Schedulability Analysis of Petri Nets Based on Structural Properties

Cong Liu[*][†]
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720, USA
congliu@eecs.berkeley.edu

Alex Kondratyev, Yosinori Watanabe
Cadence Berkeley Labs, Berkeley, CA 94704, USA
{*kalex, watanabe*}*@cadence.com*

Jörg Desel
Lehrstuhl für Angewandte Informatik, Katholische Universität Eichstätt-Ingolstadt
Ostenstr. 28, 85071 Eichstätt, Germany
joerg.desel@ku-eichstaett.de

Alberto Sangiovanni-Vincentelli
University of California, Berkeley, CA 94720, USA
alberto@eecs.berkeley.edu

Abstract. A schedule of a Petri Net (PN) represents a set of firing sequences that can be infinitely repeated within a bounded state space, regardless of the outcomes of the nondeterministic choices. Schedulability analysis for a given PN answers the question whether a schedule exists in the reachability space of this net. This paper suggests a novel approach for schedulability analysis based solely on PN structure. It shows that unschedulability can be caused by a structural relation among transitions modelling nondeterministic choices. A method based on linear programming for checking this relation is proposed. This paper also presents a necessary condition for schedulability based on the rank of the incidence matrix of the underlying PN. These results shed a light on the sources of unschedulability often found in PN models of embedded multimedia systems.

Keywords: Petri net, structural property, quasi-static scheduling

[*]Address for correspondence: Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, 253 Cory Hall #1772, Berkeley, CA 94720-1772, USA
[†]This research was partly supported by MARCO Gigascale Systems Research Center award 2003-DT-660.

1. Introduction

The use of concurrent models has become a necessity in embedded system design. This trend is driven by the growing complexity and inherent multitasking of embedded systems. Describing a system as a set of concurrently executed, relatively simple subtasks is more natural than using a single, complicated task. Modelling concurrency explicitly is essential in finding an efficient mapping of specifications into hardware, which is inherently concurrent, or programmable platforms with multiple computing engines.

Embedded systems, however, have limited resources. They often have a few processors, small memories, and limited services provided by an operating system (if any). This implies that several concurrent processes have to share a physical resource (e.g., CPU or bus). Thus, scheduling their operations is inevitable. For embedded software, it means that concurrent processes (programs) have to be sequentialized. This process is in general implemented manually. Obviously, this is a tedious, time-consuming, and error-prone task. To automate the transformation, a number of synthesis algorithms [12] [14] [16] [6] have been proposed.

In this paper, we rely on a modelling framework [6] [7] that uses Petri Nets (PNs) to represent concurrent processes. This body of work considers processes that asynchronously communicate with each other and the environment through unbounded FIFO buffers [9] [10] [11]. The synthesis starts by transforming each of the processes that can be specified in a conventional programming language (e.g., C) into PN. Then a set of communicating PNs is scheduled resulting in a single sequential process.

Consider the example shown in Figure 1. Figure 1(a) describes two concurrent processes. Process *Filter* receives samples from the environment and then conditionally keeps sending processed samples to Process *Multiplier*. Depending on the availability of data, Process *Multiplier* either receives a processed sample and outputs the product of the sample and a coefficient, or updates the coefficient supplied by the environment. If both sample and coefficient are present, the choice of which one to consume is done nondeterministically.

Figure 1(b) shows the generated PN, which represents the control flow and communications of the processes. Specifically, a token in place p_2 models the program counter of Process *Filter*, and a token in place p_4 or p_5 models the program counter in Process *Multiplier*. Places p_1, p_3, and p_6 model unbounded buffers *IN, CHAN, and COEF*, respectively. Transitions t_1 and t_6 model environment stimuli, and other transitions model the inlined operations. Note that the data-dependent choice is modelled by a *free choice* (Definition 1).

Figure 1(c) depicts the initialization and a *schedule* (Definition 2) of the PN. The schedule represents an infinite sequential execution of the system within a bounded state space, no matter which environment stimulus (t_1 or t_6) is present, and which data-dependent choice (t_3 or t_4) is taken during execution. The single process shown in Figure 1(d) is generated from the schedule by substituting transitions with inlined operations. Note that although originally buffers are assumed to be unbounded, the sequential process can repeatedly execute using buffers of size one.

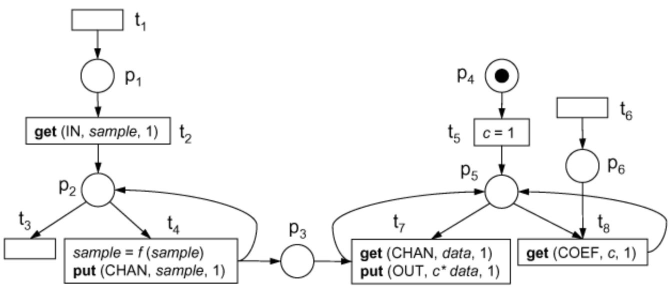
The synthesis method suggested in [6] is based on heuristics because the existence of a solution for the scheduling problem is proven only for simple subclasses of PNs (such as Marked Graphs, see [12]). For general PNs the decidability of the scheduling problem remains open. Therefore, discovering powerful sufficient conditions for unschedulability of PN is important for gaining efficiency in analysis.

This paper suggests a structural approach to schedulability analysis of PNs. This approach is chosen due to two reasons. First, the underlying algorithms have polynomial-time complexity. Second, the results are applicable to all initial markings.

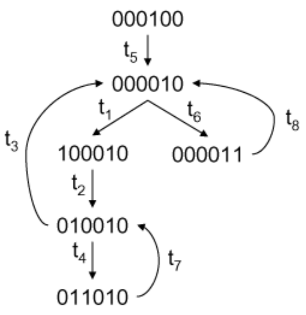
```
process Filter
(InPort IN; OutPort CHAN)
begin
    loop
        get (IN, sample, 1)
        while (sample > threshold)
            sample = f (sample)
            put (CHAN, sample, 1)
        end while
    end loop
end proc
```

```
process Multiplier
(InPort CHAN, COEF; OutPort OUT)
begin
    c = 1
    loop
        select
            case CHAN
                get (CHAN, data, 1)
                put (OUT, c*data, 1)
            case COEF
                get (COEF, c, 1)
        end select
    end loop
end proc
```

(a)



(b)



(c)

```
process Filter&Multiplier
(InPort IN, COEF; OutPort OUT)
begin
    c = 1
    loop
        select
            case IN
                get (IN, sample, 1)
                while (sample > threshold)
                    sample = f (sample)
                    data = sample
                    put (OUT, c*data, 1)
                end while
            case COEF
                get (COEF, c, 1)
        end select
    end loop
end proc
```

(d)

Figure 1. (a) Communicating concurrent processes. (b) A PN representation. (c) A schedule of the PN with initialization. (d) The process transformed from the schedule.

Our contribution can be summarized as follows.

- After the basic concepts of PNs are reviewed (Section 2) and a schedule of a PN is defined (Section 3), we introduce *cyclic dependence relation*, a structural property defined on transitions of PN (Section 4). To the best of our knowledge, this property provides the most general sufficient condition for PN unschedulability.

- Though cyclic dependence is defined through T-invariants, which are nonnegative *integer* solutions to a homogenous linear system, we show that it is not necessary to solve an integer programming problem to check the property. We propose an exact algorithm that is based on linear-programming (Section 5).

- We prove another sufficient condition for unschedulability based on the *rank* of the incidence matrix (Section 6). Checking this condition is computationally efficient.

- We demonstrate the effectiveness and efficiency of our approach by applying it to check unschedulability of PNs generated from concurrent models of industrial JPEG and MPEG codecs (Section 7).

2. Basic Definitions and Notations

A *PN* is a 4-tuple (P, T, F, M_0). $P = \{p_1, p_2, \dots, p_m\}$ is a set of *places*. $T = \{t_1, t_2, \dots, t_n\}$ is a set of *transitions*. $F\colon (P \times T) \cup (T \times P) \to \mathbb{N}$ is the flow relation. $M_0\colon P \to \mathbb{N}$ is the *initial marking*, where $\mathbb{N}$ denotes the set of nonnegative integers. Let $N = (P, T, F)$ denote a PN structure without any specific initial marking, and (N, M_0) denote a PN with a given initial marking. Let $v \in P \cup T$. Its preset and postset are given by ${}^\bullet v = \{u \in P \cup T | F(u, v) > 0\}$, $v^\bullet = \{u \in P \cup T | F(v, u) > 0\}$.

A transition t is *enabled* at a given marking M, if for each places $p \in P$, $M(p) \geq F(p, t)$. When a transition is enabled it can *fire*. The new marking M' reached after the firing of transition t is defined as: $p \in P$, $M'(p) = M(p) - F(p, t) + F(t, p)$. In this paper we use nets with *source* transitions, i.e. with empty presets. These transitions model the behavior of the input stimuli to a reactive system. They are enabled in any PN marking.

A marking M' is *reachable* from the marking M if there exists a sequence of firings that transforms M to M'. This is denoted by $M[\sigma > M'$, where σ represents a *firing sequence* $(t_{\sigma1}, t_{\sigma2}, \cdots, t_{\sigma k})$. The firing sequence is said to be *cyclic* if $M' = M$. The *firing count vector* $\bar{\sigma}$ of a firing sequence σ is a $|T|$-vector, where the $i-$th entry denotes the number of times transition t_i appears in σ. The set of markings reachable from the initial marking M_0 is denoted as $R(N, M_0)$.

The *incidence matrix* $\boldsymbol{A} = [a_{ij}]$ is a $|T| \times |P|$ matrix, where $a_{ij} = F(t_i, p_j) - F(p_j, t_i)$. If a marking M' is reachable from M through a firing sequence σ then $M' = M + \boldsymbol{A}^{\mathrm{T}}\bar{\sigma}$. A *T-invariant* is a nonnegative integer solution to $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{x} = 0$. It is known that a $|T|$-vector $\boldsymbol{x}$ is a T-invariant if and only if there exists a marking M and a firing sequence σ from M back to M with $\bar{\sigma} = \boldsymbol{x}$ [15]. A T-invariant $\boldsymbol{x}$ is *minimal* if there exists no T-invariant $\boldsymbol{x}' \neq 0$ with $\boldsymbol{x}' \leq \boldsymbol{x}$. The set of transitions corresponding to non-zero entries in a T-invariant x is called the *support* of an invariant and is denoted by $\|\boldsymbol{x}\|$. A support is said to be *minimal* if no proper nonempty subset of the support is also a support. Given a minimal support of a T-invariant, there exists a unique minimal T-invariant corresponding to the minimal support [15], that is called a *minimal support* T-invariant.

A PN (N, M_0) is *bounded* if there exists a nonnegative integer k, such that for each reachable marking $M \in R(N, M_0)$, $M(p) \leq k$ for each place $p \in P$. A PN (N, M_0) is said to be *live*, if for each transition $t \in T$ and for each reachable marking $M \in R(N, M_0)$, there exists a firing sequence σ from M such that σ contains t. A PN is said to be *deadlock-free* if for each reachable marking, there exists at least one enabled transition.

3. Schedule

First, we introduce free choice sets, a key concept in the definition of a schedule.

Definition 1. (Free choice)
Two distinct transitions t and t' are in free choice relation, if for each place $p \in {}^\bullet t \cup {}^\bullet t'$ and for each transition $t'' \in p^\bullet$, $F(p, t) = F(p, t') = F(p, t'')$.

The relation is symmetric and transitive. We call the maximal set of transitions that are pairwise in free choice relation a *free choice set* (FCS). Note that source transitions make a single FCS according to Definition 1.

We introduce the notion of FCS mainly to model environmental stimuli and run-time data-dependent choices, which are unknown at design time. For the sake of simplicity, when referring to FCSs we only refer to FCSs that contain exactly two transitions. The assumption can be satisfied by representing an arbitray FCS with n transitions as a a tree of binary FCSs. The PN shown in Figure 1(b) has exactly two binary FCSs $\{t_1, t_6\}$ (which are source transitions) and $\{t_3, t_4\}$.

Definition 2. (Schedule)
A *schedule* of a PN (N, M) with a set T_s of source transitions is a digraph (V, E, r) rooted by r with the following properties:

1. V and E are finite and nonempty.
2. There exists a mapping $\mu\colon V \to R(N, M)$ with $\mu(r) = M$. For each $(u, v) \in E$, there exists $t \in T$ such that $\mu(u)[t > \mu(v)$. This is denoted by $u \xrightarrow{t} v$.
3. Given $u \xrightarrow{t} v$, there exists $u \xrightarrow{t'} v'$ if and only if t, t' are in a FCS.
4. For each $v \in V$, there exists a directed path to an *await* vertex v_a satisfying $\forall t \in T_s, v_a \xrightarrow{t}$.
5. For each $v \in V$, there exists a directed path from r to v and from v to r.

The requirement in Property 1 ensures that any execution of the schedule visits a finite subset of a state space: the underlying system can be executed with bounded memory.

Property 2 states that a vertex in a schedule (called schedule state or simply state) corresponds to a reachable marking of the PN, an edge corresponds to a transition, and a path corresponds to a firing sequence. Note that it is allowed that several schedule states are mapped to the same marking, and these states fire different transitions at their output edges.

Property 3 has two implications. First, if an outgoing edge of a vertex corresponds to a transition in a FCS, then there must exist another outgoing edge of the vertex that corresponds to another transition in the FCS. It ensures that a schedule is complete, because all possible outcomes of FCSs are considered.

In this case, the FCS is said to be *involved* in the schedule. Second, if a state of a schedule has more than one outgoing edge then these edges must correspond to transitions in a FCS. Other enabled transitions are not considered at a state. This property ensures that the schedule can be sequentially executed.

Property 4 guarantees the progress of a schedule, i.e. from any state of a schedule one can reach an await state in which source transitions fire and therefore inputs from the environment are served. This is a necessary condition for reactive systems. The progress notion could be formulated with respect to any set of transitions that are in FCS (not necessarily source ones). This makes it possible to treat PNs without source transitions within the same framework.

Property 5 implies that a schedule is *strongly connected*. Thus, following a schedule, the underlying system can be infinitely executed without deadlock. Note that Property 5 of returning to the root makes the definition of schedule more stringent than the one suggested in [7]. We believe that this is acceptable because most practical applications has a designated reset state that is reachable from any other state (emergency exit) and from which the operation restarts. Such reset state may be interpreted as the root state in Definition 2 of a schedule.

Given a schedule graph $G(V, E, r)$ one can construct its spanning tree that imposes a partial order between schedule states, i.e. $v_1 < v_2$ if and only if there exists a path from v_1 to v_2 in the spanning tree. Given the ordering relation between schedule vertices, the graph G can be unfolded into a rooted tree $G'(V', E', r')$ (we denote objects in the unfolding by decorating the corresponding objects in a schedule with $'$). Unfolding terminates at states v_i' such that for the corresponding schedule state v_i, $v_i \xrightarrow{t} v_j$ and $v_j < v_i$ (i.e. (v_i, v_j) is a backward edge). Clearly an unfolding of a schedule is finite because a schedule is finite.

Definition 3. (Schedulability)
A PN (N, M_0) is said to be schedulable if there exists a schedule (N, M) for some of its reachable markings $M \in R(N, M_0)$. A PN N is said to be schedulable, if there exist a marking M of N and a schedule of (N, M).

Thus, a PN N is said to be unschedulable, if for any marking M of N there exits no schedule of (N, M). The results presented in this paper give sufficient conditions for unschedulability.

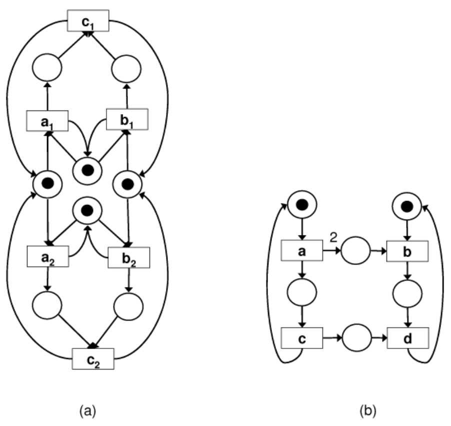


Figure 2. (a) A non-live, bounded, schedulable Petri net. (b) A live, unbounded, and unschedulable Petri net.

In general, schedulability is independent of other Petri net properties, such as boundedness and liveness. We illustrate this with the examples shown in Figure 2. The Petri net in Figure 2(a) models two dining philosophers. If both philosophers pick one chop stick, which is modelled by firing transition a_1, b_2 or a_2, b_1, the system deadlocks and no one can obtain a pair. The Petri net is schedulable, because there are no free-choice sets. $(a_1 b_1 c_1)$ and $(a_2 b_2 c_2)$ are cyclic firing sequences that could be contained in its schedules. Figure 2(b) shows a live, unbounded Petri net. The left part models a producer, and the right part models a consumer. The production/consumption rate is inconsistent. A cycle of the producer consists of firing transition a and c once, which produces two tokens and one token in the two channel places, respectively. However, one cycle of the consumer consists of firing transition b and d once, which consumes one token from each channel place. The Petri net has no T-invariants, and is not schedulable.

4. The Cyclic Dependence Theorem

We first introduce a pairwise transition dependence relation to give readers some intuition, and prove a proposition that relates the dependence relation to schedulability of a PN. Then, we generalize the dependence relation to sets of transitions.

4.1. Pairwise Transition Dependence Relation

Definition 4. (Pairwise transition dependence)
A transition t of a PN N is said to be *dependent* on a transition t', if for each T-invariant $\boldsymbol{x}$ of N, $t \in \|\boldsymbol{x}\|$ implies $t' \in \|\boldsymbol{x}\|$. This is denoted by $t \rightarrowtail t'$.

The pairwise transition dependence is a binary relation on T. It is reflexive and transitive, but not symmetric in general.

Proposition 1. Given a PN N with two FCSs $S_1 = \{t_1, t_2\}$, $S_2 = \{t_3, t_4\}$, if $t_1 \rightarrowtail t_3$ and $t_4 \rightarrowtail t_2$, then for any marking M of N, there exists no schedule of (N, M) involving S_1 or S_2.

Proof:
We show that the unfolding $G'(V', E', r')$ of a schedule $G(V, E, r)$ with S_1 or S_2 involved is infinite, which violates finiteness of a schedule.

The proof proceeds by showing the validity of at least one of the two statements:

I1: G' contains an infinite path $r' \rightsquigarrow v'_1 \xrightarrow{t_1} y'_1 \rightsquigarrow v'_2 \xrightarrow{t_1} y'_2 \cdots$, such that the firing sequence corresponding to the path from r to v_k ($k = 1, 2, \dots$) does not contain transition t_3.

I2: G' contains an infinite path $r' \rightsquigarrow u'_1 \xrightarrow{t_4} z'_1 \rightsquigarrow u'_2 \xrightarrow{t_4} z'_2 \cdots$, such that the firing sequence corresponding to the path from r to u_k ($k = 1, 2, \dots$) does not contain transition t_2.

In G' let us choose vertex v' in which transitions from S_1 or S_2 are enabled and v' to be the closest vertex to the root r' with this property. This vertex exists because S_1 or S_2 is involved in a schedule. Without loss of generality we may assume that S_1 is enabled in v'. Then $v'_1 = v'$ in proving I1.

From Property 5 of a schedule, follows that there exists $w', v' \in V'$ such that the path from v' to w' contains t_1 and $\mu(w') = \mu(v')$. This path corresponds to a firing sequence σ that makes a cycle from marking $\mu(v')$ back to itself and hence $\bar{\sigma}$ is a T-invariant. $t_1 \rightarrowtail t_3$ implies $t_3 \in \sigma$. Therefore, σ contains a vertex u' such that $u' \xrightarrow{t_3}$. Let u' be the closest descendant of v' with t_3 enabled.

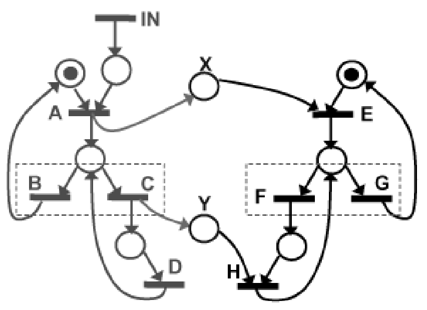


Figure 3. A PN containing pairwise dependent transitions in FCSs.

Let us consider path $\sigma_1 \subset \sigma$ that goes from v' to u'. Two cases are possible.

Case 1. If $t_2 \in \sigma_1$ then σ_1 goes through vertex v_2' with enabled t_2. t_1 and t_2 are from the same FCS and hence t_1 is also enabled in v_2'. Clearly the path from r' to v_2' does not contain t_3 and therefore v_2' satisfies the conditions of I1 and is a descendant of v_1'. Repeating the consideration for v_2' one can conclude about the existence of infinite path satisfying I1.

Case 2. Suppose that $t_2 \notin \sigma_1$. Then the path from r' to u' does not contain transition t_2. In addition t_4 is enabled in u' (being in the same FCS as t_3) and therefore one can use u' as u_1' in proving I2.

Bearing in mind that $t_4 \rightarrowtail t_2$ and applying the same arguments for closing the cycle from u_1' one can conclude that there must exist a path δ from u_1' to v_2' in which t_2 is enabled. If δ does not contain t_3 then v_2' satisfies the conditions of I1, which is the basis for constructing an infinite path. If δ contains t_3 then by choosing the first firing of t_3 in δ, one can obtain a vertex u_2' in which t_3 is enabled together with t_4, and u_2' is a descendants of u_1'. This proves I2. $\square$

Figure 3 shows a PN that satisfies the condition of Proposition 1. It has two FCSs, $\{B, C\}$ and $\{F, G\}$, and two minimal T-invariants with supports $\{IN, A, B, E, G\}$ and $\{C, D, F, H\}$. It is easy to see that $C \rightarrowtail F$ and $G \rightarrowtail B$. Thus, according to Proposition 1 there exists no schedule of the PN that involves either $\{B, C\}$ or $\{F, G\}$.

4.2. General Transition Dependence Relation

Definition 5. (General transition dependence relation)
A transition t of a PN N is said to be *dependent* on a set S of transitions, if for each T-invariant $\boldsymbol{x}$ of N, $t \in \|\boldsymbol{x}\|$ implies $\exists t' \in S : t' \in \|\boldsymbol{x}\|$. This is denoted by $t \rightarrowtail S$.

The pairwise transition dependence relation can be viewed as a special case of the general transition dependence relation with $|S| = 1$. Note that by definition the general transition dependence relation, or simply *dependence relation* is monotonically non-decreasing, i.e. if $t \rightarrowtail S$, then for each $S', S \subseteq S'$, $t \rightarrowtail S'$.

Definition 6. (Cover of a set of FCSs)
A cover S of a set $\mathbb{S}$ of FCSs is a minimum subset of transitions such that for each FCS $F \in \mathbb{S}$, there exists a transition $t \in S \cap F$.

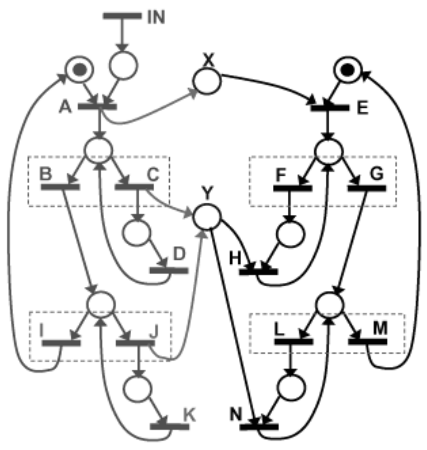


Figure 4. A PN containing FCSs in cyclic (general) dependence relation.

A cover of a set of FCSs contains exactly one transition from each FCS.

Definition 7. (Cyclic dependence relation)
A set $\mathbb{S}$ of FCSs of a PN N is said to be in cyclic dependence relation, if there exists a cover S of $\mathbb{S}$, such that for each transition $t \in S$, $t \rightarrowtail \mathbb{S} \backslash S$.

A FCS is said to be *cyclic dependent* if there exists a set $\mathbb{S}$ of FCSs, such that $\mathbb{S}$ contains the FCS, and is in cyclic dependence relation. Note that although the general dependence relation is monotonic, the cyclic dependence relation is not monotonic in general. Hence we can not prove the existence of the relation in a set of FCSs by proving the existence of the relation for its subsets and vice versa.

Theorem 1. (Cyclic dependence theorem)
No schedule of a PN involves a cyclic dependent FCS.

The proof can be done in a way similar to the proof of Proposition 1.

Figure 4 shows a PN that contains FCSs in cyclic dependence relation. There are four FCSs $\{B, C\}$, $\{F, G\}$, $\{I, J\}$, $\{L, M\}$, and five minimal T-invariants with supports $\{IN, A, B, I, E, G, M\}$, $\{C, D, F, H\}$, $\{C, D, L, N\}$, $\{J, K, F, H\}$, $\{J, K, L, N\}$. Note that there exists no cyclic pairwise dependence relation among transitions in the FCSs. However, there exists a general dependence relation, and furthermore a cyclic dependence relation for the set of all FCSs in the PN. Let $S = \{C, G, J, M\}$ be a cover of the $\mathbb{S}$, then $\mathbb{S} \backslash S = \{B, I, F, L\}$. For each transitions t in S, $t \rightarrowtail \mathbb{S} \backslash S$. Thus, by Theorem 1, there exists no schedule involving any of the FCSs and PN is not schedulable for any initial marking.

5. Checking Cyclic Dependence with Linear Programming

Although the dependence relation is defined on the set of T-invariants of a PN, interestingly enough, such a set does not have to be explicitly computed to check the relation. We propose an algorithm based on linear programming to check the cyclic dependence relation.

Algorithm 1 Checking cyclic dependence relation using linear programming

INPUT: $\boldsymbol{A}$: the incidence matrix of a PN, $\mathbb{S}$: the set of FCSs to be checked.

OUTPUT: returns TRUE if there exists a cyclic dependence relation in $\mathbb{S}$, FALSE otherwise.

```
1:  for all covers $S$ of $\mathbb{S}$ do
2:     $dependent \Leftarrow$ TRUE
3:     for all $t_i \in S$ do
4:        $LP \Leftarrow (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{x} = 0) \cap (\boldsymbol{x} \geq 0) \cap (x_i > 0) \cap (x_j = 0, \forall j, t_j \in \mathbb{S} \backslash S)$
5:        if $LP \neq \emptyset$ then
6:           $dependent \Leftarrow$ FALSE
7:           break
8:        end if
9:     end for
10:    if $dependent$ = TRUE then
11:       return TRUE
12:    end if
13: end for
14: return FALSE
```

Given a PN N, its incidence matrix $\boldsymbol{A}$, and a set $\mathbb{S}$ of FCSs of N to be checked, the algorithm iterates through all possible covers of $\mathbb{S}$ till one cover leads to a cyclic dependence relation. For each cover, a feasibility problem of linear programming is constructed. As proved in Theorem 2, a solution to the feasibility problem provides a counterexample to the dependence relation. If no solution is found, the dependence relation holds. Since the task of linear programming has a polynomial-time complexity, checking that a cover S leads to a cyclic dependence relation for $\mathbb{S}$ can be done in polynomial-time.

Theorem 2. A transition t_i is dependent on a set S of transitions if and only if the following linear system has no solution:

$$\begin{aligned}
&\boldsymbol{A}^{T}\boldsymbol{x} = 0\\
&\boldsymbol{x} \geq 0\\
&x_i > 0\\
&\forall j, t_j \in S, x_j = 0
\end{aligned}$$

Proof:

(if): We prove the contrapositive. If $t_i \rightarrowtail S$ does not hold, by definition, there exists a T-invariant $\boldsymbol{x} \in \mathbb{N}^{|T|}$, such that $x_i \geq 1$ and for each $t_j \in S, x_j = 0$.

(only-if): We prove the contrapositive. Since the incidence matrix $\boldsymbol{A}$ is an integer matrix, if there exists a real vector that satisfies all the constraints, then there exist a rational vector $\boldsymbol{x}$ also satisfying the constraints. Let θ be a common multiple of all the denominators of the elements of $\boldsymbol{x}$ and let $\boldsymbol{x}' = \theta\boldsymbol{x}$. By definition, $\boldsymbol{x}'$ is a T-invariant, and $\boldsymbol{x}' \geq 0, x'_i > 0$, for each $t_j \in S, x'_j = 0$. Thus, $\boldsymbol{x}'$ is a counterexample for $t_i \rightarrowtail S$. □

The complexity of checking a cyclic dependence using Algorithm 1 is exponential on the number of FCSs. This comes from the need to check explicitly all possible covers of the set $\mathbb{S}$. The next section

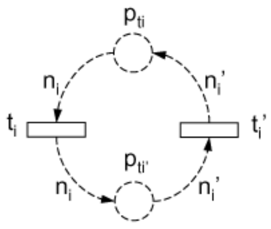


Figure 5. Illustration of the proof of the rank theorem

presents a more efficient way to establish unschedulability based on checking the rank of the incidence matrix of PNs.

6. The Rank Theorem

The connection between behavioral properties of PNs and the rank of the incidence matrix was first observed for free-choice PNs, see [8]. The rank property shown there states that a PN of a specific class of free-choice nets has a live and bounded marking if and only if the number of conflict clusters exceeds the rank of the incidence matrix by one. In our terminology, conflict clusters of free-choice nets are either free-choice sets or transitions which are not in conflict with any other transition. Since we assume that free-choice sets have exactly two transitions, the number of all transitions equals the number of conflict clusters plus the number of conflict sets. In other words, the number of conflict clusters can be written as $|T| - k$, where T is the set of all transitions and k is the number of free-choice sets. So, the rank condition translates to $\mathrm{rank}(\boldsymbol{A}) = |T| - k - 1$. The same property is shown in [8] to be sufficient for the existence of a live and bounded marking in case of certain non-free-choice nets.

Our setting is quite different to the one considered in [8]. The class of nets is different, and so is the behavioral property under investigation. Nevertheless, the rank condition provided in Theorem 3 looks pretty similar to the characterization mentioned above.

Theorem 3. (Rank theorem)
If there exists a schedule of a PN which involves k FCSs, then $\mathrm{rank}(\boldsymbol{A}) \leq |T| - k - 1$.

Proof:
Let sch be a schedule of a PN $N(P, T, F)$ for some marking M of N. First, we construct a PN N' by adding places and edges to N. Then, we show that $\mathrm{rank}(\boldsymbol{A}') \leq |T| - 1$. Finally, we show that $\mathrm{rank}(\boldsymbol{A}') = \mathrm{rank}(\boldsymbol{A}) + |\mathbb{S}|$, where $\mathbb{S}$ is the set of FCSs involved in sch. Thus, $\mathrm{rank}(\boldsymbol{A}) \leq |T| - |\mathbb{S}| - 1$ follows.

1. We construct a PN $N' = (P', T, F')$. As illustrated in Figure 5, for each $FCS_i = \{t_i, t_i'\} \in \mathbb{S}$, add 2 places $p_{ti}, p_{ti'}$ and 4 edges $F'(p_{ti}, t_i) = n_i, F'(t_i, p_{ti'}) = n_i, F'(p_{ti'}, t_i') = n_i', F'(t_i', p_{ti}) = n_i'$.

2. We determine n_i, n_i'. Since for each edge (u, v) in sch there exists a directed path from r to u and from v to r, and the number of edges is finite, there exists a *closed walk* sch' in sch with finite length, such that sch' visits each edge in sch at least once. Let c_i, c_i' be the number of times of t_i, t_i' of FCS_i appears in sch', respectively. Obviously, $c_i \geq 1, c_i' \geq 1$. Let c be a common multiple of $\{c_1, c_1', \ldots, c_k, c_k'\}$, and $n_i = c/c_i, n_i' = c/c_i'$.

3. We show $\mathrm{rank}(\boldsymbol{A'}) \leq |T| - 1$. First, we define

$$M'(p) = \begin{cases} M(p) & \text{if } p \in P \cap P' \\ c & \text{otherwise} \end{cases}$$

   It is easy to check sch' corresponds to a firing sequence from M' to M'. The firing count vector of a cyclic firing sequence is a T-invariant. $\boldsymbol{A'}^{\mathrm{T}}\boldsymbol{x} = 0$ has non-null solution implies $\mathrm{rank}(\boldsymbol{A'}) \leq |T| - 1$.

4. We construct the incidence matrix $\boldsymbol{A'}$ of N'. For each $FCS_i = \{t_i, t'_i\}$, two column vectors $\boldsymbol{y_i}, \boldsymbol{y'_i}$ corresponding to $p_{ti}, p_{ti'}$, respectively, are added to the incidence matrix $\boldsymbol{A}$ of N. Let $Y = [\boldsymbol{y_1} \dots \boldsymbol{y}_{|\mathbb{S}|}]$ and $Y' = [\boldsymbol{y'_1} \dots \boldsymbol{y'}_{|\mathbb{S}|}]$. Then $\boldsymbol{A'} = [\boldsymbol{A}|Y|Y']$.

$$\begin{array}{c|ccccc}
 & & p_{ti} & & p_{ti'} & \\
\hline
 & \ddots & 0 & \ddots & 0 & \ddots \\
ti & \dots & -n_i & \dots & n_i & \dots \\
 & \ddots & 0 & \ddots & 0 & \ddots \\
ti' & \dots & n'_i & \dots & -n'_i & \dots \\
 & \ddots & 0 & \ddots & 0 & \ddots
\end{array}$$

   In vector $\boldsymbol{y_i}$, there are exactly two non-zero entries $-n_i, n_i$, corresponding to transition t_i, t'_i respectively. Similarly, in vector $\boldsymbol{y_i}'$, there are entries $n'_i, -n'_i$, corresponding to transition t_i, t'_i respectively.

5. We show $\mathrm{rank}(\boldsymbol{A'}) = \mathrm{rank}(\boldsymbol{A}) + |\mathbb{S}|$.

   Since $\boldsymbol{y_i} + \boldsymbol{y_i}' = 0$, $\boldsymbol{y_i}, \boldsymbol{y_i}'$ are linearly dependent. Thus, $\mathrm{rank}(\boldsymbol{A'}) = \mathrm{rank}([\boldsymbol{A}|Y])$.

   We show that each column vector $\boldsymbol{y_i}$ of Y is linearly independent with respect to other column vectors in $[\boldsymbol{A}|Y]$. We prove by contradiction. It is known that vector $\boldsymbol{y_i}$ is linearly dependent with respect to other columns vectors in $[\boldsymbol{A}|Y]$ if and only if $\boldsymbol{y_i}$ is a linear combination of other vectors.

$$\boldsymbol{y_i} = \sum_{1 \leq j \leq |P|} a_j \boldsymbol{A}_j + \sum_{\substack{1 \leq k \leq |\mathbb{S}| \\ k \neq i}} b_k \boldsymbol{y_k}$$

   where $\boldsymbol{A} = [\boldsymbol{A}_1 \dots \boldsymbol{A}_{|P|}]$, $a_j \in Q$, $b_k \in Q$. Let vector $\boldsymbol{a} = [a_1 \dots a_{|P|}]^{\mathrm{T}}$, and $\boldsymbol{z} = \boldsymbol{y_i} - (\sum_{\substack{1 \leq k \leq |\mathbb{S}| \\ k \neq i}} b_k \boldsymbol{y_k})$. Then $\boldsymbol{z} = \boldsymbol{A}\boldsymbol{a}$. For each T-invariant $\boldsymbol{x}$ of N,

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{x} = 0 \Rightarrow \boldsymbol{a}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{x} = 0 \Rightarrow (\boldsymbol{A}\boldsymbol{a})^{\mathrm{T}}\boldsymbol{x} = 0 \Rightarrow \boldsymbol{z}^{\mathrm{T}}\boldsymbol{x} = 0.$$

   We then show that $\boldsymbol{z}^{\mathrm{T}}\boldsymbol{x} = 0$ implies the existence of a cyclic dependence relation. Note that the non-zero entries in $\boldsymbol{z}$ corresponds to transitions in FCSs, and for a FCS $\{t_a, t_b\}$, $z_a > 0$ if and only $z_b < 0$. Denote $\mathbb{S}' \subseteq \mathbb{S}$ the set of FCS that has non-zero entries in $\boldsymbol{z}$. Obviously, $FCS_i \in \mathbb{S}'$. Let $S = \{t_a | z_a > 0\}$, $S' = \{t_b | z_b < 0\}$. It is easy to see that S is a cover of $\mathbb{S}'$, and $S' = \mathbb{S}' \backslash S$. Since

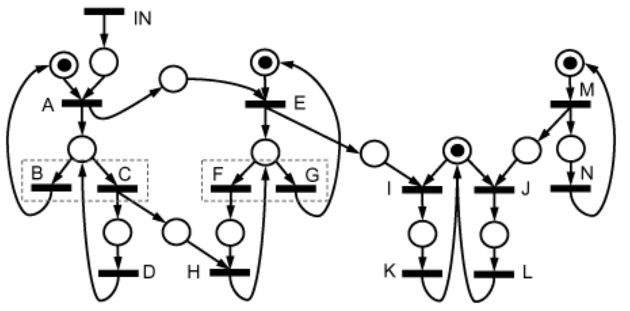


Figure 6. A PN whose unschedulability can be proved by Theorem 1, but not by Theorem 3.

$\boldsymbol{z}^{\mathrm{T}}\boldsymbol{x} = 0$, for each transition $t_i \in S$, $x_i > 0$ implies that there exists at least one transition $t_j \in S'$ such that $x_j > 0$. That is, for each transition $t \in S, t \rightarrowtail S'$. Since $S' = \mathbb{S}' \backslash S$, $\mathbb{S}'$ is in cyclic dependence relation. Thus, the schedule sch involves a cyclic dependent FCS. This contradicts Theorem 1.

Since each column vector $\boldsymbol{y_i}$ of Y is linearly independent with respect to other column vectors in $[\boldsymbol{A}|Y]$, where $Y = [\boldsymbol{y_1} \ldots \boldsymbol{y}_{|\mathbb{S}|}]$, $\mathrm{rank}([\boldsymbol{A}|Y]) = \mathrm{rank}(\boldsymbol{A}) + |\mathbb{S}|$ follows.

□

Theorem 3 is equivalent to the statement: a schedule of a PN involves at most $(|T| - \mathrm{rank}(\boldsymbol{A}) - 1)$ FCSs. It implies that if $\mathrm{rank}(\boldsymbol{A}) > |T| - |\mathbb{S}| - 1$, where $\mathbb{S}$ is the set of all FCSs of a PN N, then for any marking M of N, there exists no schedule of (N, M) that involves all FCSs of the net.

Consider the PN shown in Figure 3. $|T| = 9, \mathrm{rank}(\boldsymbol{A}) = 7, |\mathbb{S}| = 2$, thus no schedule involves all FCSs. For the PN shown in Figure 4, $|T| = 15, \mathrm{rank}(\boldsymbol{A}) = 11, |\mathbb{S}| = 4$, thus no schedule involves all FCSs.

Since computing the rank of a matrix has a polynomial-time complexity, checking unschedulability by the rank of the incidence matrix is more efficient than by a cyclic dependence because the latter requires to iterate through all possible covers of a set of FCSs (potentially exponential).

However, the rank condition is weaker in establishing the unschedulability as is shown by the following example. Figure 6 shows a PN with $|T| = 15$, $\mathrm{rank}(\boldsymbol{A}) = 12$, and $|\mathbb{S}| = 2$. Thus, we can not prove unschedulability by the rank theorem. Note that the left part of the PN is identical to the PN shown in Figure 3. There exists a cyclic dependence relation in FCSs $\{B, C\}$, $\{F, G\}$. Thus, we can prove unschedulablity by the cyclic dependence theorem.

7. Experiments

In this section, we show that PNs generated from a wide class of real-life industrial applications are not schedulable, and our approach can be effectively applied to establish that. Note that to prove unschedulability of a PN based on Theorem 1, we need to assert that each schedule involves at least one cyclic dependent FCS. To prove unschedulability based on Theorem 3, we need to assert that each schedule involves all FCSs of a PN. We use some public available JPEG and MPEG codecs as our test bench. The codecs used in our experiments are modelled as Kahn process networks [9], [10].

7.1. Benchmarks

MPEG-2 decoder We use an MPEG-2 video decoder [17] developed by Philips. The system consists of 11 concurrent processes communicating through 45 channels. It was implemented with about 5,000 lines of YAPI [11] code, a system specification language based on C++. We perform schedulability analysis on 5 processes that implement the spatial compression decoding, the temporal compression decoding, and image generation. In total, the 5 processes have 10 channels and 16 interfaces (communicating ports with the environment). Each process contains an average of 17 communication primitives in 8 control structures. They were implemented with about 2,000 lines of code.

M-JPEG* encoder We use an M-JPEG* [13] encoder also developed by Philips. The source code is obtained through the SESAME [1] project public release. The system consists of 8 processes communicating through 18 channels. Each process contains an average of 11 communication primitives in 5 control structures. They were implemented with about 2,000 lines of YAPI code. The encoder supports RGB and YUV formats and dynamically adjusts quantization and Huffman tables based on collected statistics. We perform schedulability analysis on the entire system (instance *MJPEGenc* in Table 1).

XviD MPEG4 encoder We model a XviD MPEG4 video encoder based on the C source code from [3] and the model from [5], which was developed as a SESAME application. The encoder supports two frame types: I-frame and P-frame. It performs motion estimation analysis to determine whether an incoming frame will be treated as I-frame or P-frame. Consequently two types of frame will go through different processing paths. An I-frame will be split into macro-blocks and encoded independently. In a P-frame, a macro-block could be an intra-block, an inter-block, or an not-coded-block, depending the value of Sum of Absolute Differences (SAD). The granularity of tokens passing between processes is macroblock. We perform schedulability analysis on 9 processes with 15 channels and 6 interfaces (instance *MPEG4enc* in Table 1).

PVRG JPEG encoder We obtain the Stanford Portable Video Research Group (PVRG) JPEG codec source code from [2]. Based on JPEG baseline standard, our model consists of 10 processes and 21 channels. The functional core part consists of 4 processes implementing Discrete Cosine Transform (DCT), quantization, Huffman coding, and control. The granularity of tokens passing between processes is block. We preform schedulability analysis both on PNs generated from the model of the encoder (instance *JPEGenc2* in Table 1) and its functional core (instance *JPEGenc1* in Table 1).

Additionally, we create a set of instances to test the worst case performance of our approach. Each PN instance is a chain of free choices and contains no cyclic dependence. Finding a schedule for this kind of PNs is easy. However, to prove there is no cyclic dependent FCSs, the analyzer has to check all subsets of FCSs and all covers of each subset.

7.2. Results and Analysis

We implemented our schedulability analyzer in C. All experiments were run on a 3.0 GHz Intel Pentium CPU with 512 MB memory. Since no other schedulability analyzer is available, we compare its performance with a scheduler. The scheduler performs a schedulability analysis via heuristic construction of a schedule. Table 1 summarizes the experiment results of PNs modelling JPEG MPEG codecs.

Instance	Place	Tran.	Arc	FCS	Rank	Schedulable	Runtime		
							Check Rank	Check Dependence	Scheduling
JPEGenc1	26	27	64	6	21	N	$<$0.01s	0.19s	523.75s
JPEGenc2	67	68	167	14	57	N	$<$0.01s	0.54s	$>$24hr
MJPEGenc	117	124	330	25	108	N	$<$0.01s	0.04s	$>$24hr
MPEG2dec1	116	144	358	38	111	N	$<$0.01s	0.25s	$>$24hr
MPEG2dec2	115	106	309	8	97	Y	$<$0.01s	17.28s	6.91s
MPEG4enc	72	72	184	15	63	N	$<$0.01s	0.16s	$>$24hr

Table 1. Statistics of schedulability analysis of PN models of JPEG and MPEG codecs

Instance	Place	Tran.	Arc	FCS	Rank	Schedulable	Runtime		
							Check Rank	Check Dependence	Scheduling
choice3	6	9	20	3	5	Y	$<$0.01s	0.01s	$<$0.01s
choice4	7	11	25	4	6	Y	$<$0.01s	0.02s	0.01s
choice5	8	13	30	5	7	Y	$<$0.01s	0.05s	0.01s
choice6	9	15	35	6	8	Y	$<$0.01s	0.14s	0.04s
choice7	10	17	40	7	9	Y	$<$0.01s	0.45s	0.05s
choice8	11	19	45	8	10	Y	$<$0.01s	1.42s	0.09
choice9	12	21	50	9	11	Y	$<$0.01s	4.45s	0.29s
choice10	13	23	55	10	12	Y	$<$0.01s	14.06s	0.87s
choice11	14	25	60	11	13	Y	$<$0.01s	44.86s	1.06s
choice12	15	27	65	12	14	Y	$<$0.01s	141.55s	4.43s

Table 2. Statistics of schedulability analysis of PNs in the test suite

Our analyzer typically proves a PN is not schedulable within a second, while the scheduler often fails to terminate in 24 hours. Note that there are two instances modelling MPEG2 decoders, *MPEG2dec1* and *MPEG2dec2*. The former, generated from the original YAPI source code, is not schedulable. The second, generated from a modified source code, is schedulable. The modification removes the correlated control structures that result in cyclic dependence using the technique described in [4]. Note that our schedulability analyzer computes the minimum set of FCSs that has a cyclic dependence relation, once it proves a PN is unschedulable. The minimum set of FCSs provides useful information to find the correlated control structures.

Table 2 shows that for schedulable PNs, the run time of checking cyclic dependence grows exponentially as the number of FCSs increases. However, checking rank remains efficient. In fact, checking rank takes less than 10 milliseconds for all of our experiments.

Our schedulability analyzer is effective because there exist certain program structures in the codecs. We illustrate this using a Huffman coding process of a JPEG encoder. Figure 7 shows a simplified description of the process. The process first reads from a control process a header which includes all parameters necessary to perform Huffman coding on a block. Then it iterates through all blocks of a component in a Minimum Coded Unit (MCU). The Huffman coding and reading from a zigzag process are performed at the block level inside the loop. Since JPEG standard requires samples of a component must use the same Huffman coding table, and multiple component samples could be interleaved within a compressed data stream, it needs to update the Huffman table from time to time. Also note that the loop has a variable number of iterations. The vertical and horizontal sampling factors could be different

```
PROCESS Huffman(
              In_DPORT Control_headerIn,
              In_DPORT Zigzag_blockIn,
              Out_DPORT Output)
{
  while(1) {
    READ_DATA(Control_headerIn, header, 1);
    Vi = getVSF(header);
    Hi = getHSF(header);
    Htable = getHtable(header);
    for(v=0; v<Vi; v++) {
      for(h=0; h<Hi; h++) {
        READ_DATA(Zigzag_blockIn, block, 1);
        block = HuffmanEncoding(block, Htable);
        WRITE_DATA(Output, block, 1);
} } } }
```

Figure 7. A simplified Huffman coding process.

for different components and only known at run-time. All of the above requires communications inside and outside a loop structure. The quantization process has a similar program structure to the Huffman coding process, because samples of a component are required to use the same quantization table and processing and communication data is performed at block level. The control process synchronizes the two concurrent processes such that a block is processed in the quantization process and later in a Huffman coding process with the set of parameters (e.g. vertical and horizontal sampling factors) of the same component.

Synchronized communications inside and outside a loop structure are common in the codecs. However, the controls of loops are abstracted as non-deterministic free choices in a PN. These two factors cause unschedulability that can be efficiently checked by our structural analysis. Note that this particular code structure is just a special case that results in cyclic dependence.

8. Conclusion and Future Work

We defined the notion of PN schedulability and provided several sufficient conditions for checking unschedulability using linear algebra and linear programming techniques. Our preliminary experimental results indicate that these techniques effectively and efficiently detect unschedulability of PNs for practical examples. They also identify the correlated choices that cause unschedulability.

We believe there are some interesting open problems in the research area covered in this paper. In particular:

- Does a sufficient condition exist that is tighter than the ones presented here for the class of general PNs?
- How much do we need to restrict the class of PNs to yield a necessary and sufficient condition for schedulability?
- Our results can be used to represent the unbounded parts of the schedule in an implicit but finite form. Hence, we believe an implementation could be derived that guarantees boundedness as long as a system functions in a “good” part of the schedule and raising the flag when it enters a potentially unbounded part. This approach would significantly extend the applicability of quasi-static scheduling for practical applications.

References

[1] The SESAME Software Project, URL: `http://sesamesim.sourceforge.net`.

[2] Stanford PVRG JPEG codec, URL: `http://www.dclunie.com/jpegge.html`.

[3] XviD MPEG-4 video codec, URL: `http://www.xvid.org`.

[4] Arrigoni, G., Duchini, L., Lavagno, L., Passerone, C., Watanabe, Y.: False Path Elimination in Quasi-Static Scheduling, *Proceedings of the Design Automation and Test in Europe Conference*, March 2002.

[5] Broekhof, P., Roosen, N., Verhoef, J., Jun, W.: Modeling XviD as a Kahn Process Network, a SESAME Application Design Document, URL: `http://staff.science.uva.nl/~andy/apps/xvid.pdf`.

[6] Cortadella, J., Kondratyev, A., Lavagno, L., Massot, M., Moral, S., Passerone, C., Watanabe, Y., Sangiovanni-Vincentelli, A.: Task generation and compile-time scheduling for mixed data-control embedded software, *DAC '00: Proceedings of the 37th Conference on Design Automation*, 2000.

[7] Cortadella, J., Kondratyev, A., Lavagno, L., Passerone, C., Watanabe, Y.: Quasi-static scheduling of independent tasks for reactive systems, *IEEE Transactions on Computer-Aided Design*, **24**(10), 2005, 1492–1514.

[8] Desel, J., Esparza, J.: *Free choice Petri nets*, vol. 40 of *Cambridge Tracts In Theoretical Computer Science*, Cambridge University Press, New York, NY, USA, 1995.

[9] Kahn, G.: The semantics of a simple language for parallel programming, *Information Processing*, Aug 1974.

[10] Kahn, G., MacQueen, D. B.: Coroutines and networks of parallel processes, *Information Processing*, Aug 1977.

[11] de Kock, E. A., Smits, W. J. M., van der Wolf, P., Brunel, J.-Y., Kruijtzer, W. M., Lieverse, P., Vissers, K. A., Essink, G.: YAPI: Application Modeling for Signal Processing Systems, *DAC '00: Proceedings of the 37th Conference on Design Automation*, 2000.

[12] Lee, E. A., Messerschmitt, D. G.: Static scheduling of synchronous data flow programs for digital signal processing, *IEEE Trans. Comput.*, **36**(1), 1987, 24–35.

[13] Lieverse, P., Stefanov, T., van der Wolf, P., Deprettere, E.: System level design with SPADE: an M-JPEG case study, *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, Nov 2001.

[14] Lin, B.: Software synthesis of process-based concurrent programs, *DAC '98: Proceedings of the 35th ACM/IEEE Conference on Design Automation*, 1998.

[15] Memmi, G., Roucairol, G.: Linear Algebra in Net Theory., *Lecture Notes in Computer Science: Net Theory and Applications, Proc. of the Advanced Course on General Net Theory of Processes and Systems, Hamburg, 1979* (Brauer, W., Ed.), 84, Springer-Verlag, Berlin, Heidelberg, New York, 1980.

[16] Sgroi, M., Lavagno, L., Watanabe, Y., Sangiovanni-Vincentelli, A.: Synthesis of embedded software using free-choice Petri nets, *DAC '99: Proceedings of the 36th ACM/IEEE Conference on Design Automation*, 1999.

[17] van der Wolf, P., Lieverse, P., Goel, M., Hei, D. L., Vissers, K.: An MPEG-2 decoder case study as a driver for a system level design methodology, *CODES '99: Proceedings of the 7th International Workshop on Hardware/Software Codesign*, 1999.