Graphe de Pas Persistant

Pierre-Olivier Ribet, François Vernadat, Bernard Berthomieu
e-mail : {ribet,vernadat,berthomieu}@laas.fr

LAAS-CNRS
7 avenue du Colonel Roche F-31077 Toulouse cedex - France
Fax: +33 61.33.64.11, Tel: +33. 61.33.62.65

Abstract. Une approche commune à la vérification des systèmes concurrents consiste à construire le graphe des états accessibles représentant de façon exhaustive le comportement du système et à l'analyser. Malheureusement, ce graphe d'états est souvent trop grand pour être effectivement construit: sa taille évolue le plus souvent de façon exponentielle en le nombre de composants du système. Pour limiter cette explosion, des techniques d'exploration partielle ont été développées permettant d'obtenir des graphes d'états réduits qui offrent certaines capacités d'analyse. Les techniques d'ordre partiel ont été développées pour limiter l'explosion combinatoire en s'attaquant à l'une de ses causes: la représentation du parallélisme par l'entrelacement d'actions. Ce papier étudie la coopération de deux techniques d'exploration partielle basées respectivement sur les ensembles "persistants" et les "graphes de pas couvrants". Un algorithme général d'exploration partielle combinant ces deux approches et en cumulant les bénéfices est proposé.

Mots clés: système concurrent, exploration du graphe d'états, ordres partiels, ensembles persistants, graphe de pas couvrant

1 Introduction

La construction du graphe des états accessibles constitue souvent l'étape préliminaire des méthodes de vérification de systèmes concurrents. Le graphe d'états est alors analysé par les techniques de vérification automatique de type model-checking (vérification de propriétés de logique temporelle, bisimulation entre LTS). L'explosion combinatoire est la principale limitation de cette approche. Les approches développées dans cet article font partie des techniques d'ordre partiel (voir [God96] pour un panorama complet), dont le principe de base est de réduire la taille du graphe d'états en exploitant le fait que deux séquences possédant la même trace de Mazurkiewicz[Maz86] conduisent au même état. Dans le cas des ensembles persistants [WG93] ou des stubborns [Val88a], seulement un sous-ensemble des transitions sensibilisées est examiné, le graphe construit est alors un sous-graphe du graphe exhaustif. Dans le cas du Graphe de Pas Couvrant [VAM96], toutes les transitions sensibilisées sont considérées, mais les évènements indépendants sont regroupés pour construire un seul pas, dont le tir est alors atomique.

Ce papier étudie comment ces deux méthodes peuvent être utilisées simultanément. Dans l'immédiat nous avons restreint le cadre de notre étude à la préservation des blocages. Une comparaison est effectuée entre cette approche mixte et les approches initiales : "Pas" et "ensembles persistants". La section 2 rappelle les notions élémentaires des approches ordre partiel nécessaire à l'exploration par ensembles persistants et par graphes de pas couvrants et présente brièvement ces deux approches. Un algorithme général d'utilisation simultanée des ensembles persistants et des pas est donné en section 3. La section 4 présente quelques résultats expérimentaux.

2 Méthodes d'ordre Partiel

2.1 Notions de base

Definition 1. *Un* **Système de transitions étiquetées (STE)** *est un quadruplet* $\Sigma = < S, s_o, T, \rightarrow >$ *où* S *un ensemble d'états,* s_0 *un état particulier de* S *(état initial),* T *un ensemble d'étiquettes de transitions,* $\rightarrow$ *un ensemble de transitions* $(\rightarrow \subset S \times T \times S)$[1].

Les notations suivantes seront utilisées:
$s \xrightarrow{t} s'$ ssi $(s, t, s') \in \rightarrow$, $s \xrightarrow{t}$ signifie $\exists s' : s \xrightarrow{t} s'$, et $s \not\xrightarrow{t}$ pour $\neg(s \xrightarrow{t})$[2],
$s_0 \xrightarrow{\sigma} s_n$ ssi $\sigma = t_1 t_2 \ldots t_n \wedge s_0 \xrightarrow{t_1} s_1 \wedge s_1 \xrightarrow{t_2} s_2 \wedge \ldots \wedge s_{n-1} \xrightarrow{t_n} s_n$

Definition 2. Relation d'Indépendance *[GW93] :* $\wr$ *est une relation d'indépendance sur* Σ *ssi* $\forall s, s_1, s_2 \in S, \forall t_1, t_2 \in T : (t_1 \neq t_2 \wedge s \xrightarrow{t_1} s_1 \wedge s \xrightarrow{t_2} s_2 \wedge t_1 \wr t_2) \Rightarrow (s_1 \xrightarrow{t_2} s' \wedge s_2 \xrightarrow{t_1} s')$

La relation d'indépendance est étendue aux ensembles de transitions (ou aux séquences de transitions prises comme un ensemble) par $E \wr E'$ ssi $\forall (t, t') \in (E \times E') : t \wr t'$.

Si $t_1 \wr t_2$ et qu'elles sont toutes deux sensibilisées dans s alors elles peuvent être tirées dans n'importe quel ordre à partir de s, et les deux séquences $t_1 t_2$ et $t_2 t_1$ partant de s mènent dans le même état s'. Cette propriété est appelée **propriété du losange**: les transitions indépendantes commutent.

La relation complémentaire de la relation $\wr$ est la **relation de conflit ou de dépendance**, symbolisée par $\#$. Sa fermeture transitive $[\#]$ appelée **relation de conflit faible**, est une relation d'équivalence.La relation complémentaire $[\#]^C$ de la relation $[\#]$ est appelée la **relation d'indépendance forte**. On a $\# \subset [\#]$ et $[\#]^C \subset \wr$. $[\#](t) = \{t' | t' [\#] t\}$ est l'ensemble de toutes les transitions en conflit avec t.

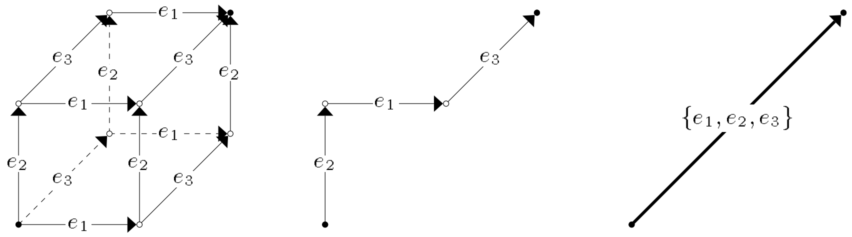
[1] Dans le cas général le STEpeut être non déterministe. Les STEconsidérés dans la suite sont déterministes

[2] s est dit alors état de blocage

Definition 3. *Deux séquences de transitions sont équivalentes au sens de Mazurkiewiczsi elles peuvent être obtenues l'une à partir de l'autre par des permutations successives de transitions adjacentes et indépendantes. Les classes d'équivalance de cette relation sont appelés des traces (de Mazurkiewicz) [Maz86]. La trace associée à la séquence w est symbolisée par $[w]$.*

Proposition 1. *Si $s_0 \xrightarrow{w_1} s_1$, $s_0 \xrightarrow{w_2} s_2$, et $[w_1] = [w_2]$, Alors $s_1 = s_2$*

À partir d'un état commun deux séquences de même trace conduisent au même état. Il peut donc être possible sous certaines conditions de n'explorer qu'un seul chemin par trace. Cette propriété est exploitée par les techniques d'"ordre partiel". Comme exemple simple, considérons un système composé de trois transitions indépendantes. Le graphe d'états associé à l'exploration de ces 3 évènements indépendants conduit à un hypercube (à gauche de la Figure 1). La proposition 1 implique l'unicité de l'état terminal. La méthode des ensembles persistants va permettre de limiter l'exploration à un seul chemin de l'état initial à l'état final. Avec la méthode des graphes de pas couvrants, les trois transitions vont être tirées simultanément pour atteindre l'état final en un seul pas.



	Exhaustif	1 chemin / Trace	**Pas**
États	2^n	$(n+1)$	2
Transitions	$n \times 2^{n-1}$	n	1

Fig. 1. Graphe d'états obtenu pour 3 transitions indépendantes

Relation d'indépendance structurelle: L'obtention d'une relation d'indépendance est une étape indispensable pour l'utilisation de techniques d'ordre partiel. Cependant il est généralement impossible de calculer la relation d'indépendance exacte. Par contre, une approximation peut être déduite de la structure du système [GP93]. Ainsi pour les réseaux de Petri place/transition une approximation de la relation d'indépendance est donnée dans [Rei85]: $t_1 \wr_s t_2$ ssi ${}^\bullet t_1 \cap {}^\bullet t_2 = \emptyset$.

Le réseau de Petri représenté par la Figure 2 sera utilisé pour illustrer les algorithmes d'exploration proposés dans ce papier. Il propose du parallélisme avec t_3 et t_4, du conflit avec t_5 et t_6, et le cas de confusion [Rei85] avec t_0 et t_1. Son graphe d'états exhaustif comporte 60 états et 160 transitions avec 4 blocages.

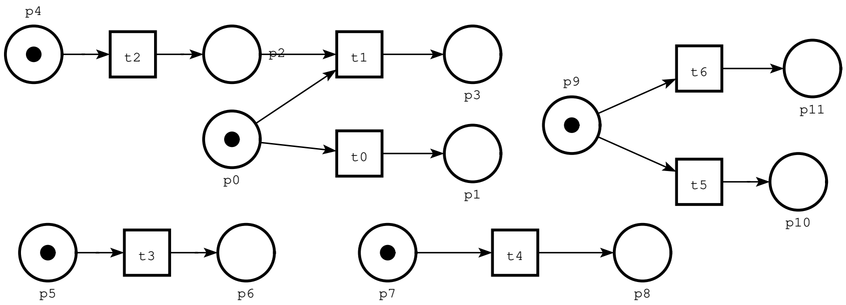


Fig. 2. Exemple de Réseau de Petri

2.2 Ensembles persistans

Les ensembles persistans sont un cas particulier des ensembles stubborns [Val88a] dans lesquels toutes les transitions sont sensibilisées. L'exploration standard par ensembles persistants conserve les blocages; de nombreuses extensions ont été proposées dans le but de préserver d'autres classes de propriétés [Val90,Val93] [GW93].

Definition 4. *Ensemble persistant [WG93] : Un ensemble T de transitions est persistant dans un état s ssi toutes les transitions qui ne sont pas dans T et sensibilisées dans s ou dans un état accessible depuis s en tirant des transitions qui ne sont pas dans T, sont indépendantes de toutes les transitions de T, i.e. ssi:*
$\forall t \in T : s \xrightarrow{t}$ *et* $\forall w \in \overline{T}^* : s \xrightarrow{w} s' \Rightarrow w \wr T$

Un ensemble persistant contient au minimum toutes les transitions qui ont à être explorées dans un état spécifique afin de découvrir tous les blocages éventuels. Si T est un ensemble persistant dans s, alors aucune transition de T ne peut être désensibilisée par une séquence de tirs de transitions qui ne sont pas dans T. Remarquons que l'ensemble des transitions sensibilisées dans un état s (noté $Enabled(s)$), est persistant dans s.

Un exemple de calcul d'ensemble persistant est donné par la propriété suivante [God90] : Si $\#(t) \subseteq Enabled(s)$ Alors $\#(t)$ est persistant dans s. Dans l'état initial du réseau de Petri de la figure 2, les ensembles $\{t2\}, \{t3\}, \{t4\} et \{t6; t5\}$ sont persistants. D'autres ensembles persistants peuvent être trouvés, il est très simple de montrer par exemple que l'union d'ensembles persistants est persistante.

Algorithme d'exploration utilisant les ensembles persistants: Cet algorithme sera noté PG dans la suite. Un algorithme générique d'exploration utilisant les ensembles persistants est donné sur la Table 1. Il est similaire à un algorithme standard d'exploration en profondeur d'abord, mis à part le fait que

$Queue \leftarrow s_0$
$H \leftarrow \{s_0\}$
$G \leftarrow \emptyset$ /* le graphe calculé */
while $NotEmpty(Queue)$ **do**
  $s \leftarrow dequeue(Queue)$
  if $Enabled(s) = \emptyset$ **then** *print "Blocage"* **else**
    $P \leftarrow A(s)$ /* choix de l'ensemble persistant */.
    foreach t **in** P **do**
      $s' \leftarrow fire(s, t)$
      $G \leftarrow G \cup \{< s, t, s' >\}$
      if $s' \notin H$ **then** $H \leftarrow H \cup \{s'\}; enqueue(Queue, s')$

Table 1. Algorithme générique d'explorationn par ensemble persistant (PG)

l'ensemble des transitions tirées d'un état s est déterminé par une fonction $A(s)$, qui choisit[3] un ensemble persistant dans s.

Proposition 2. *L'exploration par PG conserve les blocages [Val88a]*

Calcul d'un ensemble persistant: Différentes heuristiques ont été proposées [God90],[Ove81,Val89] pour définir la fonction A permettant de choisir un ensemble persistant. Une stratégie souvent utilisée est de calculer le plus petit ensemble possible. Cette stratégie minimise le branchement dans chaque état. Cette instance du PG sera dénommée l'algorithme $P_{min}G$ dans la suite. L'utilisation de $P_{min}G$ sur notre exemple de la Figure 2 produit le graphe de la Figure 3.

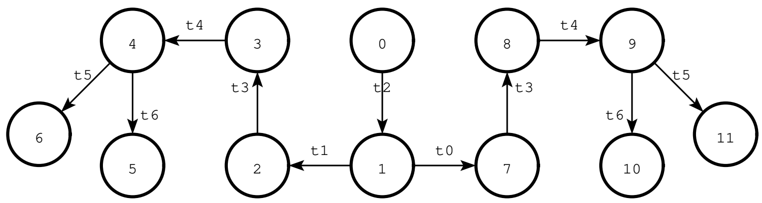


Fig. 3. $P_{min}G$: PG calculé avec une stratégie minimisant le branchement

Cependant, minimiser le branchement n'est qu'une optimisation locale; dans certains cas, le choix d'un ensemble persistant plus grand conduit à un graphe final plus petit[Val88b]. De plus il est important de remarquer que l'ensemble des transitions sensibilisées admet généralement plusieurs ensembles persistants disctincts et donc possiblement de cardinalités différentes. Par conséquent, pour calculer le plus petit ensemble persistant, il est nécessaire d'examiner toutes

[3] $A(s) \subseteq Enabled(s)$

les transitions sensibilisées comme candidates pour la génération de l'ensemble persistant. De plus, si différents ensembles persistants sont minimaux, le choix de l'un d'entre eux devient arbitraire.

2.3 Graphe de Pas couvrant

Les graphes de pas couvrants ont été introduits dans [VAM96]. Dans un graphe de pas couvrant, toutes les transitions sensibilisées sont tirées, mais les évènements indépendants sont (autant que possible) regroupés pour constituer un seul pas de transitions, dont le tir est atomique. Les graphes de pas couvrant conservent des propriétés générales d'accessibilité telles que la vivacité au sens réseau de Petri du terme [HV01], ou la préservation de blocages [VAM96]. La méthode peut également être spécialisée pour vérifier certaines propriétés telles que la bisimulation faible [VAM96] ou les sémantiques de refus [VM97].

L'idée de regrouper le tir de transitions indépendantes n'est pas récente. On trouve dans [Rei85], une notion de graphe de pas où l'on ajoute aux transitions du STE classique tous les pas de transitions possibles. Cette approche est aussi utilisée pour favoriver la convergence dans certaines méthodes d'accélération [BF00]. La principale différence est liée au fait que, dans notre approche, les pas de transitions ne sont pas ajoutés au STE classique mais substitués aux séquences de transitions "linéarisant" le pas de transitions. La décision locale de constituer un pas d'exploration n'a de sens que dans la mesure où l'on sait que toutes les linéarisations de ce pas auraient conduits à des états intermédiaires dont l'exploration n'est pas utile vis à vis de la classe de propriétés que l'on considère.

Definition 5. *Un ensemble E de transitions définit un* **pas de transitions** *par rapport à $\wr$ ssi $\forall t_1, t_2 \in E : t_1 \wr t_2$. $Step(T, \wr)$ est l'ensemble de pas de transitions dérivé de T par rapport à $\wr$.*

La relation d'accessibilité $\rightarrow$ est étendue aux pas de transitions par:

$$s \overset{\emptyset}{\twoheadrightarrow} s, \text{ et } \forall s, s' \in S, E \in Step(T, \wr) : s \overset{E}{\twoheadrightarrow} s' \text{ ssi } \forall e \in E : s \overset{e}{\longrightarrow} s_e \wedge s_e \overset{E \backslash \{e\}}{\twoheadrightarrow} s'$$

La relation d'indépendance $\wr$ est étendue aux pas par $E \wr w$ ssi $E \times ||w|| \subset \wr$, avec $||w||$ l'ensemble des transitions apparaissant dans la séquence w. La propriété du losange peut être généralisée aux séquences de transitions et aux pas [VAM96].

Definition 6. *Graphe de Pas Couvrant : Soit $\Sigma =< S, s_o, T, \rightarrow>$ un STE, $\wr$ une relation d'indépendance sur Σ, et $\# = \wr^C$.*

$\mathbb{\Sigma} =< S', s_o, \mathbb{T}, \mapsto\!\!\!\!\twoheadrightarrow>$ est un **Graphe de Pas Couvrant** *pour Σ, par rapport à $\wr$, ssi*

(1) $S' \subset S$

(2) $\mathbb{T} \subset Step(T, \wr)$

(3) $\forall s, s' \in S \cap S' : s \overset{E}{\mapsto\!\!\!\!\twoheadrightarrow} s'$ implique $s \overset{E}{\twoheadrightarrow} s'$

(4) $\forall s \in S \cap S', \forall s' \in S'$:

$$s \overset{\omega}{\longrightarrow} s' \ \ implique \ \ \begin{cases} \exists s" \in S', \exists \omega' \in T^*, \exists P_{\omega.\omega'} \in \mathbb{T}^* : \\ s' \overset{\omega'}{\longrightarrow} s", s \overset{P_{\omega.\omega'}}{\mapsto} s" \ \ et \ [w.w']_{(T,\#)} = [P_{\omega.\omega'}]_{(T,\#)} \end{cases}$$

La condition (1) signifie que chaque état du graphe de pas est un état du STE-standard. La condition (3) que chaque pas du CSG correspond à une séquence de tir dans le STE standard. Finalement, (4) exprime une "condition de couverture" entre les séquences de tir du STE standard et les séquences de pas du CSG: Toute séquence du STE peut être étendue de telle sorte qu'elle soit couverte par une séquence de pas dans le CSG. Remarquons que tout STE peut être vu comme un CSG en prenant $\wr = \emptyset$.

Automate Trace et CSG : Les automates trace ont été introduits dans [God96] et permettent de vérifier certaines propriétés de sûreté. Pour générer un automate trace [God96] l'algorithme d'exploration à base d'ensembles persistants est modifié, pour résoudre l' "ignoring problem"[Val88a]. Nous allons voir qu'un CSG peut très facilement, par dépliage, être ramené à un automate trace.

Definition 7. *Soit* $G_E =< S, s_o, T, \to>$ *un STE et* $\wr$ *une relation d'indépendance.*

$G_R =< S', s_o, T, \to>$ *est un* **automate trace** *pour* G_E*, si pour toute séquence* w_E *de transitions de* G_E*, il existe une séquence* w_R *de transitions de* G_R *telle que* w_R *soit une linéarisation d'une trace définie par une extension de* w_E*, i.e.* $w_E \in Pref([w_R])$*, avec* $Pref([w])$ *l'ensemble des préfixes des séquences de* $[w]$*.*

Principe du dépliage: On peut obtenir un automate trace à partir d'un CSG en "dépliant" chacun de ses pas. A chaque pas est substitué la séquence d'états et de transitions associée à l'une [4] des linéarisations possibles du pas.

Graphe de Pas Couvrant : La Table 2 donne la structure d'un algorithme pour calculer un CSG. L'algorithme est paramétré par une relation d'indépendance $\wr$. Les transitions sensibilisées sont séparées en deux sous-ensembles grâce aux fonctions T_U et T_M. T_u contient les transitions à explorer d'une façon standard, et T_m contient celles dont l'exploration sera menée par des pas. Ces transitions seront nommées "fusionnables" dans la suite. Π_{T_M} est l'ensemble des pas de transitions, construit par la fonction Π. Une instance possible de cet algorithme avec les différentes fonctions est donnée ci-après.

Graphe de Pas de Conflits Croisés : D'après [VAM96], l'ensemble des pas maximaux peut être construit à l'aide de l'*orthoproduit* [DHP90] sur les classes d'équivalence de $\#\!\!\#$ (la fermeture transitive de la relation de conflit).L'orthoproduit de $\mathbb{E} = \{E_1, E_2, \ldots E_n\}$ est l'ensemble $\Pi_C(\mathbb{E}) = \{\{e_1, e_2, \ldots e_n\} \mid (e_1, e_2, \ldots e_n) \in E_1 \times E_2 \ldots \times E_n\}$. L'algorithme est obtenu en prenant: $T_M(T, \wr) = \{t \in T : t'\#t \Rightarrow t' \in T\}$, $T_U(T, \wr) = T \setminus T_M(T, \wr)$ et $\Pi(T_M(T, \wr)) = \Pi_C((T, \wr)/\#\!\!\#)$ avec Π_C l'orthoproduit. La construction du graphe de pas de conflit croisés sur notre exemple de la Figure 2 produit le CSG de la Figure 4.

[4] Chaque pas P peut être linéarisé en $Cardinal(P)!$ séquences differentes

$Queue \leftarrow s_0$
$H \leftarrow \{s_0\}$
$G \leftarrow \emptyset$ /* le graphe calculé */
while $NotEmpty(Queue)$ **do**
  $s \leftarrow dequeue(Queue)$
  $T \leftarrow Enabled(s)$;
  if $T = \emptyset$ **then** *print "Blocage"* **else**
    $T_u \leftarrow T_U(T, l)$
    $T_m \leftarrow T_M(T, l)$
    foreach t **in** T_u **do** /* Exploration Standard */
      $s' \leftarrow fire(s, t)$
      $G \leftarrow G \cup \{< s, \{t\}, s' >\}$
      if $s' \notin H$ **then** $H \leftarrow H \cup \{s'\}; enqueue(Queue, s')$
    $\Pi_{T_m} \leftarrow \Pi(T_m, l)$
    foreach π **in** Π_{T_m} **do** /* Exploration par Pas */
      $s'' \leftarrow s$
      foreach p **in** π **do** $s' \leftarrow fire(s'', p); s'' \leftarrow s'$
      $G \leftarrow G \cup \{< s, \pi, s'' >\}$
      if $s'' \notin H$ **then** $H \leftarrow H \cup \{s''\}; enqueue(Queue, s'')$

Table 2. Algorithme générique de Graphe de Pas Couvrant

3 Pas Persistants, utilisant *PG* et *CSG*

Les ensembles persistants et les pas contribuent à la réduction du graphe d'états par différents aspects. Dans la suite nous proposons une méthode d'exploration utilisant les avantages des deux techniques. La préservation des blocages est prouvée, puis ses bénéfices en termes de réduction du graphe d'états sont discutés.

3.1 Algorithme *PSG*

Le graphe de pas utilise l'indépendance des transitions pour ne faire qu'un seul pas entre l'état de départ et l'état d'arrivée. Les ensembles persistants sont eux utilisés pour ne pas tirer toutes les transitions sensibilisées mais uniquement un sous-ensemble de celles-ci.

Une approche qui combinerait les deux, utiliserait les ensembles persistants pour ne pas tirer toutes les transitions, et les pas pour grouper autant que possible les transitions que l'on a décidé de tirer.

La Table 3 présente la structure d'un algorithme d'exploration combinant pas et ensembles persistants, qui sera appelé Algorithme *PSG* dans la suite. Cet algorithme est semblable à l'algorithme des graphes de pas couvrant décrit dans la table 2. Les principales différences sont:

- *PSG* utilise une fonction de calcul d'ensembles persistants comme pour les *PG* (cf. Table 1);

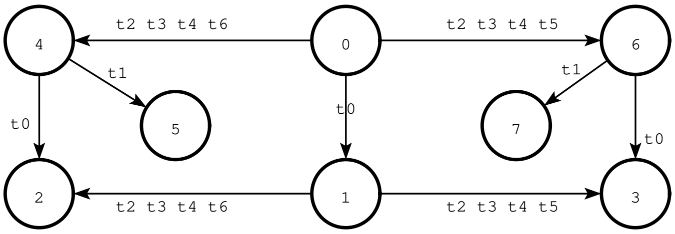


Fig. 4. CSG utilisant les conflits croisés

- Les transitions non fusionnables T_U ne sont pas forcément tirées. Elles ne sont effectivement tirées que lorsque $Enabled(s)$ n'admet pas d'ensemble persistant propre (i.e. $T_m = \emptyset$). Dans ce seul cas une exploration classique est effectuée.
- Dans les CSG toutes les transitions de T_M sont tirées. Tandis que dans les PSG seul un sous-ensemble de T_M, défini par la fonction ensemble persistant $A()$, sera considéré

3.2 Préservation des blocages

Proposition 3. *Les explorations PSG conservent les blocages.*

La démonstration de cette proposition 3 est similaire à la démonstration faite pour les CSG dans [VAM96] (Lemme de normalisation, puis démonstration par récurrence sur la longueur de la séquence menant au blocage...). Une idée intuitive du déroulement de l'algorithme permet de "visualiser" la conservation des blocages; Un ensemble persistant est utilisé pour ne pas tirer toutes les transitions sensibilisées mais seulement un sous-ensemble de celles-ci. Sur ce sous-ensemble on ne tire pas séquentiellement chaque transition mais au contraire des pas sont calculés enfin d'éviter tous les états intermédiaires.

De la même façon que PG est un schéma général d'algorithme admettant différentes instanciations ($P_{min}G$ est l'une d'entre elles), nous allons maintenant envisager différentes instanciations de PSG en choisissant des fonctions $A()$ et $\Pi()$ spécifiques. La première instance vise à minimiser le branchement et s'apparente à $P_{min}G$, la seconde favorisant les pas se rapproche de CSG, et la troisième est de nature plus hybride.

3.3 Comparaison de *PG* et *PSG*

D'après la proposition 3, l'algorithme PSG général conserve les blocages quelle que soit la fonction $A()$ utilisée pour le calcul de l'ensemble persistant. Dans la suite nous montrons que quelle que soit la fonction $A()$, le graphe construit par PSG est plus petit ou de même taille que le graphe construit avec PG.

```
Queue ← s_0
H ← {s_0}
G ← ∅ /*  Graphe calculé  */
while NotEmpty(Queue) do
    s ← dequeue(Queue)
    T ← Enabled(s)
    if T = ∅ then print "Blocage" else
        T_u ← T_U(T, ≀)
        T_m ← T_M(T, ≀)
        if T_m = ∅ then
            foreach t in T_u do
                s' ← fire(s, t); G ← {G} ∪ {< s, {t}, s' >}
                if s' ∉ H then H ← H ∪ {s'}; enqueue(Q, s')
        else
            P ← A(s) ∩ T_m     /*  Choix d'un ensemble Persistant  */
            Π_P ← Π(P, ≀)  /*  Choix des Pas  */
            foreach π in Π_P do
                q'' ← q
                foreach p in π do s' ← fire(s'', p); s'' ← s'
                G ← G ∪ {< s, π, s'' >}
                if s'' ∉ H then H ← H ∪ {s''}; enqueue(Queue, s'')
```

Table 3. Algorithme Générique de Graphe de Pas Persistant

Proposition 4. PSG **généralise** PG

Idée de la Démonstration. Tout PG est un PSG: En considérant $\Pi(St, \wr) = \{\{t\} \mid t \in St\}$ dans l'algorithme de la Table 3, le graphe généré est exactement le même que le graphe obtenu avec l'algorithme du PG de la Table 1 (en supposant que les deux utilisent la même fonction $A()$).

Il est donc évident que le PSG peut produire des graphes plus petits que les graphes produits par PG. Cependant, le PSG étant plus général, tout PSG n'est pas la contraction d'un PG.

Nous considérons un algorithme PSG particulier noté $P_{min}SG$ dans la suite: Les pas sont construits en utilisant l'orthoproduit π_C. La fonction $A()$ choisit un ensemble persistant minimal (donc minimise le branchement local) et est définie par:

$A(s) =$ **Si** $F_E \neq \emptyset$ **alors** F_E **sinon** $\#\!\!\#(t_{min})$, avec
- F_E le plus grand sous-ensemble des transitions sensibilisées, pour lequel aucune transition n'est en conflit avec une transition de $Enabled(s)$, donc tel que $\#\!\!\#(t) = \{t\}$.
- t_{min} est choisie dans $Enabled(s)$ telle qu'elle minimise $Card(\#\!\!\#(t))$

Remarquons que l'heuristique utilisée par le $P_{min}SG$ est la même que celle utilisée pour le $P_{min}G$. Notons que, comme le $P_{min}G$, le $P_{min}SG$ n'est pas déterministe (le choix entre différents ensembles persistant minimaux est arbitraire). L'utilisation de $P_{min}SG$ sur notre exemple produit le graphe de la figure 5; Le premier pas tire toutes les transitions de F_E.

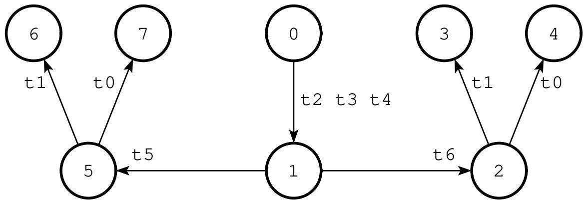


Fig. 5. $P_{min}SG$

Proposition 5. *Tout $P_{min}SG$ est la contraction d'un ou plusieurs $P_{min}G$.*

Démonstration. Nous allons montrer comment construire un $P_{min}G$ à partir d'un $P_{min}SG$. Pour cela nous utilisons la fonction de calcul des ensembles persistants utilisée pour le $P_{min}SG$ pour définir une nouvelle fonction pour construire $P_{min}G$. Lorsque l'ensemble utilisé pour $P_{min}SG$ contient seulement des transitions libres de tout conflit, tout sous-ensemble est persistant, et nous pouvons prendre un ensemble persistant avec une seule transition dans chaque état intermédiaire. La preuve que cet ensemble est minimal dans chaque état est triviale, et le graphe résultant est plus grand.

Compte tenu du non-déterminisme de l'algorithme PG il parait difficile de trouver un algorithme PSG qui soit plus efficace dans tous les cas qu'un algorithme PG. Ce problème général est illustré avec le réseau de Petri de la Figure 6, constitué de composants non connectés : *Loop*, $\mathcal{N}_0$, $\mathcal{N}_1$, $\ldots\mathcal{N}_n$.

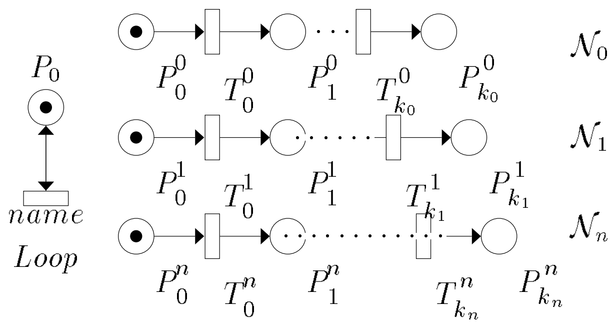


Fig. 6. Comparaison de $P_{min}SG$ et $P_{min}G$

Différentes explorations par ensembles persistants **minimaux** peuvent être construites: L'exploration qui commencerait par choisir la transition du composant *Loop* comme ensemble persistant, serait optimale avec seulement 1 état et 1 arc. L'exploration la plus catastrophique choisirait le composant *Loop* en dernier et conduirait à un graphe avec $\Sigma_{i=0}^{i=n} k_i + 1$ états. Ces deux explorations sont du type $P_{min}G$ mais conduisent à des résultats très différents à cause du non déterminisme de la construction. Une exploration par $P_{min}SG$ est déterministe sur cet exemple, et conduit à un graphe ayant $Max\{k_i + 1 : i \in [1, n]\}$ états, ce qui, comme le prouve la propriété 5, est bien inférieur au pire des cas obtenu avec $P_{min}G$.

3.4 Comparaison de CSG et PSG

Bien que PSG ne soit pas une généralisation de CSG (car un CSG n'est pas un PSG), nous pouvons comparer les graphes obtenus par les deux types d'exploration. Dans l'exploration PSG, les transitions non fusionnables T_U ne sont pas explorées dans tous les cas. Cela a pour conséquence la perte de la propriété de couverture que l'on avait pour les CSG, mais va nous permettre de construire des graphes plus petits que ceux obtenus par l'exploration CSG. Pour cela nous allons introduire une nouvelle instanciation de PSG, qui construit des pas les plus grand possibles afin d'éviter au maximum les états intermédiaires. Cet algorithme, noté $PS_{max}G$ dans la suite, est defini par la fonction $A(s) = Enabled(s)$ pour avoir P (l'ensemble sur lequel sont fait les pas) aussi grand que possible. Les pas sont calculés en utilisant l'orthoproduit π_C. L'application de $PS_{max}G$ sur notre exemple donne le graphe de la Figure 7.

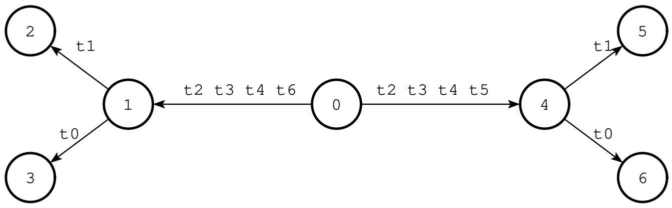


Fig. 7. $PS_{max}G$

Proposition 6. Le $PS_{max}G$ est toujours plus petit que le CSG

Démonstration. Puisque T_M est persistant, $PS_{max}G$ ne tire pas les transitions de T_U. Le graphe obtenu est donc trivialement un sous-graphe du CSG.

3.5 Un PSG hybride

Les algorithmes $P_{min}SG$ et $PS_{max}G$ diffèrent seulement par leur fonction de choix de l'ensemble persistant; qui retourne l'ensemble minimal dans un cas, et l'ensemble total des transitions sensibilisées dans le second cas. La démonstration de la proposition 3 assure que toutes les solutions intermédiaires possibles préservent les blocages. Parmi celles-ci, nous en considérons une troisième que nous notons $HPSG$, qui présente l'avantage d'être déterministe dans le sens où le graphe ne dépend plus d'un choix arbitraire d'ensemble persistant.

L'algorithme PSG **Hybride**, nommé $HPSG$, est défini par:

> $A(s) =$ **Si** $F_E \neq \emptyset$ **alors** F_E **sinon** $Enabled(s) - F_E$
> avec F_E le plus grand sous-ensemble de transitions sensibilisées qui sont indépendantes de toute transition de $Enabled(s)$, donc telles que $[\#](t) = \{t\}$.

Le graphe résultant obtenu sur notre exemple est représenté figure 8.

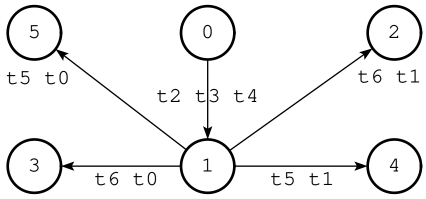


Fig. 8. Exploration $HPSG$

4 Résultats expérimentaux

Les résultats comparatifs pour notre exemple sont résumés dans la table suivante, en terme de nombre d'états et de transitions, pour tous les algorithmes developpés. Observons que toutes les stratégies PSG améliorent ou égalent les méthodes $P_{min}G$ et CSG.

	$Exhaustif$	$P_{min}G$	CSG	$P_{min}SG$	$PS_{max}G$	$HPSG$
$Nbre\ d'etats$	60	12	8	8	7	6
$Nbre\ de\ transitions$	160	11	9	7	6	5

Les mêmes algorithmes ont été appliqués sur des modèles de taille significative et d'un interêt pratique. Les résultats sont résumés dans la table suivante. Le premier modèle est une version du scheduler de Milner à 300 sites [Mil80], le second est le modèle classique des philosophes avec 8 philosophes. Le dernier modélise l'algorithme distribué d'exclusion mutuelle de Naimi-Trehel avec 5 sites [NT87]. $P_{min}SG$, $PS_{max}G$ et $HPSG$ donnent les mêmes résultats sur ces exemples, mais notre exemple d'étude montre que ce n'est pas toujours le cas.

Il peut être observé sur ces exemples que le PSG améliore le meilleur des cas entre le PG et le CSG.

$Modele$	$Exhaustif$	$P_{min}G$	CSG	PSG
$Scheduler$ 300	$2^n * n = 6.10^{92}$	1 394	301	301
$Philosophe$ 8	103 681	233	31 231	227
$Naimi - Trehel$ 5	202 500	40 006	52 681	40 001

Pour être expérimentés, les différents algorithmes d'exploration présentés dans le papier ont été implémentés dans un outil proposant différents algorithmes d'exploration et d'analyse pour les réseaux de Petri, et les réseaux de Petri temporels. Cet outil est téléchargeable à l'adresse `http://www.laas.fr/tina`.

5 Conclusion

Les techniques d'ordre partiel ont été développées pour limiter l'explosion combinatoire en s'attaquant à l'une de ses causes: la représentation du parallélisme par l'entrelacement d'actions. Ce papier a étudié la coopération de deux de ces approches: les explorations partielles basées sur les ensembles "persistants" et les "graphes de pas couvrants". Un algorithme général d'exploration partielle mêlant ces deux approches et en cumulant les bénéfices est proposé: le PSG. Différentes instanciations de cet algorithme ont été comparées à l'utilisation séparée des ensembles persistants ou des pas.

Il est montré que pour des fonctions d'ensemble persistant analogues, un graphe construit par l'algorithme PSG peut être plus petit qu'un graphe construit par les algorithmes connus PG et CSG. Bien que nous ne puissions pas exhiber de fonction persistant $A()$ générant à tous les coups un PSG meilleur que le PG et le CSG, l'évaluation de ces explorations sur des exemples significatifs montrent que ce but est souvent atteint dans la pratique.

Nos études futures consisteront entre autres à comparer les "trace automata" et les CSG, et en particulier les algorithmes qui permettent de les obtenir (exploration à base d'ensembles persistants, ou par dépliage de CSG). Plus généralement nous nous intéressons à d'autres types de propriétés pour lesquelles des explorations partielles adéquates peuvent être proposées.

References

[BF00] J-P. Bodeveix and M. Filali. Expérimentation de méthodes d'accélération. In *Journées Formalisation des Activités Concurrentes*, 2000.

[DHP90] D. Freestone D. H. Pitt. The derivation of conformance tests from lotos specifications. *IEEE Transactions on Software Engineering*, 1990.

[God90] P. Godefroid. Using partial orders to improve automatic verification methods. In *Proceedings of CAV'90*, pages 321–340. ACM, DIMACS volume 3, 1990.

[God96] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems.* LNCS 1032, 1996.

[GP93] P. Godefroid and D. Pirottin. Refining dependencies improves partial-order verification methods. In *Proceedings of CAV'93*. Springer Verlag, LNCS 697, 1993.

[GW93] P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. *Formal Methods in System Design*, 2(2):149–164, 1993.

[HV01] S. Haddad and F. Vernadat. *Méthodes d'analyse des réseaux de Petri in Les Réseaux de Petri. Modèles fondamentaux.* Hermes Science, Traité IC2 Information-Commande-Communication, NISBN 2-7462-0250-6, 2001.

[Maz86] A. Mazurkiewicz. Trace theory. In *Petri Nets: Applications and Relationships to Other Model of Concurrency, Advances in Petri nets 1986, Part II; Proceedings of an advanced Course*, pages 279–324. Springer Verlag, LNCS 255, 1986.

[Mil80] R. Milner. *A Calculus of communicating Systems*, chapter A case study in synchronisation, and proof techniques, pages 32–46. LNCS 92. Springer-Verlag, 1980.

[NT87] M. Naimi and M. Trehel. An improvement of the log N distributed algorithm for mutual exclusion. In *7th International Conference on Distributed Computing Systems*, pages 371–377, Washington, D.C., USA, September 1987. IEEE Computer Society Press.

[Ove81] Overman. *Verification of concurrent systems: function and timing.* PhD thesis, University of California, 1981.

[Rei85] W. Reisig. *Petri Nets : an Introduction.* Springer-Verlag,EATCS, 1985.

[Val88a] A. Valmari. Error detection by reduced reachability graph generation. In *Proceedings of Application and Theory of Petri Nets.* Springer Verlag, LNCS, 1988.

[Val88b] A. Valmari. *State Space Generation: Efficiency and Practicality.* PhD thesis, Tampere University of Technology, 1988.

[Val89] A. Valmari. Stubborn sets for reduced state space generation. In *Proceedings of Application and Theory of Petri Nets.* Springer Verlag, LNCS 483, 1989.

[Val90] A. Valmari. A stubborn attack on state explosion. In *Proceedings of CAV'90*, pages 25–42. ACM, DIMACS volume 3, 1990.

[Val93] A. Valmari. On-the-fly verification with stubborn sets. In *Proceedings of CAV'93*, pages 397–408. Springer-Verlag, LNCS 697, 1993.

[VAM96] F. Vernadat, P. Azéma, and F. Michel. Covering step graph. In *Proceedings of Application and Theory of Petri Nets 96.* Springer Verlag, LNCS 1091, 1996.

[VM97] F. Vernadat and F. Michel. Covering step graph preserving failure semantics. In *Proceedings of Application and Theory of Petri Nets 97.* Springer Verlag, LNCS 1248, 1997.

[WG93] P. Wolper and P. Godefroid. Partial-order methods for temporal verification. In *Proceedings of Concur'93.* LNCS 575, 1993.