Méthode active pour l'identification de réseaux de Petri saufs

Manon Le Moigne[1,2], Rémi Parrot[2], and Olivier H. Roux[2]

[1] ENS Paris-Saclay, Université Paris-Saclay, Gif-sur-Yvette, France
manon.le_moigne@ens-paris-saclay.fr
[2] Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004,
F-44000 Nantes, France
{surname}.{name}@ec-nantes.fr

Abstract

L'identification de systèmes à événements discrets a pour but de déterminer un modèle du comportement d'un système à partir d'observations de signaux de ce système (boite noire). Il existe deux familles d'identification : les méthodes passives, qui reposent sur des observations du système dans son environnement et les méthodes actives, qui interagissent avec le système pour provoquer des comportements d'intérêt.

Nous nous intéressons à des systèmes concurrents pour lesquels la modélisation par des réseaux de Petri est particulièrement adaptée. Si l'identification de réseaux de Petri a été explorée par des méthodes passives, elle n'a, à notre connaissance pas été abordée par des méthodes actives.

Dans cet article nous présentons une méthode active d'identification de réseaux de Petri saufs. Notre approche se déroule en deux temps : l'apprentissage du langage régulier avec par exemple l'algorithme L*, puis l'inférence des réseaux de Petri saufs qui pourraient engendrer ce langage, basée sur un solveur SAT. Parmi les solutions trouvées, nous proposons plusieurs critères pour les discriminer selon l'usage, tels que l'interprétabilité du réseau de Petri ou des connaissances structurelles du système (nombre de places, invariants, ...).

1 Introduction

Les réseaux de Petri permettent de représenter des systèmes à événements discrets. Ils sont particulièrement adaptés pour la modélisation d'actions concurrentes qui évoluent en parallèle [8]. Les réseaux de Petri saufs sont des réseaux dont le nombre de jeton par place est limité à un.

Dans l'étude des systèmes à événements discrets, l'identification consiste à déterminer un modèle représentant le fonctionnement d'un système le plus exactement possible. Pour cela, le système se présente sous la forme d'une boite noire dont on peut observer les signaux (entrées et sorties). Un cas particulier est le *process mining* qui a pour but d'identifier les modèles et les relations nécessaires à une prise de décision et à une planification.

Il existe deux familles d'identification.

i) Premièrement, les méthodes passives utilisent des observations du système qui évolue librement dans son environnement. Il n'y a pas d'interaction entre le système et l'observateur qui le modélise. Plusieurs méthodes ont été proposées dans la littérature, en particulier dans le domaine des réseaux de Petri : [21] utilise les T-invariants pour identifier un modèle, [12] propose une méthode d'identification se basant un modèle de connaissance fait par un expert et des observations, [4, 23] sont des revues de la littérature sur le *process mining* de réseaux de Petri.

Un cas particulier des méthodes passives consiste à observer l'ensemble des mots d'un langage fini. Ainsi, dans [5], une méthode d'identification de réseau de Petri à partir de langage

fini est proposée, grâce à la résolution d'un problème de programmation entier (ILP). Le langage est supposé totalement connu pour les mots de longueurs inférieures à une valeur donnée. Une extension est proposée pour les langages réguliers, incluant des langages avec des mots infinis venant d'un répétition de cycle. Cette méthode permet donc de passer d'un automate à un réseau de Petri dont le nombre de place est fixé. Enfin, l'article propose des méthodes pour affiner les solutions en fonction d'informations supplémentaires comme les invariants ou le marquage initial.

ii) Deuxièmement, les méthodes actives interagissent avec le système pour provoquer et observer des comportements intéressants. Cela va, par exemple, consister à choisir une partie des entrées de la boite noire pour observer ses sorties. Dana Angluin pose les fondations de l'apprentissage actif de systèmes à événements discrets avec l'algorithme L* [1]. Cette approche a donné lieu à plusieurs extensions : [14] propose l'algorithme TTT qui a une organisation des observations sans redondance, [11] utilise des arbres de discrimination adaptatifs, L# étudie dans [22] la séparation plutôt que l'équivalence entre les observations. D'autres travaux s'intéressent à l'application de ces approches à des modèles plus expressifs, par exemple [24] propose une extension de L* pour des automates temporisés.

Notre contribution. À notre connaissance, il n'existe pas de méthode active d'apprentissage de réseau de Petri. Cet article a pour objectif d'en proposer une, en combinant et transposant plusieurs briques présentes dans la littérature. Notre méthode est basée sur les algorithmes d'identification actifs, type L*, pour déterminer le langage du système. Puis nous proposons une modélisation SAT permettant de déterminer les réseaux de Petri qui engendre le langage identifié. La détermination du réseau de Petri à partir du langage est réalisée avec une méthode présentant des différences par rapport à celle proposée dans [5], bien que l'objectif soit le même. Dans l'article cité, la détermination passe par la résolution d'un problème de programation entier, alors que notre contribution utilise un solveur SAT pour résoudre un problème logique.

Notre méthode, pour identifier des réseaux de Petri saufs, est ainsi composée de trois étapes :

- utilisation d'un algorithme type L* pour déterminer le langage accepté par le système,
- utilisation d'un solveur SAT pour trouver tous les réseaux de Petri saufs, avec un nombre minimal de place ou avec une contrainte du nombre de places, jouant ce langage,
- discrimination des solutions pour garder les plus pertinentes.

Pour discriminer les solutions, plusieurs critères peuvent être utilisés. On peut souhaiter des propriétés sur les P-invariant, par une connaissance supplémentaire partielle du système, ou encore pour faire apparaitre des processus concurrents. Pour cela, nous utilisons l'algorithme de Farkas [10, 16].

Nous proposons aussi de déterminer les graphes les plus lisibles lorsque le but est de proposer une représentation graphique du réseau de Petri interprétable par l'humain. Il existe une corrélation entre le nombre de croisements d'arcs dans un graphe et sa lisibilité [18, 9].

Plan de l'article. La section 2 donne les définitions principales : les automates déterministes, les réseaux de Petri saufs, les langages. La section 3 présente les problèmes d'apprentissage actif et décrit la méthode utilisée pour déterminer, à partir d'un langage régulier, les réseaux de Petri correspondant. Nous présenterons des critères de discrimination des solutions dans la section 4. Nous proposons une implémentation et un exemple d'application dans la section 5. Enfin, nous conclurons dans la section 6.

2 Definitions

2.1 Automate déterministe

Définition 1 (Automate Déterministe)**.** *Un automate déterministe est un n-uplet* $\mathcal{A} = (\Sigma, Q, q_0, \delta, G)$*, où :*

- Σ *est un ensemble fini de lettres, appelé* alphabet*,*
- Q *un ensemble d'états,*
- $q_0 \in Q$ *est l'état initial,*
- $\delta \colon Q \times \Sigma \mapsto Q$ *est la fonction de transition,*
- $G \subseteq Q$ *un ensemble d'états accepteurs.*

Si Q *est fini,* $\mathcal{A}$ *est un automate* fini*.*

L'état d'un automate évolue en partant de son état initial q_0 est en suivant les transitions possibles de δ. L'automate passe de l'état q à q' avec la lettre a, si $\delta(q, a) = q'$. On note $q \xrightarrow{a} q'$.

Exécution. Une exécution de l'automate est une séquence $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots q_{n-1} \xrightarrow{a_n} q_n$, telle que $\forall i \in [1..n], \delta(q_{i-1}, a_i) = q_i$.

2.2 Réseau de Petri

Définition 2 (Réseau de Petri)**.** Un réseau de Petri ordinaire est un n-uplet $\mathcal{N} = (P, T, F, M_0)$, où P est un ensemble fini de places, T est un ensemble fini de transitions, $F \subseteq (P \times T) \cup (T \times P)$ est la relation de flots entre les places et les transitions. Un marquage M est une fonction $P \to \mathbb{N}$ qui associe à chaque place un entier non négatif (représenté par des jetons). $M(p)$ est le marquage de la place $p \in P$ (et donc le nombre de jetons dans la place p) et M_0 est le marquage initial.

Dans la suite de l'article, nous supposons que les places $p_1, p_2, \dots, p_n$ du réseau sont ordonnées. Un marquage peut alors être vu comme le vecteur $\begin{bmatrix} M(p_1) \\ \vdots \\ M(p_n) \end{bmatrix}$.

Matrice d'incidence. Un réseau de Petri est classiquement représenté par sa matrice d'incidence. Cette matrice est analogue à la matrice d'adjacente d'un graphe orienté.

Pour un réseau de Petri ordinaire $\mathcal{N} = (P, T, F, M_0)$, soit $C \in \{-1, 0, 1\}^{|P| \times |T|}$ sa matrice d'incidence, $pre \in \{0, 1\}^{|P| \times |T|}$ et $post \in \{0, 1\}^{|P| \times |T|}$ ses matrices de pré-condition et de post-condition définies $\forall p \in P$, $\forall t \in T$ par :

$$C(p,t) = \begin{cases} -1 \text{ si } \langle p, t\rangle \in F \text{ et } \langle t, p\rangle \notin F \\ 1 \text{ si } \langle t, p\rangle \in F \text{ et } \langle p, t\rangle \notin F \\ 0 \text{ sinon.} \end{cases}$$

$$pre(p,t) = \begin{cases} 1 \text{ si } \langle p, t\rangle \in F \\ 0 \text{ sinon.} \end{cases} \quad \text{et } post(p,t) = \begin{cases} 1 \text{ si } \langle t, p\rangle \in F \\ 0 \text{ sinon.} \end{cases}$$

On a donc $C = post - pre$

Comme raccourcis, pour une transition $t \in T$, nous notons $pre(t)$ et $post(t)$ les vecteurs correspondant à la colonne t des matrices pre et $post$.

Enfin, pour une transition $t \in T$, ${}^\bullet t = \{p, \mid \langle p, t\rangle \in F\}$ (resp. $t^\bullet = \{p, \mid \langle t, p\rangle \in F\}$) est l'ensemble des places d'entrée (resp. de sortie) désignées par F.

Règle de tir. Le marquage d'un réseau évolue selon la règle de tir des transitions :

- Une transition t est *sensibilisée* par M si $M \geq pre(t)$ c.a.d. si il existe au moins un jeton dans chacune de ses places d'entrée. $En(M) = \{t, \mid M \geq pre(t)\}$ est l'ensemble des transitions sensibilisées par M.
- Une transition $t \in En(M)$ peut être tirée. Le tir de t transforme M en M' (noté $M \overset{t}{\mapsto} M'$) en retirant un jeton de chacune de ses places d'entrée $p \in {}^\bullet t$ et en ajoutant un jeton dans chacune de ses places de sortie $p \in t^\bullet$. Nous avons donc $M' = M - pre(t) + post(t)$.

Réseau sauf. Un réseau de Petri est *sauf* (resp. *k-borné*) si pour tout marquage accessible, chaque place contient au maximum un jeton (resp. k jetons).

Séquence de tirs. Il y a une *séquence de tirs* $t_1 t_2 \dots t_n \in T^*$ à partir du marquage M_0 si il y a une séquence de marquages $M_0 M_1 M_2 \dots M_n$ telle qu'une transition t_i est sensibilisée par M_{i-1} et son tir transforme M_{i-1} en M_i.

P-invariant. Soit un réseau de Petri $\mathcal{N} = (P, T, F, M_0)$ et sa matrice d'incidence $C \in \{-1, 0, 1\}^{|P|\times|T|}$. Un p-invariant $X \in \mathbb{N}^{|P|}$, est une solution entière de $X^T.C = 0$. Un p-invariant X est dit minimal s'il n'existe pas un autre p-invariant X' tel que $X' < X$. Plusieurs algorithmes de calcul des p-invariants minimaux sont proposés dans [6]. Nous considérerons seulement les p-invariants minimaux.

Exemple: Considérons le réseau de Petri $\mathcal{N}_0$ de la figure 1.

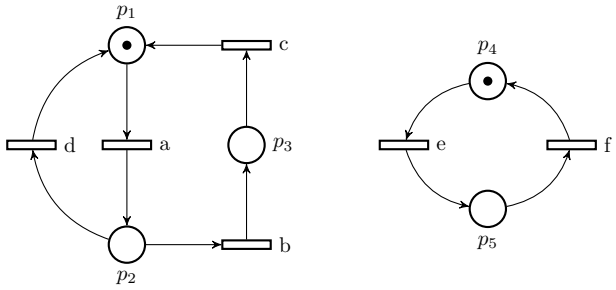


Figure 1: Petri net $\mathcal{N}_0$.

Les matrices d'incidence sont :

$$pre = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}, \quad post = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}, \quad C = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{bmatrix} -1 & 0 & 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \end{array}$$

Les P-invariants minimaux sont $\mathtt{P}_{inv} = \left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \right\}$.

Graphe de marquages. Soit un réseau de Petri $\mathcal{N} = (P, T, F, M_0)$. On appelle *graphe de marquages* du réseau, l'automate déterministe $\mathcal{A}^{\mathcal{N}} = (\Sigma^{\mathcal{N}}, Q^{\mathcal{N}}, q_0^{\mathcal{N}}, \delta^{\mathcal{N}}, G^{\mathcal{N}})$, tel que :

- $\Sigma^{\mathcal{N}} = T$
- $Q^{\mathcal{N}}$ est l'ensemble des marquages accessibles
- $q_0^{\mathcal{N}} = M_0$
- $\delta^{\mathcal{N}}(M_i, t) = M'$ ssi $t \in En(M)$ et $M' = M - pre(t) + post(t)$
- $G^{\mathcal{N}} = Q^{\mathcal{N}}$

Si $\mathcal{N}$ est borné alors son graphe de marquages est fini.

Le graphe des marquages du réseau de Petri $\mathcal{N}_0$ de la figure 1 est donné Figure 2.

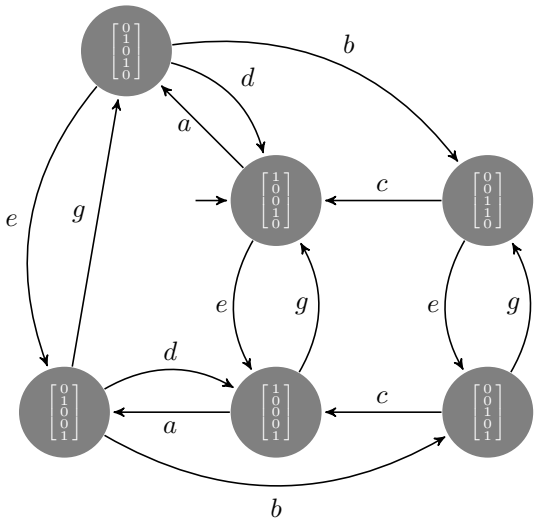


Figure 2: Graphe des marquages de $\mathcal{N}_0$.

2.3 Langages

Soit un automate déterministe $\mathcal{A} = (\Sigma, Q, q_0, \delta, G)$ et soit $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots q_{n-1} \xrightarrow{a_n} q_n \dots$ une exécution de l'automate. On appelle *mot* la séquence de lettre $a_1 a_2 \dots a_n \in \Sigma^*$ correspondant à cette exécution. Un mot est *accepté* par l'automate si l'exécution qui correspond se termine dans un état accepteur $q_n \in G$.

Le langage d'un automate déterministe est l'ensemble des mots qu'il accepte. On note $L(\mathcal{A})$ le langage de $\mathcal{A}$.

Définition 3 (Langage régulier)**.** *Un langage L est un* langage régulier *ssi il existe un automate fini déterministe $\mathcal{A}$ tel que $L = L(\mathcal{A})$.*

Soit un réseau de Petri $\mathcal{N}$ et $\mathcal{A}^{\mathcal{N}}$ son graphe de marquages. Le langage du réseau, noté $L(\mathcal{N})$ est le langage de son graphe de marquages : $L(\mathcal{N}) = L(\mathcal{A}^{\mathcal{N}})$.

Un langage L est dit *préfixe-clos* si pour tout mot $w \in L$, tous les préfixes de w sont également dans L. Le langage d'un réseau de Petri est préfixe-clos. De plus, si le réseau est borné alors son langage est régulier.

3 Apprentissage actif

L'apprentissage ou identification active consiste à déterminer un modèle de fonctionnement d'un système en exerçant ses entrées et en observant ses sorties. Les travaux de Dana Angluin [1] sont fondateurs dans le domaine de l'apprentissage actif de systèmes à évènements discrets.

Dans cette approche, le système est modélisé par un langage régulier caché, qui modélise les entrées/sorties du système, et l'objectif est de découvrir ce langage. Angluin a formalisé ce problème et a proposé le fameux algorithme L* pour le résoudre.

3.1 Problème

Cette partie reprend les travaux de Dana Angluin sur L* et adapte la formulation du problème aux réseaux de Petri.

Pour identifier le langage L d'un système donné, nous avons recours à deux types de requêtes :

- la requête d'*appartenance* $\mathsf{MEMBER}(w)$: demande si un mot $w \in \Sigma^*$ est dans le langage $w \stackrel{?}{\in} L$, et attend en réponse *oui* ou *non*,
- la requête d'*équivalence* $\mathsf{EQUIV}(H)$: demande si l'automate hypothèse H accepte le même langage $L(H) \stackrel{?}{=} L$, et attend en réponse *oui* ou un *contre-exemple* (un mot qui est dans $L(H)$ et n'est pas dans L ou inversement).

Un système permettant de répondre à ces deux requêtes est appelé un *enseignant minimal adéquat*.

Problème 1 (Identification d'automate fini déterministe)**.** *Le problème d'identification d'automate fini déterministe consiste à construire un automate fini déterministe acceptant un langage régulier caché L uniquement à partir de requêtes à un enseignant minimal adéquat.*

L* est un des algorithmes permettant de résoudre ce problème.

Cependant, l'existence d'un enseignant minimal adéquat est hautement improbable dans un cas réel. En effet, la requête d'équivalence demande d'avoir une connaissance presque totale du langage caché. Une approximation probabiliste a donc été proposée dans les travaux d'Angluin.

Supposons une distribution de probabilité $\mathcal{P}$ sur l'ensemble des mots de Σ^*. L'objectif est d'identifier une hypothèse qui a une probabilité de se tromper suffisamment faible.

Définition 4 (Automate ε-approximant)**.** *Soit L un langage régulier, ε une précision souhaitée, H un automate fini déterministe. H est un automate ε-approximant L ssi $\sum_{w \in L \oplus L(H)} \mathcal{P}(w) \leq \varepsilon$ avec $L \oplus L(H) = (L \setminus L(H)) \cup (L(H) \setminus L)$*

L'enseignant minimal adéquat peut alors être remplacé par un *oracle* répondant également à deux types de requêtes :

- la requête d'*appartenance* $\mathsf{MEMBER}(w)$,

- la requête d'*échantillonnage aléatoire* EX() : tire aléatoirement un mot $w \in \Sigma^*$ en suivant la loi $\mathcal{P}$, demande s'il est dans le langage $w \overset{?}{\in} L$, et attend en réponse *oui* ou *non*.

Problème 2 (Identification d'automate ε-approximant)**.** *Le problème d'identification d'automate ε-approximant consiste à obtenir un automate ε-approximant un langage régulier caché L uniquement à partir de requêtes à un oracle.*

Une version probabiliste de L* (sensiblement identique) permet de résoudre ce problème.

Cette formulation est celle utilisée dans la pratique car elle permet d'identifier des systèmes réels. Une méthode classique consiste à construire l'oracle *autour* du système à identifier. C'est-à-dire que lorsqu'une requête est faîte à l'oracle, il fourni une entrée au système et observe les sorties. Puis, il détermine la réponse à la requête à partir des observations.

Si on considère un système qui fourni exactement une sortie pour chaque entrée reçue. Une approche simple est de s'intéresser à l'alphabet consistué de tous les couples possibles d'entrée/sortie du système. Un mot *accepté* serait alors une séquence possible d'entrée/sortie.

Définition 5 (Réseau de Petri ε-approximant)**.** *Soit L un langage régulier préfixe-clos, ε une précision souhaitée, $\mathcal{N}$ un réseau de Petri borné. $\mathcal{N}$ est un réseau de Petri ε-approximant L ssi graphe des marquages est un automate ε-approximant L.*

Nous formalisons le problème d'identification de réseau de Petri de la manière suivante :

Problème 3 (Identification de réseau de Petri ε-approximant)**.** *Le problème d'identification de réseau de Petri ε-approximant consiste à obtenir un réseau ε-approximant un langage régulier préfixe-clos caché L uniquement à partir de requêtes à un oracle.*

Nous nous intéressons ici à ce problème, qui est, à notre connaissance, encore ouvert. Dans ce travail, nous nous restreignons aux réseaux de Petri saufs.

3.2 Du langage aux réseaux de Petri : solveur SAT

Nous cherchons maintenant à synthétiser les réseaux de Petri répondant au problème 3. Pour ce faire, nous résolvons le problème 2 avec un algorithme *type L**, ce qui nous donne un langage ε-approximant le langage caché du système. Nous synthétisons ensuite les réseaux de Petri qui acceptent ce langage. C'est-à-dire que nous cherchons tous les réseaux de Petri qui jouent ce langage : tout mot du langage est une séquence tirable et toute séquence tirable est un mot du langage.

Pour cela, nous utilisons un solveur SAT qui doit trouver les matrices d'incidence et les marquages initiaux qui correspondent à partir du graphe de marquage du réseau.

Dans un premier temps, nous construisons un graphe de marquage partiel à partir du langage obtenue avec L*. On présente la grammaire sous la forme d'un graphe : pour chaque état (le marquage exact est inconnu, on sait seulement qu'il est distinct de celui des autres états), on a l'état auquel mène chacune des transitions dont le tir est possible. On sait aussi quel état est l'état initial (mais sans son marquage).

Ensuite, nous nous ramenons à un problème de satisfaisabilité booléenne (SAT). On a trois matrices de 0 ou 1 : la matrice pre, la matrice $post$ et le marquage initial. On va aussi utiliser des matrices intermédiaires M_i qui représentent les marquages dans chaque état. Les valeurs de ces matrices seront interprétées comme des booléens. Ainsi, on va créer des relations entre les matrices sous la forme d'assertions booléennes à partir du graphe des marquages.

La table 1, nous donne toutes les évolutions possibles de marquage par le tir d'une transition t. C'est-à-dire que le marquage de la place p, $M_i(p)$ devient $M_{i+1}(p)$ par le tir de t. Lorsque

le tir est impossible la valeur de $M_{i+1}(p)$ est notée X dans la table. Il y a deux évolutions impossibles lorsque la transition n'est pas sensibilisée par le marquage M_i (les lignes 3 et 4). Il y a une évolution impossible, car le réseau est sauf, donc il est impossible d'ajouter un jeton dans une place déjà marquée (la ligne 6). Les cinq autres évolutions sont possibles et suivent la règle de tir $M_{i+1}(p) = M_i(p) - pre(p,t) + post(p,t)$.

Supposons qu'il existe dans le graphe de marquage une transition par le tir de t entre deux marquages inconnus M_i et M_{i+1}. On sait que pour chaque place p du réseau, au moins une ligne, parmi les lignes 1, 2, 5, 7 et 8, est vérifiée. Chaque ligne peut s'écrire sous la forme d'une assertion booléenne. Par exemple, la ligne 2 s'écrit : $\overline{M_i(p)} \wedge \overline{pre(p,t)} \wedge post(p,t) \wedge M_{i+1}(p)$ Ainsi, il suffit d'écrire une disjonction entre chacune de ces lignes, puis de faire une conjonction pour chaque place du réseau :

$$\begin{aligned} \forall p,\ & \left(\overline{M_i(p)} \wedge \overline{pre(p,t)} \wedge \overline{post(p,t)} \wedge \overline{M_{i+1}(p)}\right) \\ & \vee \left(\overline{M_i(p)} \wedge \overline{pre(p,t)} \wedge post(p,t) \wedge M_{i+1}(p)\right) \\ & \vee \left(M_i(p) \wedge \overline{pre(p,t)} \wedge \overline{post(p,t)} \wedge \overline{M_{i+1}(p)}\right) \\ & \vee \left(M_i(p) \wedge pre(p,t) \wedge \overline{post(p,t)} \wedge \overline{M_{i+1}(p)}\right) \\ & \vee \left(M_i(p) \wedge pre(p,t) \wedge post(p,t) \wedge M_{i+1}(p)\right) \end{aligned} \tag{1}$$

Supposons qu'une transition n'existe pas entre deux marquages inconnu M_i et M_{i+1}. Alors, on sait qu'il s'agit du tir d'une transition t impossible. Il existe donc une place p pour laquelle, l'une des lignes 3, 4 ou 6 est vérifiée :

$$\exists p \mid \left(\overline{M_i(p)} \wedge pre(p,t)\right) \vee \left(M_i(p) \wedge \overline{pre(p,t)} \wedge post(p,t)\right) \tag{2}$$

	$M_i(p)$	$pre(p,t)$	$post(p,t)$	$M_{i+1}(p)$
(1)	0	0	0	0
(2)	0	0	1	1
(3)	0	1	0	X (t non sensibilisée)
(4)	0	1	1	X (t non sensibilisée)
(5)	1	0	0	1
(6)	1	0	1	X (sauf : $M_{i+1}(p) \leq 1$)
(7)	1	1	0	0
(8)	1	1	1	1

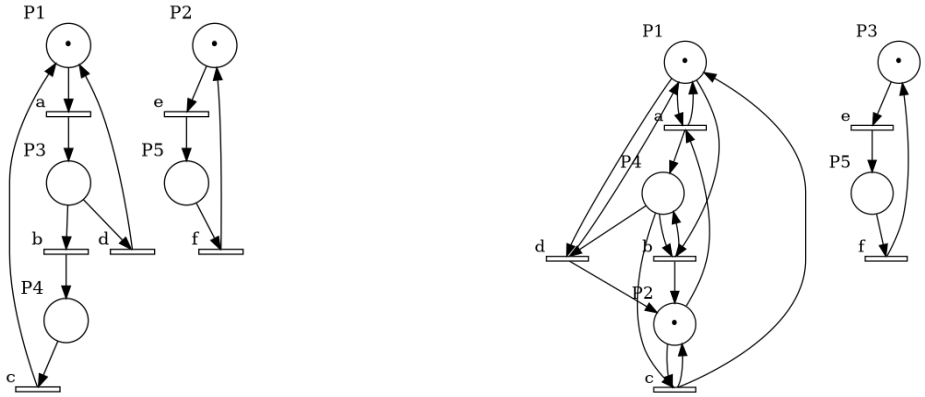
Table 1: Table des évolutions du marquage par le tir de t. Le symbole X représente un tir impossible.

Le problème 3 de synthèse des réseaux de Petri qui acceptent le langage accepté avec L* est donc représenté par la formule booléenne obtenue par la conjonction des équations 1 et 2 pour toutes les transitions possibles.

Nous utilisons un solveur SAT pour trouver toutes les configurations possibles, avec un nombre minimal de places. Dans notre cas, nous avons utilisé PySAT qui permet de prendre une formule booléenne, de la transformée en clauses, de résoudre le problème et de retourner les valeurs binaires des inconnues pour chaque solution. Une dernière étape est nécessaire pour mettre les résultats sous formes de matrices, plus facilement manipulables, et de retirer les

solutions identiques à une permutation du nom des places près. Les noms des places donnés lors de la construction des solutions est totalement indépendant des noms des places du système à identifier, car cette information n'est pas donnée par l'oracle.

Exemple : On reprend le réseau de Petri $\mathcal{N}_0$ précédent. Après avoir obtenu le langage et le graphe des marquages, on utilise le solveur SAT pour trouver les réseaux de Petri correspondants. Il y a 240 solutions, cependant il n'y en a que 2 distinctes une fois les permutations de place retirées. Les deux solutions sont tracées dans la figure 5. $\mathcal{N}_1$ est isomorphe au réseau initial $\mathcal{N}_0$, et possède un marquage initial de deux jetons, 12 arcs et les P-invariants [1, 0, 1, 1, 0] et [0, 1, 0, 0, 1]. $\mathcal{N}_2$ a un marquage initial de trois jetons, 20 arcs et les P-invariants [1, 1, 0, 1, 0] et [0, 0, 1, 0, 1].



(a) $\mathcal{N}_1$ (b) $\mathcal{N}_2$

Figure 3: Réseaux de Petri obtenu avec le solveur SAT

4 Discrimination des solutions

Par construction, tous les réseaux de Petri trouvés lors de l'étape précédente ont le même nombre de transitions que le nombre de lettres de l'alphabet : chaque lettre correspond à une unique transition.

De plus, notre approche consiste à chercher le nombre minimum de places : nous commençons avec une place et tant qu'aucune solution n'est trouvée par le solveur SAT, ce nombre est incrémenté d'une unité. Toutefois, si on a une information supplémentaire sur le réseau recherché, on peut fixer le nombre de places.

Notre algorithme retourne donc tous les réseaux de Petri avec un nombre de transitions fixé et un nombre de places minimal ou fixé, en ayant enlevé les solutions identiques à une permutation près des places. Ce nombre de solutions peut être très important, et on cherche donc des critères pour les discriminer.

Nous proposons quatre critères :

- le nombre de places (mentionné ci-dessus)
- la lisibilité du graphe

- les P-invariants des réseaux obtenus
- des connaissances partielles du système

Il est en effet possible de filtrer selon des critères qui seront pertinents en fonction des informations supplémentaires disponibles sur le système que l'on cherche à identifier. Par exemple, il serait possible de combiner un critère sur le nombre de places avec un autre sur le nombre de P-invariants.

On peut noter l'opération de discrimination :

$$\varphi' = \varphi \backslash \mathcal{F}$$

où φ est l'ensemble des solutions du solveur SAT, φ' l'ensemble des solutions après filtrage et $\mathcal{F}$ le filtre appliqué.

4.1 Lisibilité du graphe

Dans le cas où l'on cherche à identifier un système sur lequel on n'a pas d'autre information et que l'on souhaite obtenir une représentation graphique, on peut choisir de privilégier la lisibilité du graphe. En effet, si le principal but de l'identification est l'analyse du modèle obtenu, il est souhaitable d'en simplifier la lecture.

Dans la littérature, on trouve des études qui cherchent à mesurer la lisibilité d'un graphe, c'est-à-dire à quel point un·e lecteurice est capable de le comprendre facilement et rapidement. Il ressort de [18] que au moins deux critères améliorent la lisibilité : réduire le nombre de *courbures d'arc* et réduire le nombre de *croisement des arcs*.

Ces deux critères sont étroitement liés à la question de *planarité* : peut-on dessiner un graphe donné dans le plan sans croisement d'arcs. Cette question a été longuement étudiée et des algorithmes en temps linéaire permettent d'y répondre efficacement, par exemple [3]. De plus, le théorème de Fáry nous apprend qu'un graphe planaire peut être étiré pour être dessiné dans le plan avec uniquement des arcs droits qui ne se croisent pas.

Pour un graphe non-planaire, il devient bien plus compliqué de prendre en compte ces critères. Les problèmes de dessin de graphes non-planaires ont une littérature très fournie, comme le montre la revue de la littérature [7]. Cependant, ces problèmes sont pour la plupart très complexes, par exemple compter le nombre de croisements d'un graphe donné est un problème NP-complet [13].

Dans notre cas, nous comparons des réseaux de Petri qui ont le même nombre de places et le même nombre de transitions, c'est-à-dire des graphes qui ont le même nombre de nœuds. Ainsi, pour comparer leur lisibilité, on peut dans un premier temps tout simplement compter le nombre d'arcs.

Exemple : Dans la figure 5, il y a deux réseaux de Petri. Celui de gauche a 12 arcs, pas de croisement et est intuitivement plus lisible. Celui de droite a 20 arcs, 2 croisements (qui est le nombre minimal de croisements possibles d'après la conjecture sur le nombre minimal de croisements des graphes bipartis complets [19]) et est moins lisible. Avec le critère du nombre d'arcs, on garderait le réseau de gauche qui est plus simple à lire.

4.2 P-invariants

Lors de la modélisation classique d'un système sous la forme d'un réseau de Petri, læ concepteurice connaît sa structure (physique, logicielle...) et construit le modèle comme une composition de composants qui modélisent les sous-systèmes. Dans notre étude, nous devons identifier

ces composants sans connaissance a priori. Plusieurs études de décomposition de réseaux de Petri se basent sur les invariants de place [17, 20, 2] et montrent qu'ils permettent de mettre en évidence des composants du système.

Nous proposons donc comme critère de choisir parmi les solutions, celle ayant le nombre maximum de P-invariants.

Nous utilisons l'algorithme décrit par [6] pour déterminer les P-invariants minimaux.

Exemple : Les réseaux de la figure 5 ont chacun deux P-invariants, respectivement [1, 0, 1, 1, 0], [0, 1, 0, 0, 1] et [1, 1, 0, 1, 0], [0, 0, 1, 0, 1], donc on ne peut pas les différencier par le nombre de P-invariants.

4.3 Connaissances partielles

Un dernier critère consiste à prendre on compte une connaissance partielle du système. Par exemple le fait que le système devrait avoir un certain nombre de places ou encore comporte un composant dont le comportement doit être décrit par un P-invariant de n places. Cette approche utilise une combinaison des deux approches précédentes et peut se ramener à filtrer les solutions par rapport à un critère en augmentant le nombre de places visées jusqu'à obtenir une solution satisfaisant le critère. Nous illustrerons cette situation dans le cas d'étude de la section suivante.

5 Application

5.1 Implémentation

Nous avons implémenté, en Python, les différentes parties de la méthode proposée. Cette implémentation peut être intégrée avec l'outil Roméo [15].

D'une part, nous construisons un oracle pour L* avec un réseau de Petri (alphabet, matrices d'incidences, marquage initial). Le fichier contenant ces informations peut être généré par Roméo. Si le réseau d'un système à identifier est inconnu, il faut l'interfacer de manière à ce qu'il remplisse le rôle de l'oracle.

D'autre part, nous proposons une implémentation de la méthode pour trouver les réseaux de Petri (L* puis solveur SAT) et de les filtrer (nombre de places, lisibilité, P-invariants). D'autres critères de discrimination pourraient être implémentés selon les besoins. Cette implémentation est une preuve de concept avec une complexité importante qui nécessitera une amélioration de l'efficacité avant un usage plus important.

Les réseaux de Petri obtenus peuvent être enregistrés soit sous la forme d'un graphique (généré avec graphviz), soit sous la forme d'un fichier pour Roméo (format XML).

5.2 Exemple

On propose un exemple plus complet basé sur un système existant : les feux d'un croisement tram-voiture modélisé par le réseau de Petri de la figure 4.

La partie de gauche du réseau de Petri représente le comportement du tram et le reste représente la commande des feux de croisement. Par souci de simplicité, nous supposons que lorsque le feu est rouge pour les voitures, il est vert pour le tram et réciproquement.

Sur cet exemple, nous cherchons à illustrer l'identification de réseau de Petri sauf, et le filtrage par le nombre d'arcs, le nombre de P-invariants et une connaissance partielle du système.

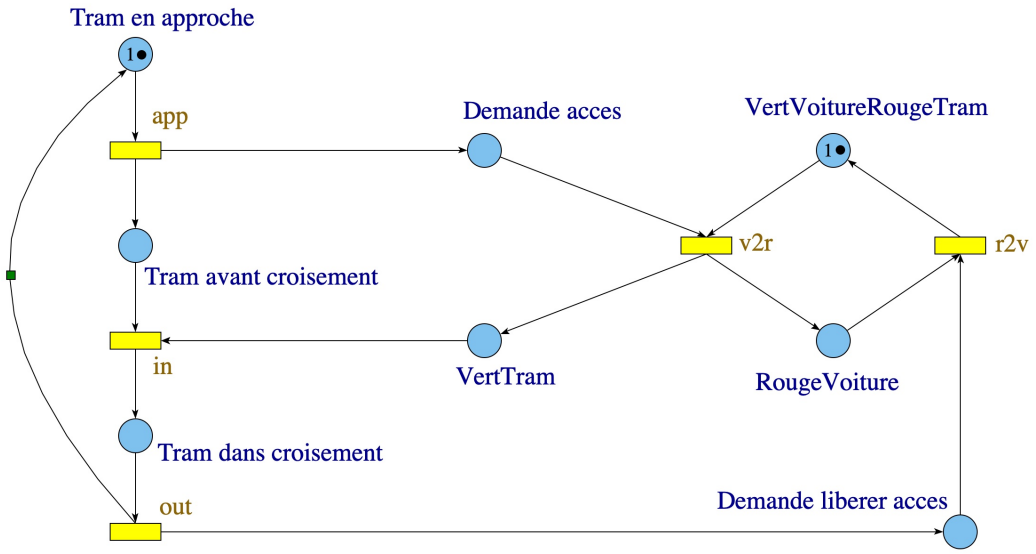


Figure 4: Réseau de Petri modélisant la commande des feux d'une intersection tram-voiture

Discrimination par nombre de places et nombre d'arcs Nous n'obtenons aucune solution avec strictement moins de 5 places.

Avec 5 places, il y a 5760 solutions, mais seulement 48 sont conservées, car les autres sont des permutations des places. Il y a 3 réseaux avec deux places initialement marquées et 45 avec trois places initialement marquées. Le nombre de réseaux obtenus en fonction des nombres d'arcs obtenus sont donnés dans la table 2. L'une des solutions avec le nombre minimal d'arcs est illustrée dans la figure 5a.

Nombre d'arcs	16	18	22	24	26
Nombre de solutions	2	2	12	20	12

Table 2: Table du nombre d'arcs des solutions trouvées pour des réseaux à 5 places

Discrimination par nombre de P-invariants Parmi les solutions à 5 places, il y a 4 solutions avec deux P-invariants, le nombre maximal. Chacune de ces 4 solutions présente 26 arcs, il n'est donc pas possible de les discriminer en plus par le nombre d'arc. Les 44 autres solutions ont un seul P-invariant. Par exemple le réseau de la figure 5b a deux P-invariants. Ces résultats sont récapitulés dans la table 3.

Nombre de P-invariant	1	2
Nombre de solutions	44	4

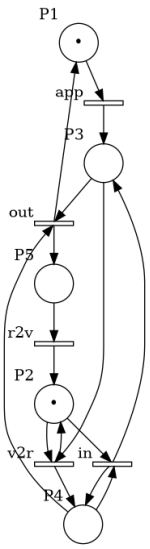
Table 3: Table du nombre de P-invariants des solutions trouvées pour des réseaux à 5 places

Discrimination par connaissance partielle Nous savons que le train peut prendre 3 états et les feux 2 états donc nous souhaitons une solution avec au moins deux P-Invariants respec-

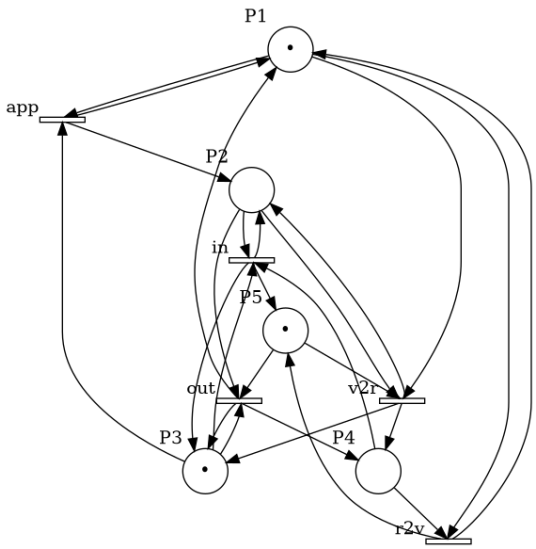
tivement de 3 et 2 places.

Les 4 solutions avec 2 P-invariants obtenues précédemment vérifient ce critère. Nous choisissons arbitrairement la solution de la figure 5b qui présente les P-invariants [1,1,1,0,0] et [0,0,0,1,1]. Ces P-invariants correspondent au tram (trois places) et au feu (deux places). Toutefois, on remarque que la modélisation du tram implique deux marques initiales sur les trois places, ce qui n'est pas exactement ce que l'on pourrait souhaiter pour représenter le comportement du tram. Par contre, pour le feu, on observe bien un cycle `P4`, `r2v`, `P5` et `v2r` avec une marque.

Avec cet exemple, le critère par connaissance partielle des P-invariants est équivalent au critère du maximum de P-invariants, mais ce n'est pas toujours le cas.



(a) Réseau de Petri avec le minimum d'arc



(b) Réseau de Petri avec les P-invariants [1,1,1,0,0] et [0,0,0,1,1]

Figure 5: Réseaux de Petri identifiés sur le système du tram-voiture

6 Conclusion

Dans cet article, nous avons présenté une méthode d'identification active de réseau de Petri.

Pour cela, nous utilisons un algorithme type L* pour déterminer une approximation du langage accepté par le système à identifier. Ensuite, nous synthétisons tous les réseaux de Petri pouvant jouer ce langage grâce à un solveur SAT. Enfin nous proposons des méthodes de discrimination des solutions trouvées pour obtenir la ou les plus adaptées. Une partie de ces méthodes est intégrée dans la formulation SAT pour limiter le nombre de solutions proposées par le solveur, le reste est appliqué a posteriori pour filtrer les solutions.

Nous avons implémenté cette méthode, qui peut être interfacée avec l'outil Roméo, et nous l'avons testé sur un cas d'étude.

Nous souhaitons poursuivre ce travail dans plusieurs directions. Nous voudrions étendre cette approche à d'autres classes de réseaux de Petri. Tout d'abord, il serait intéressant

d'intégrer les arcs particuliers tels que les arcs de lecture, de reset ou encore les arcs inhibiteurs. Ensuite, nous souhaitons relâcher les contraintes de bornitude en étudiant les réseaux de Petri non saufs puis non bornés (ce qui nécessite une extension aux langages non réguliers). Enfin, nous envisageons d'étudier les extensions de réseau de Petri avec du temps.

De plus nous souhaitons améliorer l'efficacité de la méthode en intégrant le plus possible les critères de discrimination dans la modélisation du problème SAT afin de réduire le nombre de solutions qu'il propose. Nous souhaitons également étudier d'autres critères de discrimination pouvant prendre en compte les spécificités de ces nouvelles classes de réseaux en particulier concernant les aspects temporels.

References

[1] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, November 1987.

[2] Sandie Balaguer, Thomas Chatain, and Stefan Haar. A concurrency-preserving translation from time Petri nets to networks of timed automata. *Formal Methods in System Design*, 40(3):330–355, June 2012.

[3] John M. Boyer and Wendy J. Myrvold. On the cutting edge: simplified planarity by edge addition. *Journal of Graph Algorithms and Applications*, 8(3):241–273, 2004.

[4] Nadia Busi and G. Michele Pinna. Process discovery and Petri nets. *Mathematical Structures in Computer Science*, 19(6):1091–1124, December 2009.

[5] Maria Paola Cabasino, Alessandro Giua, and Carla Seatzu. Identification of Petri Nets from Knowledge of Their Language. *Discrete Event Dynamic Systems*, 17(4):447–474, November 2007.

[6] José Manuel Colom and Manuel Silva Suárez. Convex geometry and semiflows in P/T nets. A comparative study of algorithms for computation of minimal p-semiflows. In *10th International Conference on Applications and Theory of Petri Nets, Bonn, Germany, June 1989, Proceedings*, volume 483 of *Lecture Notes in Computer Science*, pages 79–112. Springer, 1989.

[7] Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. A survey on graph drawing beyond planarity. *ACM Computing Surveys (CSUR)*, 52:1 – 37, 2018.

[8] Ana Paula Estrada-Vargas, Ernesto López-Mellado, and Jean-Jacques Lesage. A Black-Box Identification Method for Automated Discrete-Event Systems. *IEEE Transactions on Automation Science and Engineering*, 14(3):1321–1336, July 2017. Conference Name: IEEE Transactions on Automation Science and Engineering.

[9] Guy Even, Sudipto Guha, and Baruch Schieber. Improved Approximations of Crossings in Graph Drawings and VLSI Layout Areas. *SIAM Journal on Computing*, 32(1):231–252, January 2002.

[10] Julius Farkas. Theorie der einfachen Ungleichungen. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1902(124):1–27, January 1902.

[11] Markus Theo Frohme. Active Automata Learning with Adaptive Distinguishing Sequences, February 2019. arXiv:1902.01139 [cs].

[12] Clement Galetta, Jean-Marc Roussel, and Jean-Marc Faure. A relative identification method for reactive systems. *IFAC-PapersOnLine*, 51(7):152–159, 2018. Publisher: Elsevier BV.

[13] M. R. Garey and David S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 4:312–316, 1983.

[14] Malte Isberner, Falk Howar, and Bernhard Steffen. The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *Runtime Verification*, volume 8734, pages 307–322. Springer International Publishing, Cham, 2014. Series Title: Lecture Notes in Computer Science.

[15] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez. Romeo: A parametric model-checker for Petri nets with stopwatches. In Stefan Kowalewski and Anna Philippou,

editors, *15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009)*, volume 5505 of *Lecture Notes in Computer Science*, pages 54–57, York, United Kingdom, March 2009. Springer.

[16] D.C. Marinescu, M. Beaven, and R. Stansifer. A parallel algorithm for computing invariants of Petri net models. In *Proceedings of the Fourth International Workshop on Petri Nets and Performance Models PNPM91*, pages 136–143, Melbourne, Vic., Australia, 1991. IEEE Comput. Soc.

[17] Enric Pastor, Jordi Cortadella, and Marco A. Peña. Structural methods to improve the symbolic analysis of petri nets. In Susanna Donatelli and Jetty Kleijn, editors, *Application and Theory of Petri Nets 1999*, pages 26–45, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[18] Helen C. Purchase, Robert F. Cohen, and Murray James. Validating graph drawing aesthetics. In G. Goos, J. Hartmanis, J. Van Leeuwen, and Franz J. Brandenburg, editors, *Graph Drawing*, volume 1027, pages 435–446. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996. Series Title: Lecture Notes in Computer Science.

[19] Guy Richard K. The decline and fall of zarankiewicz's theorem. In Hubert Garavel and John Hatcliff, editors, *Proof Techniques in Graph Theory (Proc. Second Ann Arbor Graph Theory Conf., Ann Arbor, Mich., 1968)*, pages 63–69. Academic Press, New York-London, 1969.

[20] Karsten Schmidt. Using petri net invariants in state space construction. In Hubert Garavel and John Hatcliff, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 473–488, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[21] Tonatiuh Tapia-Flores, Ernesto Lopez-Mellado, Ana Paula Estrada-Vargas, and Jean-Jacques Lesage. Discovering Petri Net Models of Discrete-Event Processes by Computing T-Invariants. *IEEE Transactions on Automation Science and Engineering*, 15(3):992–1003, July 2018.

[22] Frits Vaandrager, Bharat Garhewal, Jurriaan Rot, and Thorsten Wißmann. A New Approach for Active Automata Learning Based on Apartness. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 13243, pages 223–243. Springer International Publishing, Cham, 2022. Series Title: Lecture Notes in Computer Science.

[23] B. F. Van Dongen, A. K. Alves De Medeiros, and L. Wen. Process Mining: Overview and Outlook of Petri Net Discovery Algorithms. In Kurt Jensen and Wil M. P. Van Der Aalst, editors, *Transactions on Petri Nets and Other Models of Concurrency II*, volume 5460, pages 225–242. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. Series Title: Lecture Notes in Computer Science.

[24] Masaki Waga. Active Learning of Deterministic Timed Automata with Myhill-Nerode Style Characterization, May 2023. arXiv:2305.17742 [cs].