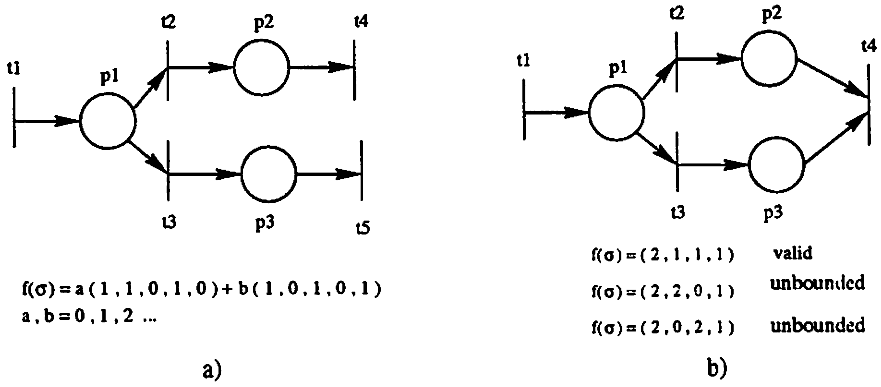


Figure 6: Schedulable (a) and not schedulable (b) EFCPNs

is given in figure 5 [12]. After the three tokens in place p_1 are consumed by transition t_2, there are not enough tokens to enable any transition. Therefore, once a firing vector $f(\sigma)$ is obtained as a solution of the state equations, it is necessary to verify by simulation that there exists a **valid firing sequence**, i.e. a sequence that contains transition t_j as many times as $f_j(\sigma)$ and such that the net does not deadlock during execution.

3.2 SDF and Petri Nets

Synchronous Dataflow (SDF) graphs are a special case of Petri Nets. SDFs can be mapped into Marked Graphs where actors become transitions and arcs places. The approach proposed by Lee [2] to find a static schedule for a SDF graph is the following. The first step consists of solving the state equations and obtaining, when the graph is consistent, the set of T-invariants, that in SDF define a one-dimensional space. Lee showed that, to find a firing sequence that returns the net to the initial state without occurrence of deadlock, it is sufficient to simulate the firing sequence that corresponds to the minimal T-invariant. (A T-invariant is **minimal** when its set of non-zero entries is not a strict superset of that of any other T-invariant and the greatest common divisor of its elements is one [4].) This approach can be adopted for Marked Graphs, but is not adequate for larger classes of Petri Nets that present not only concurrency but also conflict. In fact, if a Petri Net contains non-deterministic choices, the basic assumption of the existance of only one linearly independent T-invariant is no longer valid, as shown in the example in figure 6a. In general, when the null space of the incidence matrix is not one-dimensional, it is not possible to identify a priori what T-invariants are to be simulated to check if the net eventually deadlocks.

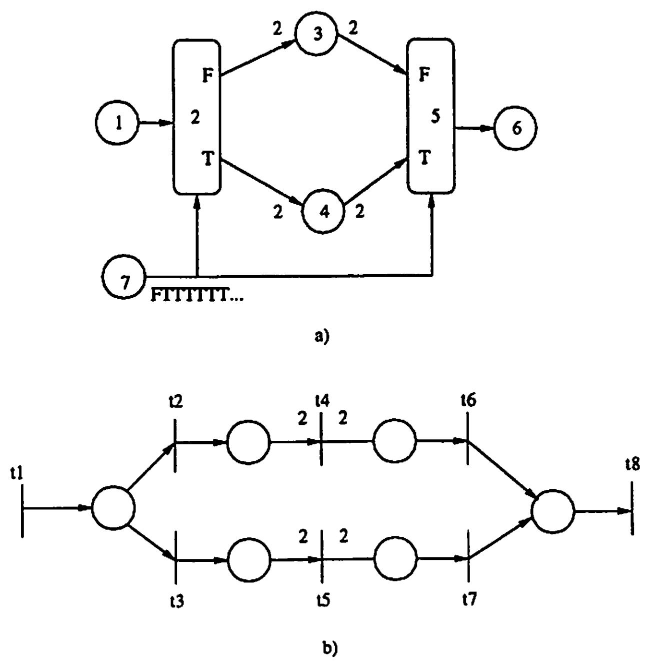


Figure 7: Modified If-then-else

3.3 BDF and Petri Nets

Petri Nets and Boolean Dataflow (BDF) [7] are both non-static Data Flow models, but their expressive power is different. One difference is in the semantics of the communication channels among blocks. Channels in BDF are FIFO queues that preserve the order of the tokens. Instead, Petri Nets do not have FIFO semantics because the tokens do not carry values [7]. This implies that a specification where the order of tokens does not matter can be scheduled if modelled with Petri Nets, but it is not schedulable in BDF. In this case BDF is simply an inappropriate model because it overspecifies the system.

Another difference is that BDF is a determinate model, since all the valid executions of the network produce the same streams regardless of the order in which the actors are executed [7], while Petri Nets are not determinate because of the non-determinism of choice and merge structures. However, it is possible to make a Petri Net determinate by imposing conditions to the schedule.

If we consider the modified if-then-else construct described in figure 7a, there exists no bounded schedule [5] [7], although there is a solution of the state equations independent of the boolean values (the network is strongly consistent). In fact, when actor 7 produces a single false token followed by an infinite sequence of true tokens, the false token in the control arc blocks the select actor until a second occurrence of a false token. Therefore, accumulation of tokens occurs at the input of the select, precisely at the port labelled T.

The same specification can be modelled with the Petri Net in figure 7b. In this case there is no unbounded accumulation of tokens: every token produced by the input transition t_1 enters one of the two branches and follows the path to the output transition t_8. However, the net is not determinate because of the presence of the non-deterministic merge: there is no guarantee on the order of the tokens in the output stream if t_6 and t_7 are concurrently enabled. To solve this problem, it is necessary to impose some restrictions to the scheduler to ensure that the non-deterministic states where two merging transitions are concurrently enabled are never reached. One solution could be to allow tokens to enter the modified if-then-else structure (i.e. firing t_1) only if no other transition is enabled. Further investigation in this direction is necessary to formally define and prove the conditions under which a PN is determinate.

4 Quasi-static Scheduling of FCPN

4.1 Definition of schedulability

Let $\Sigma = \{\sigma_1, \sigma_2 ...\}$ be a non-empty finite set of finite firing sequences such that $\forall \sigma_i \in \Sigma, M_0[\sigma_i > M_0$ and let σ_i^j be the j-th transition in sequence $\sigma_i = (\sigma_i^1 \sigma_i^2 ... \sigma_i^{j-1} \sigma_i^j \sigma_i^{j+1} ... \sigma_i^N)$. Every sequence $\sigma_i \in \Sigma$ contains at least one occurrence of each source transition of the net. We define the operator Θ over a pair of transitions t and $t' \in T$ as follows:

$$\Theta(t, t') = \begin{cases} 1 & \text{if } t \text{ and } t' \text{ are in Equal Conflict Relation,} \\ 0 & \text{otherwise.} \end{cases}$$

Definition 4.1.1 *The set Σ is a* **valid schedule** *if $\forall \sigma_i \in \Sigma, \forall \sigma_i^j \in \sigma_i$ s.t. $\sigma_i^j \neq \sigma_i^h$ $\forall h < j$, $\forall t_k \in T$ s.t. $t_k \neq \sigma_i^j$ and $\Theta(t_k, \sigma_i^j) = 1$, $\exists \sigma_l \in \Sigma$ s.t.*
(1) $\sigma_l^m = \sigma_i^m$, $\forall m \leq j-1$
(2) $\sigma_l^m = t_k$, $m{=}j$
(3) $\sigma_l^m \neq \sigma_i^j$, $\forall m \geq j+1$

Definition 4.1.2 *Given an EFCPN N and an initial marking M_0, the pair (N, M_0) is* **(quasi-statically) schedulable** [1], *if there exists a valid schedule.*

This definition of schedulability extends to non-static Data Flow networks the concept of SDF scheduling given in Section 3. If the net contains non-deterministic choices that model data dependent structures like if-then-else or while-do, a valid schedule is a set of firing sequences, one for every combination of boolean values of the control tokens. A valid schedule must contain a valid firing sequence for every possible outcome of a choice because the value of the control tokens is unknown at compile time when the valid schedule is computed.

Schedulability implies the **existence** of at least a valid schedule that ensures that there is no unbounded accumulation of tokens in any place. This property is different from k-boundedness, that implies that **for all** the reachable states, the number of tokens in any place does not exceed a certain number k. In most cases a schedulable PN is not k-bounded (see for example the Petri Net shown in figure 4), while a live and bounded net (also called well-behaved) is always schedulable.

Let us consider some examples. Given the net in figure 6a $\Sigma = \{\sigma_1, \sigma_2\}$, where $\sigma_1 = (t_1 t_2 t_4)$ and $\sigma_2 = (t_1 t_3 t_5)$, is a valid schedule because it contains a valid firing sequence for every value carried by the token in choice p_1 and therefore, whichever conflicting transition fires, there is a sequence that consumes all the tokens and returns the net to the initial state.

Instead, the net shown in figure 6b is not schedulable because it is impossible to find a schedule set that satisfies all three conditions in Definition 4.1.1. For example a set containing sequences $t_1 t_2 t_1 t_3 t_4$ and $t_1 t_3 t_1 t_2 t_4$ is not considered a valid schedule because it does not contain any firing sequence beginning with $t_1 t_2 t_1 t_2$ (conditions (1) and (2)). This correspond to the fact that when transition t_2 (t_3) is always fired, there exists no firing sequence that returns the net to the initial state and therefore unbounded accumulation of tokens occurs in place p_2 (p_3).

If we consider the net shown in figure 8, $\Sigma = \{(t_1 t_2 t_1 t_2 t_4)(t_1 t_3 t_5)\}$ is a valid schedule. The presence of the arc of weight 2 requires that the conflicting transition t_2 is fired twice before transition

[1] Here we consider only schedules that return the net to the initial state. It is also possible to provide a similar definition for the case where the schedule has an initial acyclic sequence.

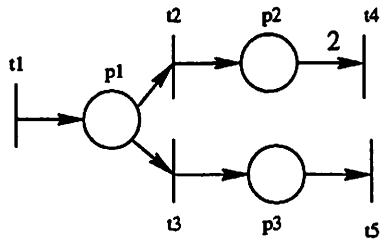


Figure 8: Another example of schedulable net (with weighted arcs)

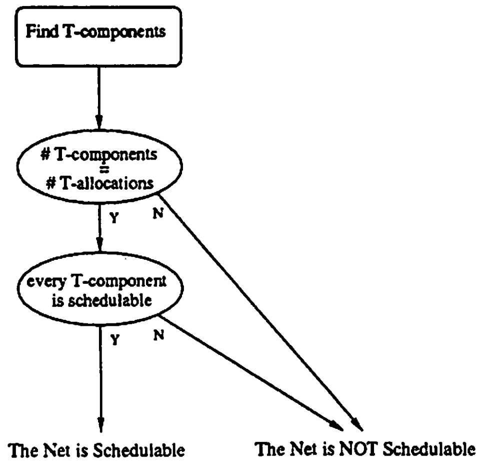

Find T-components
# T-components = # T-allocations
Y
N
every T-component is schedulable
Y
N
The Net is Schedulable
The Net is NOT Schedulable

Figure 9: Flow to check schedulability

t_4 is enabled. However, there is not guarantee that this happens within a cycle. If transitions $t_1t_2t_1t_3t_5$ fire in this order, one token remains in place p_2 and the net does not return to the initial marking. The net is considered schedulable because repeated executions of this sequence do not result in unbounded accumulation of tokens (as soon as there are two tokens in place p_2, transition t_4 is fired and they are consumed).

In the next Sections we present an algorithm that finds a valid schedule, if the net is schedulable. In the example shown in figure 8 where the net contains weighted arcs, $\{t_1t_3t_5, t_1t_2t_1t_2t_4\}$ is a valid schedule although it does not include all the possible cyclic firing sequences, even of infinite length, that can occur $(\{t_1t_3t_5, t_1t_2(t_1t_3t_5)^n t_1t_2t_4, n = 0, 1, 2...\})$. Therefore, a valid schedule should be intended only as a set of cyclic firing sequences that ensure bounded memory execution of the net. This set is complete in the sense that it is possible to derive with further manipulation a C-code implementation of the scheduler.

4.2 How to find a valid schedule

The algorithm is based on the following strategy: the net is first decomposed into as many Conflict Free (CF) components as the number of possible solutions for the non-deterministic choices of the net. Then, each component is statically scheduled. If every component is schedulable, we take as valid schedule of the net a set that contains one valid firing sequence for every CF component. If any of the CF components is not schedulable, the net itself is said to be not schedulable (figure 9).

Previous work has already used a similar approach called MG decomposition. The fundamental

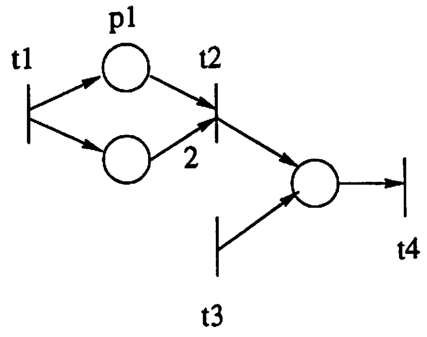


Figure 10: Conflict Free net

theorem of Hack [8] on live and safe strongly connected FCPNs is based on the decomposition of the net into as many MG reductions as the number of the possible allocations of the non-deterministic choices. In [3] Best proposes an iterative algorithm to decompose a strongly connected ordinary Petri Net into a set of strongly connected Marked Graphs. More recently, Teruel in [4] extends to weighted nets known results for ordinary nets [1]. These works have their main application in checking whether a given strongly connected net is bounded. However, in the domain of embedded reactive systems most applications usually have lots of interactions with the environment, that are naturally modelled as source and sink transitions. As a result, nets modelling embedded systems are not strongly connected. Moreover, as we outlined in the previous Section, boundedness is a too restrictive property for our objective. For this reason we modify Hack's MG decomposition algorithm and apply it to the class of FCPNs that have source and sink transitions. Note that source and sink places correspond only to finite execution and therefore are not allowed in the PNs that we consider.

Definition 4.2.1 *Let $N = (T, P, F)$ and $N' = (T', P', F')$ be two PNs. N' is a* **subnet** *of N if $T' \subseteq T$, $P' \subseteq P$ and $F' = F \cap ((T' \times P') \cup (P' \times T'))$.*

Definition 4.2.2 *A subnet N' is a* **transition-generated subnet** *of N if P' is the set of all predecessors and successors places in N of the transitions in T' (i.e. $t \in T' \Rightarrow t^{\bullet} \subseteq P', {}^{\bullet}t \subseteq P'$).*

Definition 4.2.3 *A* **T-component** *N' of a net N is a set of transition-generated subnets such that each of them is a consistent Conflict Free net and $\forall t_s \in T_s$ ($T_s \subseteq T$ is the set of source transitions in N) there exists a T-invariant T_i s.t. $t_s \in support(T_i)$.*

The condition that every source transition of a subnet is in the support of a T-invariant ensures that tokens produced by any source transition do not accumulate and that for a T-component there exists a finite complete cycle that guarantees bounded execution. Consider the net in figure 10. It is consistent because there exists a T-invariant ($T = (0, 0, 1, 1)$), but unbounded accumulation of tokens produced by source transition t_1 occurs in place p_1. This example shows that simple consistency is not enough to ensure bounded scheduling of a Conflict Free net.

Definition 4.2.4 *A T-component is schedulable if there exists a firing sequence that returns it to the initial state without any deadlock when its execution is simulated.*

Definition 4.2.5 *A set of T-components* **covers** *a FCPN if each transition of the net belongs to at least one T-component.*

Definition 4.2.6 *A* **T-allocation** *over a FCPN is a function $\alpha : P \to T$ that chooses one among the various successors of each place ($\forall p \in P \alpha(p) \in p^{\bullet}$).*

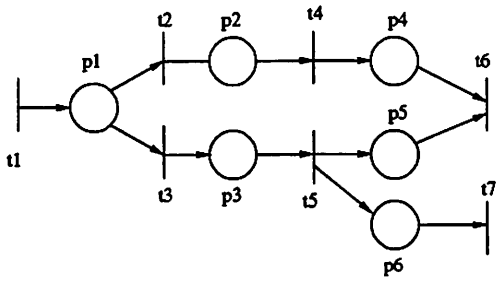


Figure 11: Not schedulable FCPN

A net with P places and n_i successors each has $\prod_{i=1..P} n_i$ distinct T-allocations.

Definition 4.2.7 *The* **T-reduction** *associated with a T-allocation is a set of subnets generated from the image of the T-allocation using the following Reduction Algorithm.*

Let C_1,C_2, ...C_M be the Equal Conflict Sets defined in Section 2.2 as the equivalence classes of the Equal Conflict Relation and $\alpha_i = \{t_1, t_2, t_3, ...t_M\}$ the i-th T-allocation where $t_k \in C_j$. The T-reduction R_i corresponding to T-allocation α_i is generated as follows (see figure 12).

Reduction Algorithm (modified from [8])

1. $R_i = N$ ($R_{T_i} = T$, $R_{P_i} = P$, $R_{F_i} = F$).
2. For all $t_k \in R_{T_i}$ and $t_k \notin \alpha_i$
   (a) Remove t_k.
   (b) $\forall s \in t_k^\bullet$, remove place s unless one of the following conditions holds:
      i. s has a predecessor transition different from t_k ($\exists t \in^\bullet s$ s.t. $t \in R_{T_i}$).
      ii. the successor transition of s has a predecessor place that is different from s and is not a source place ($\exists t \in^\bullet (^\bullet(s^\bullet))$ s.t. $t \in R_{T_i}$).
   (c) If s_i is a removed place, $\forall t_j \in s_i^\bullet$, remove t_j if one of the following conditions holds:
      i. t_j has no predecessor place ($|^\bullet t_j| = 0$).
      ii. all predecessors of t_j are source places. In this case remove every $s \in^\bullet t_j$.
   (d) Apply the previous two steps until they cannot be applied any longer.

Theorem 4.2.1 *The T-reduction R_i obtained by applying the reduction algorithm is*

1. a set of transition-generated Conflict Free nets. $\{R_i^1, R_i^2, ...R_i^M\}$.

2. a T-component of N iff every subnet R_i^l is consistent and every source transition of N is in the support of at least a T-invariant of R_i^l.

Proof: 1. Each R_i is a Conflict Free net by construction because it contains at most one transition for every Equal Conflict Set. Let's show that each subnet is transition-generated. We need to prove that $\forall t_k \in R_{T_i}$ and $\forall s \in {}^\bullet t_k \cup t_k^\bullet$, $s \in R_{P_i}$. For places $s \in t_k^\bullet$, it is easy to see that the algorithm does not remove a place s if $\exists t \in^\bullet s$ s.t. $t \in R_{T_i}$ (condition (b)i). For places $s \in^\bullet t_k$, s is removed only if all other predecessor places of t_k are source places (condition (b)ii). In this case at the next iteration step also t_k is removed (condition (c)ii). Therefore, if $t \in R_{T_i}$ it follows that $t^\bullet \cup^\bullet t \subseteq R_{P_i}$.

2. Both directions trivially follow from Theorem 4.2.1 and Definition 4.2.3 of T-component. □

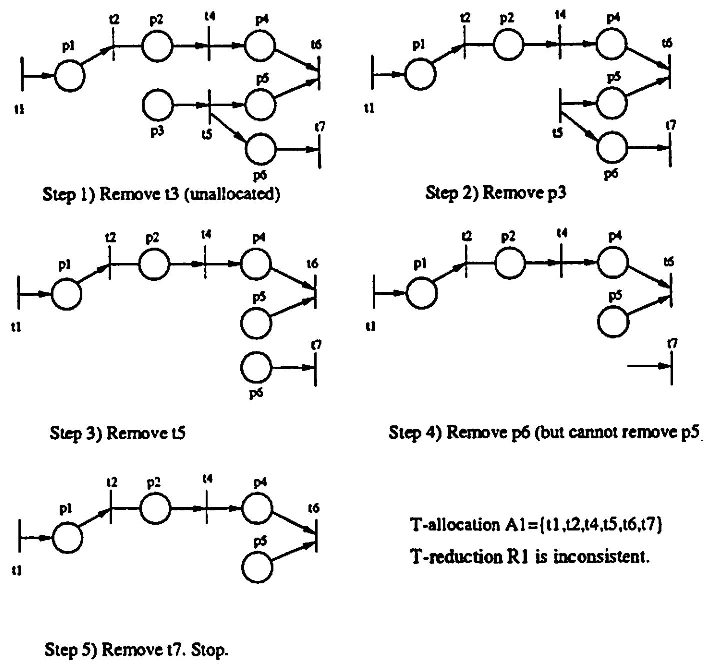

Step 1) Remove t3 (unallocated)
Step 2) Remove p3
Step 3) Remove t5
Step 4) Remove p6 (but cannot remove p5
Step 5) Remove t7. Stop.
T-allocation A1={t1,t2,t4,t5,t6,t7}
T-reduction R1 is inconsistent.

Figure 12: How to obtain a T-reduction

Definition 4.2.8 *A FCPN N is* **T-decomposable** *if there exists a set* $\{T_1, T_2 ...\}$ *of T-components that covers the net.*

Definition 4.2.9 *A FCPN N is* **T-allocatable** *if every T-reduction generated from a T-allocation is a T-component.*

Lemma 4.2.1 *If a FCPN is T-allocatable, it is T-decomposable; the converse is not always true. [8]*

The following fundamental theorem states that T-allocatability is a necessary and sufficient condition for schedulability. Intuitively, T-allocations can be interpreted as control functions that choose which transition fires among several conflicting ones and at the same time which component of the net is active at every cycle. A net is T-allocatable if all its components, each of them corresponding to a sequence of choices, are T-components. Therefore, if every T-component is also schedulable, a T-allocatable net can be executed forever with bounded memory, because for every choice there is always the possibility to complete successfully a finite cycle of firings that returns the net to the initial state.

Theorem 4.2.2 *A FC-PN is schedulable iff*

1. *it is T-allocatable,*
2. *every T-component is schedulable,*

Proof: ($\Rightarrow$) Let's prove that if the net is not T-allocatable it is not schedulable. If the net is not T-allocatable, there is at least one T-allocation α_i for which the generated T-reduction R_i is not a T-component. This may occur in one of the following cases:

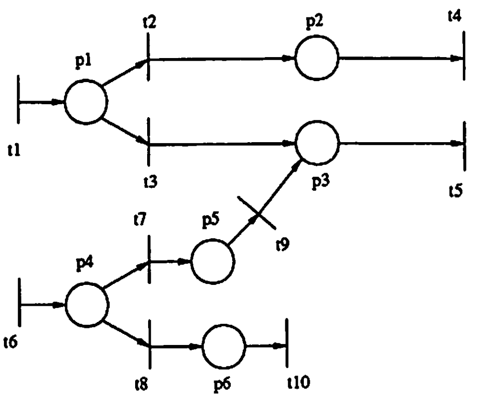


Figure 13: Schedulable FCPN

- Case 1. There exists a source transition t_i that is not in the support of any T-invariant of R_i. This implies that there is no finite complete cycle of R_i that contains t_i. Therefore there exists no schedule set with a finite complete cycle containing transition t_i and all the conflicting transitions allocated in α_i. This violates the hypothesis that every sequence of the valid schedule set contains at least one occurrence of each source transition of the net.
- Case 2. R_i is a strongly connected net and it is inconsistent. Therefore, there is no firing sequence that returns R_i to its initial state. This means also that there exists no schedule set with a finite complete cycle containing all the conflicting transitions that are allocated in α_i.

In both cases the net is not schedulable because there is no schedule set that contains one finite complete cycle for every combination of boolean values.

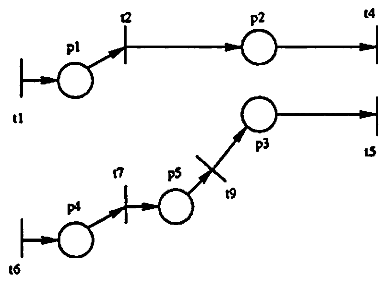
($\Leftarrow$) By construction. The net is T-allocatable, then there is a T-component for each T-allocation. By hypothesis every T-component is a schedulable Conflict Free net. The set of all the valid firing sequences, one for each T-component, is a valid schedule of the net because it contains one sequence for every solution of the non-deterministic choices. Therefore the net is schedulable. $\square$

To check if a given net is T-allocatable, it is necessary to verify that every T-reduction obtained from the Reduction Algorithm is a T-component. For this purpose it is necessary to solve the state equations for every subnet of the T-reduction and check consistency; in the case any subnet of the T-reduction contains merge places, it is also necessary to check for this subnet that every source transition is in the support of at least one T-invariant. Then, the schedulability of a T-component is detected by simulation of the T-invariants found in the previous step.

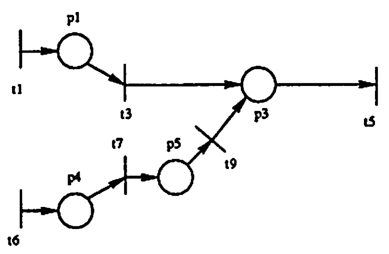
Let us describe two examples. The FCPN presented in figure 11 is not schedulable. Figure 12 shows the steps of the Reduction Algorithm applied to T-allocation $A_1 = \{t_1, t_2, t_4, t_5, t_6, t_7\}$. The generated T-reduction R_1 is not consistent because it contains a source place, therefore it is not a T-component and the net is not schedulable. In fact, if the sequence $\sigma = (t_1 t_2 t_4 t_6)$ is fired infinitely often, there is unbounded accumulation of tokens in place p_4.

The FCPN shown in figure 13 is schedulable. The T-components for the corresponding T-allocations are represented in figure 14. A valid schedule is given by the following firing sequences $\{(t_1 t_2 t_4 t_6 t_7 t_9 t_5), (t_1 t_3 t_5 t_6 t_7 t_9 t_5), (t_1 t_2 t_4 t_6 t_8 t_{10}), (t_1 t_3 t_5 t_6 t_8 t_{10})\}$.

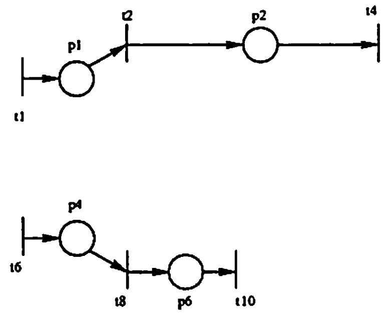
The Reduction Algorithm, whose cost is linear in the number of nodes in the net, must be applied in the worst case as many times as the number of T-allocations. However, in most cases it is not necessary to apply it exhaustively for all the T-allocations. Assume that, during application of the Reduction Algorithm to T-allocation α_k, transition $t_i \in \alpha_k$ s.t. $\exists t_j \neq t_i$ and $\Theta(t_i, t_j) = 1$ is removed. Consider the T-allocation α_h where $t_j \in \alpha_h$ (and therefore $t_i \notin \alpha_h$) and $\forall t \in \alpha_k$



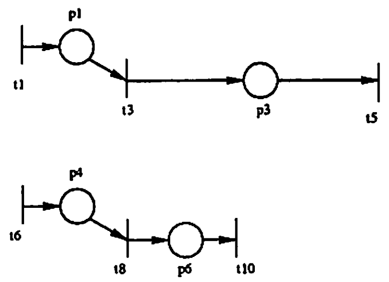
T-allocation A1={t1,t2,t4,t5,t6,t7,t9,t10}



T-allocation A2={t1,t3,t4,t5,t6,t7,t9,t10}



T-allocation A3={t1,t2,t4,t5,t6,t8,t9,t10}



T-allocation A4={t1,t3,t4,t5,t6,t8,t9,t10}

Figure 14: T-components

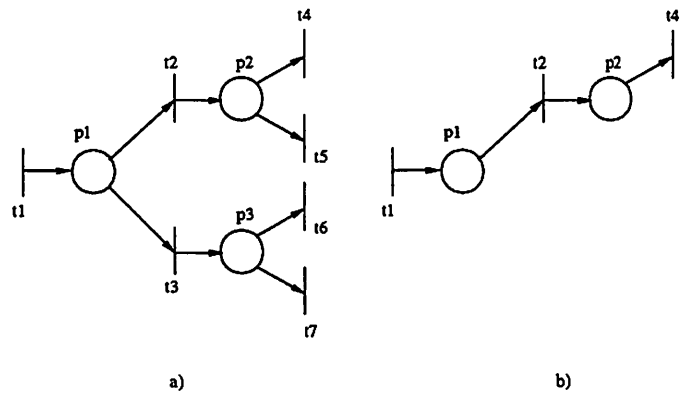


Figure 15: Optimization

s.t. $t \neq t_i, t \in \alpha_h$. The T-reductions R_k and R_h are equal, because the Reduction Algorithm removes both transition t_i and t_j when applied to T-allocation α_h and α_k. By taking in account this property, it is possible to apply the Reduction Algorithm only once for every different T-reduction. The example shown in figure 15a presents one of these cases. The net has three Conflict Relation Sets $\{t_2, t_3\}, \{t_4, t_5\}, \{t_6, t_7\}$. If we consider T-allocations $\alpha_1 = \{t_1 t_2 t_4 t_6\}$ and $\alpha_2 = \{t_1 t_2 t_4 t_7\}$ and apply the Reduction Algorithm, we obtain for both of them the same T-reduction that is shown in figure 15b. This happens because, when t_2 is allocated, both t_6 and t_7 cannot be enabled (since t_3 is not allocated) and therefore are removed in the reduction process.

5 Conclusions and future work

In this paper we have proposed an algorithm to find a quasi-static schedule for Extended Free Choice Petri Nets whenever it exists. This result is important because it allows to reduce considerably the amount of run-time scheduling overhead. We also explained, by showing the differences of the two models, how this algorithm can find a larger domain of applications than the well-behaved graphs [5] handled successfully in the BDF domain. So far, we restricted our scope of investigation to FCPNs, because it allows one to perform analysis at relatively low cost. However, not every embedded system specification can be modelled as a Free-Choice Petri Net. Future research will be done in the direction of broadening the scope to a larger class of Petri Nets and defining an algorithm to find a quasi-static schedule also for such class. In this case our decomposition-based approach is probably no longer valid, because a generic PN may be schedulable even though it is not T-allocatable. We hence believe that outside the Free-Choice domain it is necessary to take a different approach, that may make use of more expensive analysis techniques, like Petri Net unfoldings.

References

[1] J. Desel and J.Esparza. *Free choice Petri nets.* Cambridge University Press, 1995.

[2] E.A.Lee and D.G.Messerschmitt. Static scheduling of synchronous dataflow programs for digital signal processing. *IEEE Transactions on computers*, January 1987.

[3] E.Best. Structure theory of petri nets: the free choice hiatus. In *Advances in Petri Nets*, 1986.

[4] E.Teruel. *Structure theory of Weighted Place/Transition Net systems. The Equal Conflict hiatus.* Ph.D dissertation. Universidad de Zaragoza, 1994.

[5] G.R.Gao, R.Govindarajan, and P.Panangaden. Well-behaved dataflow programs for dsp computation. In *Proceedings ICASSP*, 1992.

[6] W.A. Halang and A.D. Stoyenko. *Constructing predictable real time systems.* Kluwer Academic Publishers, 1991.

[7] J.Buck. *Scheduling dynamic dataflow graphs with bounded memory using the token flow model.* Ph.D dissertation. UC Berkeley, 1993.

[8] M.Hack. *Analysis of Production Schemata by Petri Nets.* Master thesis. MIT, 1972.

[9] M.Kishinevsky, J.Cortadella, A.Kondratyev, L.Lavagno, A.Taubin, and A.Yakovlev. Coupling asynchrony and interrupts: Place charts nets. In *Application and theory of Petri Nets 1997. 18th International Conference*, 1997.

[10] J.L. Peterson. *Petri net theory and the modeling of systems.* Prentice Hall, 1981.

[11] T.Murata. Petri nets: properties, analysis and applications. In *Proceedings of the IEEE*, April 1989.

[12] T.Parks. *Bounded scheduling of Process Networks.* Ph.D dissertation. UC Berkeley, December 1995.