Identification of Discrete Event Systems Unobservable Behaviour by Petri nets using Language Projections

Jeremie Saives, Gregory Faraut, Jean-Jacques Lesage

Identification of Discrete Event Systems Unobservable Behaviour by Petri nets using Language Projections

Jeremie Saives[1], Gregory Faraut[1] and Jean-Jacques Lesage[1]

***Abstract*— The aim of behavioural identification of Discrete Event Systems is to build, from a sequence of observed inputs/outputs events, a model that exhibits both the direct relations between inputs and outputs events (*i.e.* the reactive or observable behaviour of the system) and the internal state evolutions (*i.e.* the unobservable behaviour). Once the observable behaviour has been modelled by an Interpreted Petri net, the method proposed in this paper aims at discovering the unobservable part from a firing sequence of previously discovered observable transitions. The principle is to project the firing sequence on subalphabets to discover specific patterns that are characteristic of dependancy relationships between the transitions. These relationships can be translated into Petri net structure fragments that will be assembled to form the final model. A parametric algorithm is proposed to conduct the discovery, the choice of the minimal degree of places as a parameter being motivated by the reduction of the search space and the fitness of the final model.**

I. Introduction

Mathematical models of systems are required for multiple applications, namely simulation, control, performance evaluation, fault diagnosis or reverse engineering. Whereas these models are often built off expert knowledge, *system identification* consists in building a model of the behaviour of the system from finite observations of the system, more precisely of the inputs/outputs evolution. For a reactive Discrete Event System, for instance a process and a controller in a closed-loop, the behaviour to be reproduced can be split as following:

- Observable behaviour, *i.e.* direct output changes depending on input changes
- Unobservable behaviour, *i.e.* evolutions of the internal state (and variables) of the system without changes of observable data (inputs and outputs)

Therefore, an identification algorithm should provide a model expressing both input/output causal relationships and internal state evolutions due to input changes. Petri nets (PN) provide the semantics to express sequentiality, choices and parallelism common to DES, and the subclass of Interpreted Petri nets (IPN) adds input/output interpretation to transitions and places, thus being a natural choice of model for the identification problem.

This work is located in the continuation of [6]. The authors of [6] provide a statistical approach to discover the observable behaviour as IPN fragments from an input/output observed sequence. This sequence is also converted into a firing sequence on the alphabet of observable transition. Therefore, the problem addressed by this paper is to discover the unobservable behaviour from such a firing sequence, and complete the IPN fragments by adding connecting places.

Very few contributions consider the distinction between observable and unobservable behaviour. A different understanding of the distinction is however presented in [5] or [3]: the observable behaviour is a PN model that is assumed to be well-known, and the unobservable behaviour consists in silent (ε-labelled) transitions, that are interpreted as unobservable faults. Identification as presented in these work consists in adding these unobservable fault transitions to a model, but no clue is given as to how this model was designed or identified, and no links with inputs or outputs are presented.

Furthermore, it is assumed in the identification problem that the whole behaviour of the system can not be observed in a finite time. The identified model can contain additional behaviour that was not observed. The identification problem has therefore to be distinguished from the *synthesis* problem, which aims at reproducing exactly the observed behaviour. This synthesis problem has been dealt with extensively. Given a finite observed language and considering any non-observed word as a counter-example, the authors of [8] or [4] propose to solve an Integer Linear Problem to discover a generalized, possibly unbounded PN reproducing exactly every word of the language up to a length k. Works such as [1] based on the region theory propose to compute a PN whose reachability graph isomorphic to a given transition system. An extensive literature review of identification and synthesis methods can be found in [2].

A method to discover the unobservable behaviour from a firing sequence is proposed in [7], based on the study of consecutive pairs of events in the sequence, leading to the impossibility of finding long-term dependencies, *i.e.* memory effects of the system. In [10], a method to solve this issue is proposed, based on the discovery of the T-invariants of the net, but an adjustment of the discovered net is required to account for long-term dependencies. These methods are inspired from the field of *workflow* or *process mining*. Such process models are to be discovered from a log of sequences named *cases*, each case being one possible execution of a business process. [12] proposed the α-algorithm, a discovery algorithm based on binary relationships between consecutive events, and returning Workflow nets (WF-Nets), a specific subclass of Petri nets with only one input and one output place. It therefore does not account for cyclic behaviours and the question of the initial marking is irrelevant. Extensions to this work are multiple, namely the α^{++}-algorithm [14]

[1]LURPA, ENS Cachan, Univ Paris-Sud, F-94235 Cachan, France
firstname.lastname@lurpa.ens-cachan.fr

discovers indirect dependencies, but is based on a ruleset accounting for an exhaustive list of cases and lacks genericity. Multiple algorithms used for workflow mining are summarized in [13].

Even though the aim of these work is not to solve a synthesis problem, a good fitness of the obtained model remains an objective. The model should not underfit the behaviour of the sequence (*i.e.* allow too much excessive behaviour), or overfit it(*i.e.* new observations might not fit in the model), as presented in [11]. For DES, this criterion of fitness can be expressed by the size of the exceeding language, *i.e.* the language that can be generated by the identified model but is not found in the observed behaviour.

In this paper, a new method to discover the unobservable behaviour of a DES as a Petri net from a single sequence, hence without building a language, is presented. It aims at finding all dependencies between transitions, based on relationships between transitions that can be discovered using projections of the sequence on subalphabets. Reducing the exceeding language of the net to improve the fitness is also an objective. The initial marking of the discovered PN is also computed on the fly. Section 2 recalls the notions of Petri nets. Section 3 states the problem based on the previous results of [6], and gives an overview of the method. Section 4 establishes the relationship between transitions to be discovered using projections of the sequence, and how they can lead to the discovery of unobservable places. Section 5 presents and justifies the searching strategy and algorithm, which is then applied to one example, and finally Section 6 concludes.

II. Background on Petri nets

This section presents the basic concepts and notations of ordinary and interpreted Petri nets used in this work.

Definition 1. *An ordinary Petri net structure G is a bipartite digraph represented by the 4-tuple $G = (P, T, I, O)$ where: $P = \{p_1, p_2, \ldots, p_{|P|}\}$ and $T = \{t_1, t_2, \ldots, t_{|T|}\}$ are finite sets of vertices named places and transitions respectively; $I(O) : P \times T \to \mathbb{N}$ is a function representing the edges going from places to transitions (from transitions to places).*

For a place p_i, the set of pre(post)-transitions $\{t_j \in T, I(O)(p_i, t_j) = 1\}$ will be written ${}^\bullet p_i (p_i^\bullet)$.

A marking function $M : P \to \mathbb{Z}^+$ represents the number of tokens residing inside each place; it is usually expressed as a $|P|$-entry vector. $\mathbb{Z}^+$ is the set of nonnegative integers. If $\mathbb{Z}^+$ is replaced by $\{0, 1\}$, there is at most one token residing in any place, and the net is *1-bounded* (or *safe*).

Definition 2. *A Petri net system or Petri net (PN) is the pair $N = (G, M_0)$, where G is a PN structure and M_0 is an initial marking.*

In a PN system, a transition t_j is enabled at marking M_k if $\forall p_i \in P, M_k(p_i) \geq I(p_i, t_j)$, written $M_k \xrightarrow{t_j}$; an enabled transition t_j can be fired reaching a new marking M_{k+1}, written $M_k \xrightarrow{t_j} M_{k+1}$. It can be computed as $M_{k+1} = M_k + Cu_k$ where $u_k(j) = 1; u_k(i) = 0, i \neq j$. This equation is called the PN state equation. The reachability set of a PN is the set of all possible reachable markings from M_0 firing only enabled transitions; this set is denoted by $R(G, M_0)$.

If $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_3} \ldots \xrightarrow{t_k} M_k$, then $w = t_1 t_2 t_3 \ldots t_k$ is a firing sequence leading to marking M_k, and written $M_0 \xrightarrow{w} M_k$. Firing sequences enabled by the net are words over T.

Definition 3. *Let $G = \{P, T, I, O\}$ be a PN structure and M_0 an initial marking. The language $L(G, M_0)$ generated by (G, M_0) is defined by*

$$L(G, M_0) = \{w \in T^* | \exists M \in R(G, M_0),\ M_0 \xrightarrow{w} M\}$$

To distinguish the observable part from the non-observable, and add inputs/outputs information to observable transitions and places, the definition of the extension of PNs to Interpreted Petri nets is given here:

Definition 4. *An Interpreted Petri net system (IPN) $Q = (G, M_0, \mathbb{U}, \Sigma, \lambda, \mathbb{Y}, \varphi)$ is based on an ordinary PN system (G, M_0) to which are added:*

- *$\mathbb{U}$ the known input alphabet*
- *$\Sigma = \{\uparrow u_i, \downarrow u_i \mid u_i \in \mathbb{U}\}$ the set of events.*
- *$\lambda : T \to \{0, 1\}$ the labelling function of transitions. $\forall t_i \in T, \lambda(t_i) = F_i(\mathbb{U}) \bullet G_i(\Sigma)$ where:*
  - *$F_i : \mathbb{U} \to \{0, 1\}$ is a boolean function depicting the sufficient conditions on the levels of the inputs to fire t_i*
  - *$G_i : \Sigma \to \{0, 1\}$ is a boolean function depicting the sufficient conditions on the input events to fire t_i*

  $\lambda(t_i) = 1$ iff $F_i(\mathbb{U}) = 1 \wedge G_i(\Sigma) = 1$
- *$\mathbb{Y}$ the known output alphabet*
- *$\varphi : R(G, M_0) \to \{0, 1\}^{|\mathbb{Y}|}$ the output function that returns the value of the outputs given a marking of the net.*

In this work, some additional properties of places will be considered:

Definition 5. *A place p is called selfloop-free iff ${}^\bullet p \cap p^\bullet = \emptyset$*

Definition 6. *Let p be a place. $|{}^\bullet p|$ is called the in-degree of p, while $|p^\bullet|$ is called the out-degree of p. In the remainder of this work, the minimal degree (resp maximal degree) of p is defined as the integer:*

$Dmin(Dmax)(p) = \min(\max)(|{}^\bullet p|, |p^\bullet|)$.

The place p will be said to belong to the class $\mathbb{D}(Dmin, Dmax)$.

III. Problem formulation and principle of the proposed approach

The principle of behavioural identification of a system, illustrated by Figure 1, is explained below:

The system to be identified consists of a process and a controller in a closed-loop (Fig 1-[a]). The sensors of the process are the inputs of the controller whereas actuators

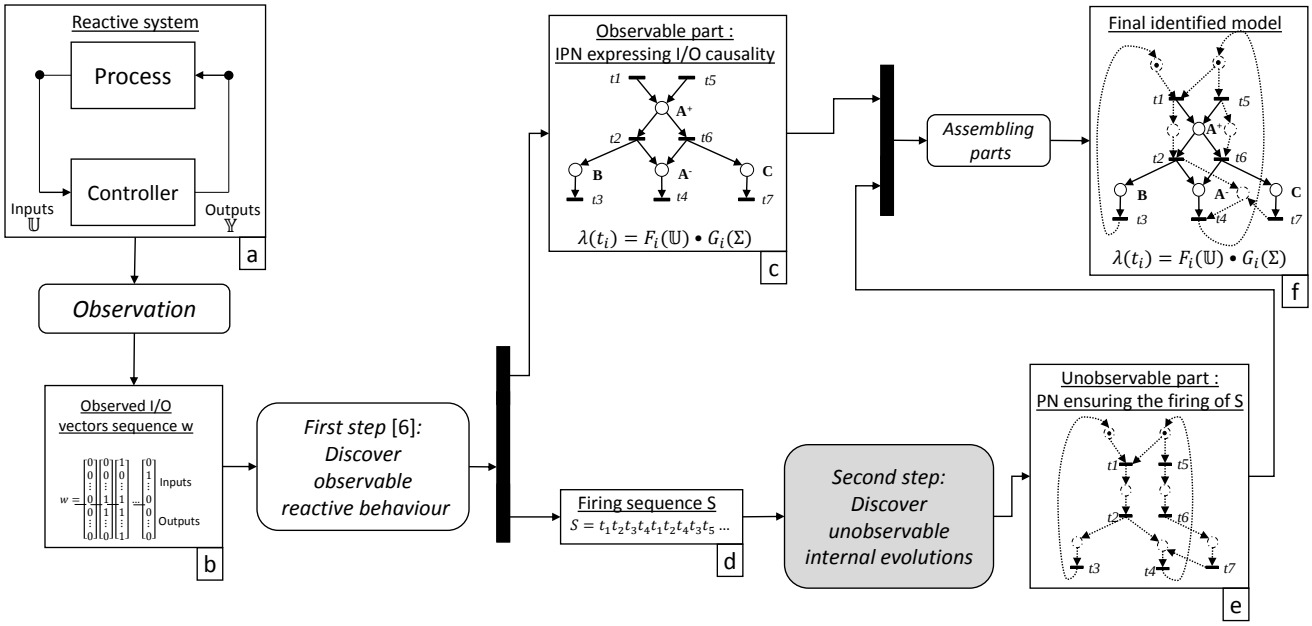


Fig. 1. Principle of behavioural identification of Discrete Event Systems in two steps

are its outputs; both inputs and outputs are assumed to be binary. At the end of every cycle of the controller, multiple inputs and outputs might have changed. The result of the observation is therefore a sequence of I/O vectors w (Fig 1-[b]). The goal of identification is to produce an IPN $Q = ((P, T, I, O), M_0, \mathbb{U}, \Sigma, \lambda, \mathbb{Y}, \varphi)$ modelling the behaviour of the system from w only. $\mathbb{U}, \Sigma, \mathbb{Y}$ are already known (inputs, input events and outputs). The behaviour of the system can be split into two parts:

- The reactive behaviour. Outputs can react directly to changes in the inputs; identifying this behaviour consists in discovering causal relationships between inputs and outputs. This behaviour is called *observable*, because output changes are observed.
- The internal behaviour. Input changes can provoke a change of internal variables of the controller without any output evolution, as is the case when delays or memory effects are implied. This behaviour is called *unobservable*.

Behavioural identification of a DES consists in discovering first the reactive behaviour as IPN fragments, then complete with fragments modelling the internal behaviour.

A. First step: Observable behaviour

This step is presented in [6]; it is based on a statistical approach, and produces two results:

- Observable PN structure fragments are constructed (Fig 1-[c]). They are composed of places P_{Obs} (one for each output, hence constructing φ), observable transitions T_{Obs} labelled with conditions on the inputs (λ), and edges (I_{Obs}, O_{Obs}) . For an observable place p mapped to an output y, ${}^\bullet p$ (resp $p^\bullet$) is the set of transitions expressing the conditions for y to be set to 1 (resp 0). Notably, all transitions are built in this step, $T = T_{Obs}$.
- The I/O sequence w is projected on the freshly built set of transitions T_{Obs}, resulting in a finite firing sequence $S \in T^*_{Obs}$ (Fig 1-[d]).

To complete the net, connexions between the fragments are to be added, and the initial marking remains to be identified as well. These missing elements can be discovered from the firing sequence S.

B. Second step: Unobservable behaviour

The unobservable behaviour consists mostly in internal states evolutions; multiple evolutions can take place between two output evolutions, *i.e.* between the firings of two observable transitions. Namely, counters can be increased, timers that delay output or input updates can be started, information to be used later is memorized, ... These internal behaviours are hard to compute from the sequence, but their explicit expression is not the goal; they can instead be agregated in *unobservable places*, without adding any silent transition. Exhaustivity in the description of non-causal relationships is of little interest for retro-modelling purposes.

Finding the unobservable behaviour consists therefore in discovering unobservable places with their edges, without adding any transition (Fig 1-[e]). It is defined as following:

Given $T_{Obs} = \{t_1, \ldots, t_n\}$ the set of observable transitions and $S \in T^*_{Obs}$ a finite firing sequence, compute a Petri net structure $G = \{P_{Unobs}, T_{Obs}, I_{Unobs}, O_{Unobs}\}$ and an initial marking M_0 such that:

- S is firable ($S \in L(G, M_0)$)
- G is 1-bounded

where $|P_{Unobs}|$ is unknown *a priori*.

Note that in this approach, observable places mirror the status of a binary output (0 or 1), and should therefore be

1-bounded. Hence the condition of 1-boundedness is set on the unobservable part as well.

After the computation of $P_{Unobs}, I_{Unobs}, O_{Unobs}$ and M_0 the final net Q is assembled with $P = P_{Obs} \cup P_{Unobs}$, $I = I_{Obs} \cup I_{Unobs}$ and $O = O_{Obs} \cup O_{Unobs}$ (Fig 1-[f]).

The method proposed in the remainder of this work aims at finding all possible unobservable places and edges satisfying the problem.

C. Principle of our method

The main idea is to make patterns between the transitions appear in the sequence S, that will later be translated into fragments of PN structure and assemble these to obtain the final model. Consider as an introductive example the following sequence on $T = \{t_1, t_2, t_3, t_4\}$, that could be the second result of the first phase:

Example 1. $S = t_1t_3t_4t_1t_4t_2t_1t_2t_4t_1t_4t_3t_1t_3t_4t_1t_2t_4$. *Two kinds of patterns are given below:*

1) $S = t_1 \dots t_4t_1t_4 \dots t_1 \dots t_4t_1t_4 \dots t_1 \dots t_4t_1 \dots t_4$
2) $S = t_1t_3 \dots t_1 \dots t_2t_1t_2 \dots t_1 \dots t_3t_1t_3 \dots t_1t_2 \dots$

In the first case, t_1 always occur between two occurences of t_4, and reciprocally. A PN strucure like the one in Figure 2(a) can represent this behaviour. The second case is almost like the first one, except this time one and only one of the two transitions t_2, t_3 occurs between two occurences of t_1, leading to a structure like the one in Figure 2(b).

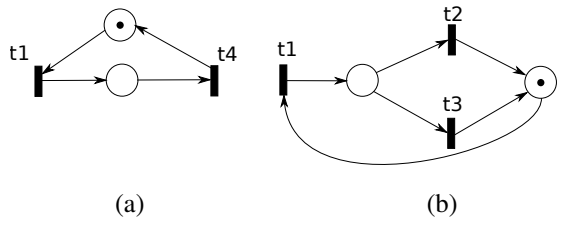


Fig. 2. (a) Structure for $t_1 \dots t_4t_1t_4 \dots t_1 \dots t_4$; (b) Structure for $t_1t_3 \dots t_1 \dots t_2t_1t_2 \dots t_1$

More generally, if Σ_i and Σ_j are two disjoint non-empty subalphabets of T_{Obs}, and $S = \dots t_i \dots t_j \dots t_i \dots t_j \dots$, with $t_i \in \Sigma_i$ and $t_j \in \Sigma_j$, then a generic PN structure fragment presented in Figure 3 can be added to the net.

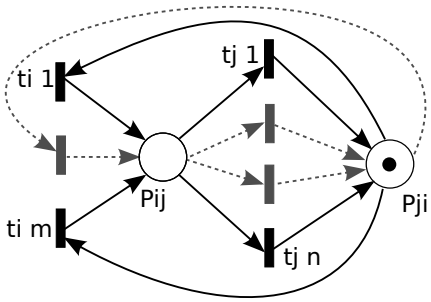


Fig. 3. A PN structure composed of two places p_{ij} and p_{ji} for $\Sigma_i = \{t_i^1, \dots, t_i^m\}$, $\Sigma_j = \{t_j^1, \dots, t_j^n\}$. t_i^1 is the first transition that is fired

To discover such patterns, the firing sequence can be projected on $\Sigma_i \cup \Sigma_j$. However, the relevant couples of subalphabets are *apriori* unknown. A full exploration would consist in checking all couples of disjoint partitions of T_{Obs}. If $|T_{Obs}| = n$, there are $2^n - 2$ partitions of T_{Obs} concerned (excluding $\emptyset$ and T_{Obs}), and the number of couples to be studied is $3^{n-2} + \sum_{k=0}^{n-3} 3^k(2^{n-k-1} - 1)$, hence an exponential complexity.

To limit the exploration, the size of the projecting alphabets is introduced as a parameter. As will be presented in Section 5, looking for relationships between smaller alphabets reduces the space of search and limits the excessive language of the net.

IV. Finding relationships between transitions using projections

The aim of this section is to define the relationships between transitions that can emerge from the projections of the firing sequence, and how these relationships can lead to the addition of places to the net. The definition of a projector is being recalled below from [9]:

Definition 7. *Let Σ and Σ_p be two alphabets such that $\Sigma_p \subseteq \Sigma$; and $S \in \Sigma^*$ a firing sequence. The projector Π_{Σ_p} is defined by:*

$$\Pi_{\Sigma_p}(S) = \begin{cases} \varepsilon, \, if \;\; S = \varepsilon \\ \Pi_{\Sigma_p}(a)t, \, if \;\; S = at, t \in \Sigma_p \\ \Pi_{\Sigma_p}(a), \, if \;\; S = at, t \notin \Sigma_p \end{cases}$$

$\Pi_{\Sigma_p}(S)$ is called the projection of S on Σ_p

This section presents two results: first, the possible outcomes of a random projection are studied, and the relationships between transitions are defined accordingly to the patterns discovered. Then, if a specific relationship is discovered, places that satisfy the problem stated in Section 3 can be added to the net.

A. Results of projections

Let Σ_i and Σ_j be two disjoint partial alphabets of T. In this section, t_i (t_j) will stand as a generic transition of $\Sigma_i(\Sigma_j)$. For instance, $\sigma = t_it_it_i$ is a sequence of any three transitions of Σ_i. Let Π be the projector on $\Sigma_i \cup \Sigma_j$. Then $\Pi(S)$ can only belong to one of the three following cases:

Case 1. $\Pi(S) = t_it_jt_it_jt_it_jt_it_jt_it_j \dots$ *In this case, two transitions observed consecutively never belong to the same alphabet, i.e.*

$$\nexists k \in \mathbb{N}^*, \Pi(S)_k \in \Sigma_i \wedge \Pi(S)_{k+1} \in \Sigma_i$$
$$\nexists k \in \mathbb{N}^*, \Pi(S)_k \in \Sigma_j \wedge \Pi(S)_{k+1} \in \Sigma_j$$

Definition 8. *Let $(\Sigma_i, \Sigma_j) \in (2^T)^2$ be two alphabets satisfying Case 1. Σ_i and Σ_j are called mutually dependant, written $\Sigma_i \leftrightarrows \Sigma_j$.*

For instance, in Example 1, two mutual dependencies are discovered: $\{t_1\} \leftrightarrows \{t_4\}$ and $\{t_1\} \leftrightarrows \{t_2, t_3\}$.

Case 2. $\Pi(S) = t_i t_i t_i t_j t_j t_i t_i t_j t_i t_j t_i \dots$ *In this case, at least once, two transitions from each alphabet have been observed consecutively, i.e.*

$$\exists k \in \mathbb{N}^*, \Pi(S)_k \in \Sigma_i \wedge \Pi(S)_{k+1} \in \Sigma_i$$
$$\exists k' \in \mathbb{N}^*, \Pi(S)_{k'} \in \Sigma_j \wedge \Pi(S)_{k'+1} \in \Sigma_j$$

No conclusion is possible. Both alphabets might be incomplete to discover a mutual dependency as in Case 1, or there is no dependency to be found between these transitions. Alphabets $\{t_3\}$ and$\{t_4\}$ from Example 1 fall in this case.

Case 3. $\Pi(S) = t_i t_i t_i t_j t_i t_j t_i t_j t_i t_i t_j \dots$ *In this case, at least once, two transitions observed consecutively belong to the same alphabet (here Σ_i), and if two consecutive transitions belong to the same alphabet, it is always the same one, i.e.*

$$\exists k \in \mathbb{N}^*, \Pi(S)_k \in \Sigma_i \wedge \Pi(S)_{k+1} \in \Sigma_i$$
$$\nexists k' \in \mathbb{N}^*, \Pi(S)_{k'} \in \Sigma_j \wedge \Pi(S)_{k'+1} \in \Sigma_j$$

Definition 9. *Let $(\Sigma_i, \Sigma_j) \in (2^T)^2$ be two alphabets satisfying Case 3. Σ_i is said to dominate Σ_j, written $\Sigma_i \to \Sigma_j$. The set of alphabets dominated by Σ_i is $Dom(\Sigma_i) = \{\Sigma_j \in 2^T, \Sigma_i \to \Sigma_j\}$*

To allow a transition of Σ_i to be fired again, the firing of a transition of Σ_j might be a prerequisite, but is not the only possibility. A possible situation of conflict is discovered, but the discovery is yet incomplete. In order to find a possible complet conflict, alphabets from $Dom(\Sigma_i)$ can be merged, and a new projection on $\Sigma_i \cup ((\Sigma_j, \Sigma_k) \in Dom(\Sigma_i)^2)$ can be studied. This new projection can give a result satisfying any of the three cases. For instance, in Example 1,$\{t_1\} \to \{t_2\}$ and $\{t_1\} \to \{t_3\}$; the extended projection on $\{t_1\} \cup (\{t_2\} \cup \{t_3\})$ leads to the discovery of a mutual dependency.

B. From relationships discovered between transitions to places discovery

The mutual dependency is a strong relationship that can be discovered between two sets of transitions. In each case of two alphabets being mutually dependant, two selfloop-free places can be added to the net following Theorem 1: the first alphabet becomes the pre-transitions of one place and the post-transitions of the other, and reciprocally with the second alphabet. Exactly one of the two places receives an initial token, depending on the first transition that must be fired according to the firing sequence.

Theorem 1. *Let $S \in T^*$ be a firing sequence, and (G, M_0) a 1-bounded Petri net such that $S \in L(G, M_0)$. Let Σ_i and Σ_j be two alphabets in 2^T ensuring $\Sigma_i \cap \Sigma_j = \emptyset$, and Π the projector on $\Sigma_i \cup \Sigma_j$. If $\Sigma_i \leftrightarrows \Sigma_j$, i.e.*

If $\forall k \in [\![1, \lfloor Card(\Pi(S))/2 \rfloor]\!]$,

$$\begin{cases} \Pi(S)_{2k-1} \in \Sigma_j \\ \Pi(S)_{2k} \in \Sigma_i \end{cases}$$

or

$$\begin{cases} \Pi(S)_{2k-1} \in \Sigma_i \\ \Pi(S)_{2k} \in \Sigma_j \end{cases}$$

Then the net G' defined by the addition of the following places p_{ij} and p_{ji} to G is 1-bounded and $S \in L(G', M_0)$

$${}^\bullet p_{ij} = \Sigma_i; p_{ij}^\bullet = \Sigma_j; \begin{cases} M_0(p_{ij}) = 0 \ if \ \Pi(S)_1 \in \Sigma_i \\ M_0(p_{ij}) = 1 \ if \ \Pi(S)_1 \in \Sigma_j \end{cases}$$

$${}^\bullet p_{ji} = \Sigma_j; p_{ji}^\bullet = \Sigma_i; \begin{cases} M_0(p_{ji}) = 0 \ if \ \Pi(S)_1 \in \Sigma_j \\ M_0(p_{ji}) = 1 \ if \ \Pi(S)_1 \in \Sigma_i \end{cases}$$

Proof. Suppose that $\Pi(S)_1 \in \Sigma_j$ (*i.e.* the first case, the reasoning being the same in the other case).

1-boundedness:
G is 1-bounded. It remains to prove that the places p_{ij} and p_{ji} are 1-bounded. They verify ${}^\bullet p_{ji} = p_{ij}^\bullet$, ${}^\bullet p_{ij} = p_{ji}^\bullet$, and $M_0(p_{ij}) + M_0(p_{ji}) = 1$. It ensures that $\forall M \in R(G', M_0), M(p_{ij}) + M(p_{ji}) = 1$, therefore G' is 1-bounded.

S $\in$ **L**(G', M_0):
Suppose that $S \notin L(G', M_0)$, and let t be the first transition that can not be fired. Since the transitions in $T - (\Sigma_i \cup \Sigma_j)$ have the same pre- and post-places in G' and in G, and t is firable in G, t must belong to $\Sigma_i \cup \Sigma_j$. Suppose that $t \in \Sigma_j$ (same reasoning for $t \in \Sigma_i$). Then p_{ij} is the only new place in ${}^\bullet t$ that can prevent t from firing, thus must be empty when t should be fired. Let k be an integer such that $\Pi(S)_{2k-1} = t$. Then, when $\Pi(S)_{2k-2} \in \Sigma_i$ was fired, p_{ij} was filled with a token. Since no other transition in Σ_j was fired between $\Pi(S)_{2k-2}$ and $\Pi(S)_{2k-1}$, p_{ij} still contains a token when t should be fired, leading to a contradiction. Therefore, $S \in L(G', M_0)$.

□

With Theorem 1, it is possible to use projections of S to discover and add all relevant selfloop-free places to G. For instance, applying it to the mutual dependencies of Example 1, $\{t_1\} \leftrightarrows \{t_4\}$ and $\{t_1\} \leftrightarrows \{t_2, t_3\}$, leads to the net structure of Figure 4.

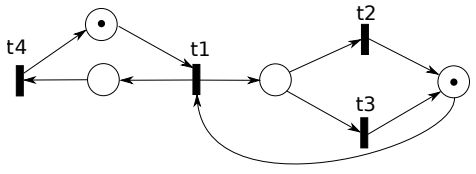


Fig. 4. A PN structure obtained for Example 1 that reproduces $S = t_1 t_3 t_4 t_1 t_4 t_2 t_1 t_2 t_4 t_1 t_4 t_3 t_1 t_3 t_4 t_1 t_2 t_4$

Note that the case of places with selfloops (which means projecting on non-disjoint alphabets) is out of the scope of this work. This section has shown that to discover places, projections on subalphabets can be studied. It remains to choose which subalphabets will be checked while looking for mutual dependencies, which is the purpose of the next section.

V. A STRATEGY FOR THE DISCOVERY OF UNOBSERVABLE PLACES BASED ON THEIR MINIMAL DEGREE

Whenever two alphabets are found in a mutual dependency, places whose in- and out-degrees match the sizes of the alphabets are added to the net. Before introducing the search algorithm, a reflexion is conducted on the relevant degrees to be studied, and leads to the choice of the minimal degree of places as a parameter of the algorithm. The reflexion is based on the exceeding language to be minimized, and on the information contained in the domination relationship.

A. Minimizing the exceeding language

In general, the language that can be generated by a net is infinite, due to the presence of loops. In order to make a comparison between the observed language, *i.e.* the language derived from the firing sequence, and the identified language, *i.e.* the language generated by the net, a length parameter n is introduced. On one hand, the observed language of length n, $L^n_{Obs}(S)$ is composed of all subsequences of S of length n:

Definition 10. *Let $S = s_1 s_2 \dots s_{|S|} \in T^*$ be a finite sequence. The observed language of length n, i.e. the set of words of length n generated by S is:*

$$L^n_{Obs}(S) = \bigcup_{1 \leq t \leq |S|-n+1} s_t . s_{t+1} \dots s_{t+n-1}$$

On the other hand, the identified language of length n, $L^n_{Id}(N)$ is composed of all words of length n that can be generated from any reachable marking of the net (and not only M_0):

Definition 11. *Let N be a PN. The identified language of length n, i.e. the set of words of length n generated by N is:*

$$L^n_{Id}(N) = \bigcup_{M \in R(N)} L^n_M(N)$$

where $L^n_M(N) = \{w \in T^* : |w| = n \ \wedge M \xrightarrow{w}\}$

The exceeding language can then be caracterized by the value of a similitude criterion:

Definition 12. *The language similitude criterion of length n between a sequence and a net that reproduces it is defined by*

$$C^n(N,S) = \frac{|L^n_{Id}(N)|}{|L^n_{Obs}(S)|}$$

It can be noted that $C^n(N,S) \geq 1$ because the net N is assumed to reproduce the sequence S. The goal is to make it as close as possible to 1.

B. Interest of the domination relationship

If a mutual dependency is to be found between two alphabets Σ_i and Σ_j, such that $|\Sigma_i| = m$ and $|\Sigma_j| = M$, $m < M$, then a domination will be discovered if a projection is made on Σ_i and any subalphabet of Σ_j, namely if the size of the subalphabet is also m:

Proposition 1. *Let Σ_i and Σ_j be two alphabets such that $\Sigma_i \leftrightarrows \Sigma_j$, and Σ' a non-empty subalphabet of Σ_j. Then, Σ_i dominates Σ', i.e. $\Sigma_i \rightarrow \Sigma'$.*

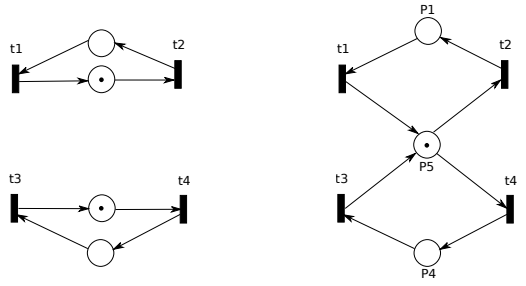
Proof. When projecting the sequence on $\Sigma_i \cup \Sigma'$, all t_i remain in the projection, but at least one $t_j \in \Sigma_j - \Sigma'$ will be missing. Therefore, at least two consecutives occurences of t_i will be observed, and $\Sigma_i \rightarrow \Sigma'$.

□

According to Proposition 1, any place in $\mathbb{D}(m, M)$ will be hinted at when looking for places in $\mathbb{D}(m, m)$. Therefore, when studying projections on two alphabets of size m, mutual dependencies will discover places in $\mathbb{D}(m, m)$, while domination will hint at places in $\mathbb{D}(m, M), \forall M > m$. The full exploration of $\mathbb{D}(m, M), M > m$ is therefore not required, and the main interesting parameter is the minimal degree m of the places to be discovered.

Places verifying $Dmin(p) = 1$ allow for the construction of loops, that will have parallel evolutions. Higher minimal degree places are synchronisation places, that restrict the free behaviour of the loops previously discovered. This phenomenon is illustrated in the next example:

Example 2. *Let $S = t_2 t_1 t_2 t_1 t_4 t_3 t_4 t_3 t_2 t_1 t_4 t_3$ be a sequence on $T = \{t_1, t_2, t_3, t_4\}$. Mutual dependecies $\{t_1\} \leftrightarrows \{t_2\}$ and $\{t_3\} \leftrightarrows \{t_4\}$ are discovered when looking for minimal degree 1 places, and two parallel loops are discovered (Figure 5(a)). However, in this case, $C^2(N,S) = 12/6 = 2$; for instance the behaviour $t_1 t_3$ is never observed in the sequence, although permitted by the net. When looking for minimal degree 2 places, the mutual dependency $\{t_1, t_3\} \leftrightarrows \{t_2, t_4\}$ is discovered, resulting in Figure 5(b), where a synchronisation is added. In this second case, $C^2(N,S) = 6/6 = 1$.*



(a) (b)

Fig. 5. (a) Degree 1 discovered net ; (b) Degree 2 discovered net

The strategy is therefore the following :

- Discover places belonging to $\mathbb{D}(1, M), \forall M \geq 1$, by studying $\mathbb{D}(1, 1)$
- If required (for example, if $C^2(N,S)$ is largely greater than one), discover places belonging to $\mathbb{D}(m, M)$, $\forall M \geq m, m > 1$, by studying $\mathbb{D}(m, m)$

The choice of the maximum value of m to be studied remains an open question and will be treated in further work

(it can be noticed that $m \leq \lfloor |T|/2 \rfloor$). Nevertheless, we noted in practice that $m = 2$ is enough to find a fitting model.

C. An algorithm for places discovery

The strategy previously defined suggests to discover places in the order given by their minimal degree. Algorithm 1 allow for the discovery of all places satisfying a given minimal degree:

Algorithm 1 Finding minimal degree m places

Require: S a sequence, T a set of transitions, m a degree
Ensure: Mutual dependencies and m-th degree places.

```
1: for (Σ_i, Σ_j) ∈ (2^T)^2, Σ_i ∩ Σ_j = ∅, |Σ_i| = |Σ_j| = n do
2:    Study Π_{Σ_i ∪ Σ_j}(S)
3:    if Case 1 Σ_i ⇆ Σ_j then
4:       Add places according to Theorem 1
5:    else if Case 3 Σ_i → Σ_j then
6:       Dom(Σ_i) ← Dom(Σ_i) ∪ {Σ_j}
7:    else if Case 3 Σ_j → Σ_i then
8:       Dom(Σ_j) ← Dom(Σ_j) ∪ {Σ_i}
9:    end if
10: end for
11:
12: for Σ_i ∈ 2^T, |Σ_i| = n  do
13:    while Dom(Σ_i) ≠ ∅ do
14:       NewDom(Σ_i) = {}
15:       for (Σ_j, Σ_k) ∈ Dom(Σ_i)^2, Σ_j ≠ Σ_k do
16:          Study Π_{Σ_i ∪ Σ_j ∪ Σ_k}(S)
17:          if Case 1 Σ_i ⇆ Σ_j ∪ Σ_k then
18:             Add places according to Theorem 1
19:          else if Case 3 Σ_i → Σ_j ∪ Σ_k then
20:             NewDom(Σ_i) ← NewDom(Σ_i) ∪ {Σ_j ∪ Σ_k}
21:          end if
22:       end for
23:       Dom(Σ_i) ← NewDom(Σ_i)
24:    end while
25: end for
```

Proposition 2. *Given an integer m, Algorithm 1 discovers all possible places P_i verifying $MinDeg(P_i) = m$, and such that S is firable.*

Proof. Places are only added when two alphabets satisfy the conditions of Theorem 1, therefore S is firable. If a mutual dependency is found, it is always $\Sigma_i \leftrightarrows \Sigma_j$, with $|\Sigma_j| = |\Sigma_j| = m$ for a dependency discovered at line 3, and $|\Sigma_j| \in [\![m, |T| - m]\!]$ for a mutual dependency discovered at line 16. The minimal degree of all places added is hence m. □

Proposition 3. *The complexity of Algorithm 1 is at worst exponential with the size of the alphabet.*

Proof. A full dependency including all transitions might exist. Suppose that there is a mutual dependency $\Sigma_i \leftrightarrows (T - \Sigma_i)$ to be discovered. For simplification, suppose that $|\Sigma_i| = m < |T| - m$. To discover this dependency, all subalphabets from size m to size $|T| - m$ will be tested, because they will all be dominated by Σ_i and be added to $NewDom(\Sigma_i)$, until finally the only candidate of size $|T| - m$ is proposed. In the worst case, when m=1, all subalphabets from size 1 to $|T| - 1$ will be studied, hence $2^T - 2$ subalphabets. □

It is worth noting that the case of such places is mostly theoretical and would not be representative of an actual industrial system or business process. To illustrate the algorithm, an example is proposed:

Example 3. *Let $T = \{t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$ be the alphabet, and $S = t_6t_1t_7t_4t_6t_1t_7t_4t_6t_1t_2t_4t_6t_0t_7t_3t_6t_0t_2t_3$ $t_6t_0t_2t_3t_6t_0t_7t_3t_6t_0t_7t_5t_4t_6t_0t_7t_3t_6t_0t_5t_2t_4t_6t_0t_5t_7t_4t_6t_1t_7$ be the sequence. The algorithm is run for $m = 1$. The following mutual dependencies are discovered:*
$\{t_6\} \leftrightarrows \{t_2, t_7\}$, $\{t_6\} \leftrightarrows \{t_1, t_3, t_5\}$, $\{t_6\} \leftrightarrows \{t_0, t_1\}$, $\{t_6\} \leftrightarrows \{t_3, t_4\}$, $\{t_1\} \leftrightarrows \{t_4, t_5\}$, $\{t_0\} \leftrightarrows \{t_3, t_5\}$.

12 places are added to build the PN. After the suppression of implicit places[1], the resulting net can be seen in Figure 6(a). It reproduces S, two places are initially marked to allow for the firing of $t_6 = s_1$, and one can notice that two loops are discovered and are parallel once t_6 has been fired. Notice also that a place links t_0 to t_3, although these two transitions are never observed consecutively. The long-term dependency here has been correctly captured. However, $C^2(N, S) = 32/17$, meaning that the identified language of length 2 is twice as big as the observed languaged. One can assume that the sequence was incomplete and this parallel behaviour is allowed, or that a higher degree dependency must be found to restrict parallel behaviour.

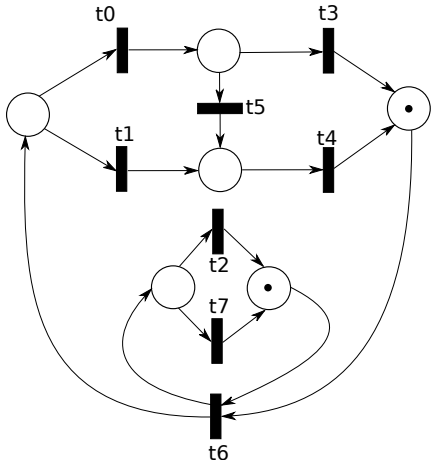


Fig. 6. Net discovered with only 1st degree places

The algorithm is rerun for $m = 2$, new mutual dependencies are discovered: $\{t_2, t_7\} \leftrightarrows \{t_0, t_1\}$, $\{t_2, t_7\} \leftrightarrows \{t_3, t_4\}$, $\{t_0, t_4\} \leftrightarrows \{t_5, t_6\}$, $\{t_3, t_4\} \leftrightarrows \{t_0, t_1\}$, leading to the addition of 8 new places. After the suppression of implicit places, the net can be seen in Figure 7. Notice that the

[1]An implicit place is a place whose removal does not alter the behaviour of the net system, *i.e.* leaves the reachability graph unmodified

behaviour of t_2 and t_7 has been restricted compared to Figure 6. Some 1st degree places (linking t_6, t_2 and t_7) became implicit with the addition of 2nd degree places, and have been removed. For this second net, $C^2(N,S) = 18/17$, the only non-observed behaviour generated by the net being t_2t_5. It is due to the short length of the sequence. The computing of this example took less than 50ms on a standard laptop.

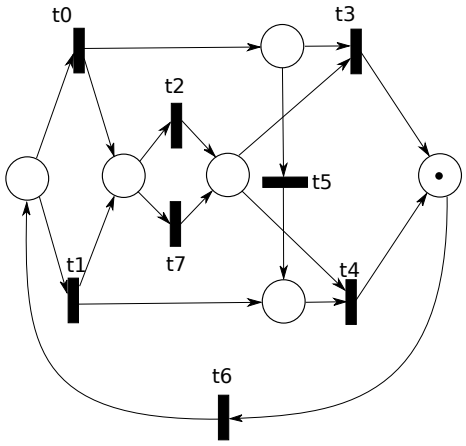


Fig. 7. Net discovered with 1st and 2nd degree places

VI. Conclusion

An efficient method has been proposed to discover Petri net models from a single firing sequence of transitions. This approach is generic, based on projections of the firing sequence, and can discover any 1-bounded Petri net system. Multiple models able to fire the sequence can be discovered, depending on the minimal degree of the places chosen. All dependencies, including long-term ones, can be easily discovered.

Future work will focus on determining if the value of the parameter of the algorithm can be fixed *a priori*, or if a stop criterion can be determined. In the latter case, the extension of this work to the case of places with selfloops seems promising. Indeed, if a selfloop is added to a place, it restricts the firing of the concerned transition, hinting at a possible missing dependency including this transition.

References

[1] E. Badouel, L. Bernardinello, and P. Darondeau. Polynomial algorithms for the synthesis of bounded nets. In *Lecture Notes in Computer Science, 915*, pp 647–679, 1995.

[2] M.P. Cabasino, P. Darondeau, M.P. Fanti, and C. Seatzu. Model identification and synthesis of discrete-event systems. In *Contemporary Issues in System Science and Engineering*. IEEE-Wiley, pp 1–22, 2014.

[3] M.P. Cabasino, A. Giua, C.N. Hadjicostis, and C. Seatzu. Fault model identification and synthesis in petri nets. In *Discrete Event Dynamic Systems*, pp 1–22, 2014.

[4] M.P. Cabasino, A. Giua, and C. Seatzu. Identification of petri nets from knowledge of their language. In *Discrete Event Dynamic Systems, 17(4)*, pp 447–474, 2007.

[5] M. Dotoli, M.P. Fanti, A.M. Mangini, and W.Ukovitch. Identification of the unobservable behaviour of industrial automation systems by petri nets. In *Control Engineering Practice, 9(9)*, pp 958–966, 2010.

[6] A.-P. Estrada-Vargas, J.-J. Lesage, and E. Lopez-Mellado. Identification of industrial automation systems: Building compact and expressive petri net models from observable behavior. In *2012 Amercian Control Conference*, Montreal, QC, pp 6095–6101, 2012.

[7] A.-P. Estrada-Vargas, J.-J. Lesage, and E. Lopez-Mellado. Identification of partially observable discrete event manufacturing systems. In *2013 IEEE 18th Conference on Emerging Technologies and Factory Automation (ETFA)*, Cagliari, pp 1–7, 2013.

[8] A. Giua and C. Seatzu. Identification of free-labeled petri nets via integer programming. In *Proc of the 44th IEEE Conference on Decision and Control, and the European Control Conference*, Seville, pp 7639–7644, 2005.

[9] Antoni Mazurkiewicz. Introduction to trace theory. In *The Book of Traces*, World Scientific Publishing Company, pp 3–41, 1995.

[10] T. Tapia-Flores, E. Lopez-Mellado, A.-P. Estrada-Vargas, and J.-J. Lesage. Petri net discovery of discrete event processes by computing t-invariants. In *2014 IEEE 19th Conference on Emerging Technologies and Factory Automation (ETFA)*, Barcelona, pp 1–8, 2014.

[11] W.-M.-P Van der Aalst. Do petri nets provide the right representational bias for process mining? In *Proc. of the Workshop Applications of Region Theory 2011*, pp 85–94, 2011.

[12] W.-M.-P Van der Aalst, A.-J.-M.-M. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. In *IEEE Transactions on Knowledge and Data Engineering 16(9)* pp. 1128–1142, 2004.

[13] B.F. Van Dongen, A.K. Alves de Medeiros, and L. Wen. Process mining: Overview and outlook of petri net discovery algorithms. In *Transactions on Petri Nets and Other Models of Concurrency - ToPNoC II, LNCS 5460*, pp. 225–242, 2009.

[14] L. Wen, W.-M.-P. Van der Aalst, J. Wang, and J. Sun. Mining process models with non-free-choice constructs. In *Data Mining and Knowledge Discovery 15(2)*, pp. 145–180, 2007.