Un dépliage par processus pour calculer le préfixe complet des réseaux de Petri

Médésu Sogbohossou — Antoine Vianou

Département Génie Informatique et Télécommunications
École Polytechnique d'Abomey-Calavi, 01 BP 2009 Cotonou, BENIN
{medesu.sogbohossou,antoine.vianou}@epac.uac.bj

RÉSUMÉ. La technique d'ordre partiel du dépliage représente implicitement l'espace d'état d'un réseau de Petri (RdP), en conservant notamment les relations de concurrence entre les événements. Cela permet de contenir le phénomène de l'explosion combinatoire en cas de forte concurrence. Un préfixe complet de dépliage sert à couvrir tout l'espace d'état d'un RdP borné: son calcul suivant l'approche classique se base sur le concept d'ordre adéquat, ne prenant directement en compte que les RdP saufs. Dans cet article, une nouvelle approche indépendante du concept d'ordre adéquat et fidèle à la sémantique d'ordre partiel, consiste à créer les événements du dépliage dans le contexte d'un unique processus à la fois. Les résultats des tests sont concluants pour les RdP saufs et non saufs. Pour améliorer la compacité du préfixe obtenu, deux solutions sont présentées.

ABSTRACT. The partial-order technique of the unfolding implicitly represents state-space of a Petri net (PN), by in particular preserving the concurrency relations between the events. That makes it possible to contain state-space explosion problem in case of strong concurrency. A complete prefix of unfolding is used to cover all the state-space of a bounded PN: its computation according to the classical approach is based on the concept of adequate order, taking directly into account only safe PN. In this paper, a new approach independent of the concept of adequate order and faithful to the partial-order semantics, consists in creating the events of the unfolding in the context of a single process at the same time. The results of the tests are conclusive for safe and nonsafe PN. To improve compactness of the prefix obtained, two solutions are presented.

MOTS-CLÉS : réseaux de Petri bornés, préfixe complet de dépliage, ordre adéquat, processus alternatifs

KEYWORDS : bounded Petri nets, complete prefix of unfolding, adequate order, alternative processes

1. Introduction

Les réseaux de Petri (RdP) [11] constituent un des formalismes bien connus pour modéliser de manière compacte et explicite la concurrence et la synchronisation entre composantes dynamiques des systèmes à événement discret. Le modèle établi permet alors de conduire des vérifications de propriétés sur le système représenté, en passant généralement par la construction de l'espace d'état. Toutefois, l'énumération exhaustive des états globaux, sous forme d'un graphe d'état, est exponentielle avec la taille du modèle en cas de concurrence : on parle d'explosion combinatoire. Les techniques dites d'*ordre partiel* constituent un des moyens utilisés pour endiguer ce problème. Ainsi, les réductions d'ordre partiel [15, 16, 17] visent à générer un espace d'état réduit, en économisant les entrelacements des événements concurrents au cours de la construction du graphe d'état. La technique d'ordre partiel du dépliage [2] est une alternative qui préserve une représentation des états globaux, mais de manière implicite en conservant notamment les relations de concurrence entre les états locaux des composantes et entre les événements. Des travaux récents [12, 1, 13] montrent que ces différentes techniques sont toujours en cours d'amélioration. Par exemples, l'article [12] propose un compromis entre rapidité [5] et moindre coût mémoire [8] du dépliage, et l'article [13] intègre certains atouts des dépliages des RdP aux techniques de réduction d'ordre partiel dites dynamiques.

Le calcul d'un préfixe fini du dépliage permet de capturer l'espace d'état du réseau de Petri : ce préfixe est alors dit *complet* [10, 6]. L'approche classique [6] (et ses généralisations dans [9, 1]) de calcul d'un préfixe complet se base sur le concept d'*ordre adéquat* qui exclut la catégorie des RdP non saufs. Dans cet article, un nouvel algorithme se passant du concept d'ordre adéquat est défini : il donne des résultats satisfaisants pour les réseaux saufs. De plus, cet algorithme prend en compte le dépliage des RdP bornés non saufs, avec le souci de préserver la concurrence. En effet, pour cette classe de réseaux, l'approche actuelle [6] consiste à passer par une conversion vers un modèle sauf, ce qui fait perdre l'expression des relations de concurrence.

La particularité du nouvel algorithme consiste à créer les événements du dépliage dans le contexte d'un unique processus à la fois, à l'instar de travaux précédents [14] qui sont valables pour une classe restreinte de réseaux temporels ; ici, aucune restriction ne s'applique à la forme des processus générés pour obtenir un préfixe complet qui soit fini. Ainsi, les événements ne sont plus créés en permettant le développement simultané de plusieurs processus en conflit, ce qui évite le recours au concept d'ordre adéquat. Pour améliorer la compacité du préfixe obtenu, nous esquissons des solutions en envisageant d'une part la détection et la suppression de redondance entre processus alternatifs, et d'autre part en éliminant les auto-conflits apparaissant dans les réseaux non saufs.

La section 2 rappelle les définitions sur les RdP et le dépliage. Ensuite, la section 3 présente le nouvel algorithme, son principe, les résultats de sa mise en œuvre, ainsi que les améliorations qui peuvent être intégrées à l'algorithme. Enfin, la section 4 présente la synthèse des résultats et énonce les perspectives à plus long terme.

2. Rappels

2.1. Réseaux de Petri

Définition 1. *Un* réseau de Petri *(ou* RdP*) est un triplet* $N \stackrel{\text{def}}{=} < P, T, W >$ *:*

– *P et T sont resp. les ensembles des places et des transitions : $P \cap T = \emptyset$;*

– *$W \subseteq P \times T \cup T \times P$ est la relation de flux.*

Pour un RdP destiné au calcul d'un espace d'état fini, P et T sont finis. L'ensemble des nœuds prédécesseurs (resp. successeurs) d'un nœud $x \in P \cup T$ est noté ${}^\bullet x \stackrel{\text{def}}{=} \{y \in P \cup T \mid (y,x) \in W\}$ (resp. $x^\bullet \stackrel{\text{def}}{=} \{y \in P \cup T \mid (x,y) \in W\}$).

Un *marquage* est une application $m : P \to \mathbb{N}$: il est interprété comme un état global du système. Le doublet $< N, m_0 >$ représente le RdP N de marquage initial m_0.

Une transition t est *sensibilisée* par un marquage m, ce qui est noté $m \xrightarrow{t}$, si ${}^\bullet t \subseteq m$. Le *tir* de t conduisant au marquage $m' = m \backslash {}^\bullet t \cup t^\bullet$ est noté $m \xrightarrow{t} m'$. Soit $m_0 \xrightarrow{\sigma} m$ t.q. $\sigma = t_1 t_2 ... t_n \in T^*$: σ désigne une séquence de tirs à partir de m_0.

L'*ensemble d'accessibilité* du RdP marqué $< N, m_0 >$ est défini par : $A(N, m_0) \stackrel{\text{def}}{=} \{m \mid \exists \sigma \in T^*, m_0 \xrightarrow{\sigma} m\}$. $< N, m_0 >$ est *borné* si $\exists n \in \mathbb{N}$ t.q. pour tout marquage $m \in A(N, m_0)$, $m(p) \leq n$, $\forall p \in P$. L'ensemble d'accessibilité est fini *ssi* le RdP est borné. Un marquage m *sauf* signifie $m(p) \leq 1, \forall p \in P$: pour un RdP sauf, tous les marquages accessibles sont saufs.

$A(N, m_0)$ fini se représente sous la forme d'un *graphe des marquages* (ou *graphe d'état*) : les nœuds sont les marquages et les arcs représentent les tirs de transition entre couples de marquages directement accessibles.

2.2. Dépliage

Un dépliage prend la forme d'un RdP $O \stackrel{\text{def}}{=} < B, E, F >$ acyclique, dénommé *réseau d'occurrence*, t.q. : $\forall b \in B, |{}^\bullet b| \leq 1$, $\forall e \in E, {}^\bullet e \neq \emptyset$ et $e^\bullet \neq \emptyset$, et F^+ (la fermeture transitive de F) est une relation d'ordre strict.

B (resp. E) est dénommé ensemble des *conditions* (resp. ensemble des *événements*). Pour $e \in E$, ${}^\bullet e$ (resp. $e^\bullet$) forme les *pré-conditions* (resp. *post-conditions*) de e.

On définit : $Min(O) \stackrel{\text{def}}{=} \{b \in B \mid {}^\bullet b = \emptyset\}$ et $Max(O) \stackrel{\text{def}}{=} \{b \in B \mid b^\bullet = \emptyset\}$.

Trois types de relations sont définis entre deux nœuds quelconques de O :

– la causalité ($\prec$) : $\forall x, y \in B \cup E, x \prec y$ ssi $(x,y) \in F^+$;

– le conflit ($\sharp$) : $\forall e_1, e_2 \in E\ (e_1 \neq e_2),\ e_1 \sharp e_2$ si ${}^\bullet e_1 \cap {}^\bullet e_2 \neq \emptyset$. De plus, si $e_1 \sharp e_2$, alors $\forall x, y \in B \cup E,\ e_1 \preceq x \wedge e_2 \preceq y \ \Rightarrow\ x \sharp y$;

– et la concurrence ($\wr$) : $\forall x, y \in B \cup E\ (x \neq y)$, $x \wr y$ ssi $\neg((x \prec y) \vee (y \prec x) \vee (x \sharp y))$.

Soit $B' \subseteq B$ t.q. $\forall b, b' \in B', b \neq b' \Rightarrow b \wr b'$: B' est appelé une *coupe*.

Soient le réseau d'occurrence $O_F \stackrel{\text{def}}{=} < B_F, E_F, F_F >$ et la fonction d'étiquetage $\lambda_F : B_F \cup E_F \to P \cup T$ t.q. $\lambda(B_F) \subseteq P$ et $\lambda(E_F) \subseteq T$.

Définition 2. Le dépliage (exhaustif) *[14]* $Unf_F \stackrel{\text{def}}{=} < O_F, \lambda_F >$ *de* $< N, m_0 >$ *est donné par :*

1) *$\forall p \in P$, si $m_0(p) \neq \emptyset$, alors $B_p \stackrel{\text{def}}{=} \{b \in B_F \mid \lambda_F(b) = p \wedge {}^\bullet b = \emptyset\}$ et $m_0(p) = |B_p|$;*

2) *$\forall B_t \subseteq B_F$ t.q. B_t est une coupe, si $\exists t \in T, \lambda_F(B_t) = {}^\bullet t \wedge |B_t| = |{}^\bullet t|$, alors :*

- ***a)*** *$\exists! e \in E_F$ t.q. ${}^\bullet e = B_t \wedge \lambda_F(e) = t$;*
- ***b)*** *si $t^\bullet \neq \emptyset$, alors $B'_t \stackrel{\text{def}}{=} \{b \in B_F \mid {}^\bullet b = \{e\}\}$ est t.q. $\lambda_F(B'_t) = t^\bullet \wedge |B'_t| = |t^\bullet|$;*
- ***c)*** *si $t^\bullet = \emptyset$, alors $B'_t \stackrel{\text{def}}{=} \{b \in B_F \mid {}^\bullet b = \{e\}\}$ est t.q. $\lambda_F(B'_t) = \emptyset \wedge |B'_t| = 1$;*

3) *$\forall B_t \subseteq B_F$, si B_t n'est pas une coupe, alors $\nexists e \in E_F$ t.q. ${}^\bullet e = B_t$.*

La définition 2 exprime succinctement l'algorithme d'un dépliage exhaustif. Les articles de Engelfriet [2] et Esparza et al. [6] par exemples en donnent une définition plus explicite.

Soit $E \subset E_F$. Le réseau d'occurrence $O \stackrel{\text{def}}{=} < B, E, F >$ associé à E tel que $B \stackrel{\text{def}}{=} \{b \in B_F \mid \exists e \in E, b \in {}^\bullet e \cup e^\bullet\}$, $F \stackrel{\text{def}}{=} \{(x, y) \in F_F \mid x \in E \vee y \in E\}$ et $Min(O) = Min(O_F)$ est *un préfixe* de O_F. Par extension, $Unf \stackrel{\text{def}}{=} < O, \lambda >$ (avec λ, la restriction de λ_F à $B \cup E$) est un préfixe du dépliage Unf_F.

Si les événements E du préfixe de dépliage Unf sont tels que $\forall (e, e') \in E \times E$, on a $\neg(e \sharp e')$, alors E constitue un *processus*.

Soit E_i un processus fini. Le réseau $C_i \stackrel{\text{def}}{=} < B_i, E_i, F_i >$ associé est appelé *réseau causal*. Il vérifie : $\forall b \in B_i, |b^\bullet| \leq 1$. $Max(C_i)$ est l'état final de C_i : il correspond au marquage final du comportement exprimé par E_i, à savoir le multi-ensemble de jetons du RdP résultant de $\lambda(Max(C_i))$, et qui est noté $Mark(E_i)$.

La *configuration locale* d'un événement $e_i \in E_i$ est le processus $E_{e_i} \stackrel{\text{def}}{=} \{e_j \in E_F \mid e_j \prec e_i \vee e_j = e_i\}$. Le marquage $Mark(E_{e_i})$ sera appelé *marquage propre* de e_i.

A l'instar d'un graphe d'état, un préfixe fini de dépliage peut capturer l'espace d'état du RdP : le préfixe est alors dit *complet*.

Définition 3. *Un préfixe* $Unf \stackrel{\text{def}}{=} << B, E, F >, \lambda >$ *de* Unf_F *est* complet *lorsque, pour tout marquage accessible* m *de* $< N, m_0 >$*, il existe un processus* $E_i \subseteq E$ *t.q. :*

1) $m = Mark(E_i)$,

2) si $\exists t \in T$ *t.q.* ${}^\bullet t \subseteq m$*, alors* $\exists e \in E$ *t.q.* ${}^\bullet e \subseteq Max(C_i) \wedge \lambda(e) = t$.

En pratique, pour représenter tous les marquages de $A(N, m_0)$ sans énumération exhaustive de l'espace d'état, la création de chaque événement du dépliage est soumise à une comparaison entre son marquage propre et ceux des événements déjà produits [10, 6].

2.3. Principe des algorithmes de calcul de préfixe complet

Au cours du calcul d'un dépliage complet, un événement à produire est candidat *cut-off* lorsque son marquage propre est équivalent à celui d'un événement (que nous qualifions de *référence*) précédemment ajouté dans le dépliage : ceci autorise à ne pas calculer les événements successeurs d'un événement cut-off. Mais contrairement à un graphe d'état où une comparaison de marquage suffit, avoir deux marquages propres identiques n'est qu'une des conditions nécessaires pour identifier effectivement un cut-off.

L'algorithme de Esparza *et al.* [6], plus général et plus optimal que celui proposé par McMillan [10], se base sur le concept d'*ordre adéquat*. Une relation d'ordre adéquat $<$ identifie les événements cut-off en comparant les configurations locales des événements produits au cours du dépliage : c'est l'élément (une configuration locale) *plus grand* qui peut être cut-off. Bien entendu, le calcul du dépliage complet consiste à produire les événements possibles un à un suivant l'ordre $<$ de leurs configurations locales, afin de toujours produire en premier les événements de référence potentiels. L'adéquation de la relation $<$ devra être préservée par toute extension en événement d'une configuration : ceci n'est garanti que pour les réseaux saufs (cf. annexe A pour plus de détails).

Les travaux subséquents [7, 8, 9, 12, 1] sur le dépliage reposent également sur ce concept. La prise en compte des RdP non saufs passe par une conversion en RdP sauf [6], avec pour conséquence une perte de concurrence qui peut nuire à la compacité du résultat.

3. Le nouvel algorithme

3.1. Processus alternatifs

Soit un dépliage $<< B, E, F >, \lambda >$. Ses événements peuvent toujours être décomposés en un ensemble $\overline{E}$ de processus tel que :

– $\bigcup_{E_i \in \overline{E}} E_i = E$ et,

– $\forall (E_i, E_j) \in \overline{E} \times \overline{E}$, si $E_i \neq E_j$ alors les deux processus sont en conflit ; ce qui signifie qu'il existe $(e_i, e_j) \in E_i \times E_j$ t.q. $e_i \sharp e_j$ et ${}^\bullet e_i \cup {}^\bullet e_j \subseteq B_i \cap B_j$.

Les processus de $\overline{E}$ sont ainsi qualifiés de *processus alternatifs*.

L'ensemble $\overline{E}$ sur un préfixe complet E est à rapprocher de l'ensemble des séquences maximales dont le calcul produit le graphe d'état. Dans le contexte de la sémantique d'ordre partiel, un processus alternatif E_i est caractérisé par un état final qui, soit est sans successeur (marquage mort), soit constitue la répétition d'un marquage interne d'un certain processus de E. Dans ce dernier cas, les éléments de E_i sans événement successeur dans E sont évidemment tous cut-off. Un processus alternatif est *maximal* s'il n'a pas d'événement successeur dans E.

3.2. Dépliage par processus : principe et algorithme

Dans un graphe d'état, le calcul des successeurs d'un état global ne dépend pas des événements qui y ont conduit : cet état est une convergence de toutes les séquences de tir possibles pouvant y conduire.

Par contre, dans un dépliage, même si des séquences (ou entrelacements) de processus alternatifs aboutissent à un même état global de RdP, leur représentation est distincte (sous forme de coupes). La décision de calculer les successeurs d'un seul des états globaux équivalents implique que toutes ses extensions possibles sont à ajouter à l'espace d'état, tandis que celles des autres états équivalents ne devraient pas l'être pour éviter des redondances. Les algorithmes actuels de dépliage n'énumérant pas des états globaux, le risque est que des extensions nécessaires soient ajoutées de manière dispersée à partir de divers états équivalents, forçant ainsi à calculer les dérivations de plusieurs états au lieu d'un seul. Un exemple de Esparza *et al.* (reproduit à la figure 3 en annexe A), qui a servi à introduire le concept d'ordre adéquat, a montré comment des cut-offs dispersés sur la succession de plusieurs états équivalents a occasionné l'arrêt prématuré d'un dépliage.

A la différence du concept d'ordre adéquat, la solution adoptée ici est d'éviter de générer simultanément des états globaux équivalents et incompatibles (du fait de conflit). L'idée est de ne pas produire un dépliage mêlant les productions concomitantes d'événements appartenant à des processus alternatifs. Le dépliage est obtenu en calculant les processus alternatifs dans un ordre total, ce qui permet d'éviter le scénario décrit précédemment. Dans ce contexte, la définition d'un événement cut-off est simplifiée :

Définition 4. *Un événement* cut-off e *est tel que* $\exists e' \in E, Mark(E_e) = Mark(E_{e'}) \wedge (e' \prec e \vee e' \sharp e)$.

Bien entendu, cette définition exclut la concurrence qui traduit que le processus $E_e \cup E_{e'}$ induit un état global à représenter, avec des dérivations potentielles.

En nous affranchissant du besoin d'ordre adéquat entre les événements, tel que défini dans [6], l'avantage immédiat apporté par ce nouveau principe est qu'on ne se restreint plus aux RdP saufs. L'algorithme 1 est l'implémentation proposée.

1 Créer B_0 ; $B \leftarrow B_0$;
2 $E \leftarrow \{e_0\}$; $Ext \leftarrow \emptyset$; $CurrentProcess \leftarrow \emptyset$; $\overline{E} \leftarrow \emptyset$;
3 $NewExt \leftarrow \{e \in E_F \mid {}^\bullet e \subseteq B_0\}$;
4 $Ext \leftarrow NewExt$;
5 **tant que** $Ext \neq \emptyset$ **faire**
6 **si** $\exists e \in Ext \mid \forall e' \in CurrentProcess, \neg(e \sharp e')$ **alors**
7 $CurrentProcess \leftarrow CurrentProcess \cup \{e\}$;
8 **sinon**
9 $\overline{E} \leftarrow \overline{E} \cup \{CurrentProcess\}$;
10 Choisir $e \in Ext$;
11 $CurrentProcess \leftarrow E_e$;
12 **fin**
13 $Ext \leftarrow Ext \backslash \{e\}$;
14 $E \leftarrow E \cup \{e\}$;
15 $Post\_e \leftarrow e^\bullet$; $B \leftarrow B \cup Post\_e$;
16 **si** *e n'est pas cut-off* **alors**
17 $NewExt \leftarrow \{e \in E_F \mid ({}^\bullet e \subseteq B) \wedge ({}^\bullet e \cap Post\_e \neq \emptyset)\}$;
18 $Ext \leftarrow Ext \cup NewExt$;
19 **fin**
20 **fin**

Algorithm 1. *Préfixe complet de dépliage par processus*

$NewExt$ contient les nouvelles extensions possibles dues aux post-conditions créées après la production de chaque nouvel événement e. L'état initial[1] B_0 est créé par l'événement fictif e_0. $CurrentProcess$ contient les événements du processus maximal en cours de calcul, par ajout un à un des extensions possibles et compatibles de Ext (lignes 6 et 7). Le développement du processus est arrêté lorsque plus aucun ajout d'extension compatible n'est pas possible (ligne 8). Un nouveau processus alternatif est alors chargé dans $CurrentProcess$ (lignes 8 à 11), après la production d'une des extensions de Ext (toutes incompatibles avec les processus alternatifs précédents) et en intégrant sa configuration locale. Ainsi, les différents processus alternatifs sont dépliés un à un. Le dépliage est complet quand il n'existe plus d'extension possible (Ext devient vide).

En gros, l'algorithme 1 ne diffère essentiellement des précédents que par le fait que les extensions possibles ne s'ajoutent pas suivant un ordre adéquat entre les configurations locales, mais suivant un ordre imposé par les processus alternatifs générés un à un. Avec les méthodes classiques de calcul de préfixe complet, les extensions choisies permettent au contraire de développer des processus alternatifs simultanément.

En matière de complexité, l'algorithme ne remet pas en cause l'affirmation selon laquelle le facteur dominant est le calcul des extensions possibles [6] (cf. lignes 3 et 17). En effet, les spécificités de l'algorithme sont d'une complexité qui reste d'ordre polynomial. Ainsi, le choix d'une extension (ligne 6) nécessite un test de conflit avec chacun des événements du processus courant, comparé au choix d'un élément minimal ($<$) de [6]. Et le test d'événement cut-off (ligne 16) semble globalement plus simple que celui de [6], qui peut nécessiter de comparer des configurations locales en plusieurs étapes (cf. annexe

1. On admet que m_0 est produit par l'événement fictif e_0, qui est pris en compte par les implémentations disponibles de [6] : en effet, e_0 peut constituer un événement de référence.

A). Enfin, le chargement d'un nouveau processus alternatif (la configuration locale d'une extension ajoutée) est une opération spécifique à l'algorithme, et qui a aussi un certain coût.

3.3. Résultats expérimentaux

L'annexe A illustre le principe du calcul par processus et montre sur quelques exemples de réseaux la spécificité de l'algorithme comparé aux autres algorithmes.

Des résultats de tests sur un bon nombre de RdP bornés, saufs ou non, sont présentés en annexe B. Ils sont comparés avec ceux obtenus avec l'algorithme classique (pour les RdP saufs) : les deux algorithmes ne donnent pas toujours le même résultat structurel. Nos tests ont toutefois montré que l'espace d'état (généré par un graphe d'état) est toujours couvert, y compris pour les réseaux non saufs.

3.4. Discussions

Le principe du dépliage par processus permet d'intégrer aisément le test pour identifier un réseau non borné (en assurant ainsi la terminaison de l'algorithme comme pour le calcul d'un graphe d'état) : il suffira de mémoriser les états globaux visités par chaque processus au cours du dépliage, leur nombre restant proportionnel au nombre d'événements créés. En effet, un dépliage par processus est similaire au calcul en profondeur d'un graphe d'état.

Avec les réseaux non saufs, le résultat du dépliage peut être parfois moins efficace qu'une conversion préalable en RdP sauf [6] (ce qui fait perdre l'expression de la concurrence), en raison des auto-conflits (cf. annexe B.2). Nous préconisons la suppression des répétitions de mêmes instances de transition en conflit et sous forme d'événements cut-off, afin d'avoir un résultat toujours au moins aussi efficace que celui obtenu via une conversion en réseau sauf.

L'algorithme s'appuie sur le calcul de processus alternatifs pour aboutir au dépliage complet. Nos tests ont révélé que plusieurs processus alternatifs pouvaient converger vers le même marquage final (avec au besoin leurs développements in extenso, par simulation du dépliage, pour les rendre maximaux). Dans certains cas, cela offre la possibilité de réduire le préfixe généré au cours du dépliage, par raccourcissement des processus alternatifs avec un suffixe redondant. Cela peut permettre de minimiser un préfixe complet à l'instar des travaux de [7]. Des illustrations sont données en annexe C.

4. Conclusion et perspectives

Dans cet article, nous avons proposé un nouvel algorithme de calcul du préfixe complet de dépliage des réseaux de Petri bornés, valable pour les réseaux non saufs. Il consiste à obtenir les processus alternatifs nécessaires pour composer un préfixe complet. Sans pour autant s'appuyer sur le concept d'ordre adéquat de [6], il produit un espace d'état complet comme l'attestent de nombreux exemples. Faute de place ici, les preuves de finitude et de complétude ne sont pas fournies.

Une suite immédiate des travaux consisterait à s'intéresser à l'optimisation du coût du dépliage [12], comme évoqué dans [1] avec la stratégie du calcul en profondeur.

Plus généralement, une perspective serait le développement d'algorithmes de vérification des propriétés sur les RdP basée sur les processus alternatifs maximaux. Vérifier par exemple les propriétés génériques à l'aide du dépliage devrait être moins coûteux qu'avec

un graphe d'état généré en entrelaçant les tirs de transitions concurrentes. Ceci pourra être comparé avec des travaux similaires [3]. Une autre perspective est de se servir du dépliage pour obtenir une réduction d'ordre partiel, à l'instar des travaux dans [13].

5. Bibliographie

[1] Blai Bonet, Patrik Haslum, Victor Khomenko, Sylvie Thiébaux, and Walter Vogler. Recent advances in unfolding technique. *Theoretical Computer Science*, 551 :84–101, 2014.

[2] Joost Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28(6) :575– 591, june 1991.

[3] Javier Esparza and Keijo Heljanko. Unfoldings : A Partial-Order Approach to Model Checking. *Monographs in Theoretical Computer Science. An EATCS Series.. Springer Publishing Company*, Incorporated, 1 edition, 2008.

[4] Javier Esparza, Pradeep Kanade, and Stefan Schwoon. A negative result on depth-first net unfoldings. *International Journal on Software Tools for Technology Transfer (STTT)*, 10(2) :161–166, 2008.

[5] Javier Esparza and Stefan Römer. An unfolding algorithm for synchronous products of transition systems. In *CONCUR'99 Concurrency Theory*, pages 2–20. Springer, 1999.

[6] Javier Esparza, Stefan Römer, and Walter Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design*, 20(3) :285–310, 2002.

[7] Keijo Heljanko. Minimizing finite complete prefixes. *Proceedings of the Workshop Concurrency, Specification & Programming 1999*, pages 83–95, Warsaw, Poland, September 1999. Warsaw University.

[8] Victor Khomenko and Maciej Koutny. Towards an efficient algorithm for unfolding Petri nets. In *CONCUR 2001 - Concurrency Theory*, pages 366–380. Springer, 2001.

[9] Victor Khomenko, Maciej Koutny, and Walter Vogler. Canonical prefixes of Petri net unfoldings. *Acta Informatica*, 40(2) :95–118, 2003.

[10] Kenneth L. McMillan. A technique of state space search based on unfolding. *Form. Methods Syst. Des.*, 6(1) :45–65, 1995.

[11] Tadao Murata. Petri nets : Properties, analysis and applications. In *ICATPN*, volume 77, pages 541–580. IEEE, April 1989.

[12] César Rodríguez and Stefan Schwoon. An improved construction of Petri net unfoldings. *Proc. of the French-Singaporean Workshop on Formal Methods and Applications (FSFMA'13)*, volume 31 of *OASICS*, pages 47–52. Leibniz-Zentrum für Informatik, july 2013.

[13] César Rodríguez, Marcelo Sousa, Subodh Sharma, and Daniel Kroening. Unfolding-based partial order reduction. *arXiv preprint arXiv :1507.00980*, 2015.

[14] Médésu Sogbohossou and David Delfieu. Dépliage des réseaux de Petri temporels à modèle sous-jacent non sauf. *ARIMA*, volume 14, pages 185–203, 2011.

[15] Antti Valmari. Stubborn sets for reduced state space generation. In *Proceedings of the 10th International Conference on Application and Theory of Petri Nets, 1989, Bonn, Germany ; Supplement*, pages 1–22, 1989.

[16] François Vernadat, Pierre Azéma, and François Michel. Covering step graph. In *ICATPN*, pages 516–535. Springer-Verlag, 1996.

[17] Pierre Wolper and Patrice Godefroid. Partial-order methods for temporal verification. In Eike Best, editor, *CONCUR*, volume 715 of *Lecture Notes in Computer Science*, pages 233–246. Springer, 1993.

A. Illustrations des deux différents algorithmes

Nous reprenons trois figures[2] dans la littérature (respectivement les figure 1 de [4], figure 3 de [6] et figure 4 de [1]) pour comparer la méthode standard [6] (notée ERV) qui est basée sur le concept d'ordre adéquat, avec notre algorithme proposé (noté DPP).

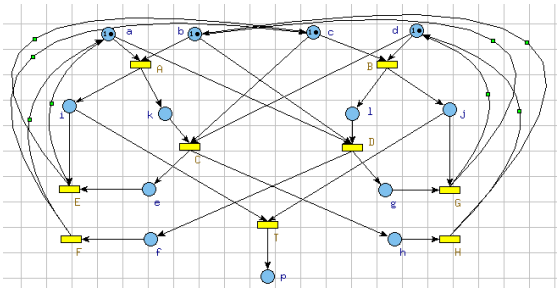

Figure 1. *Exemple 1 (fig. 1 de [4])*

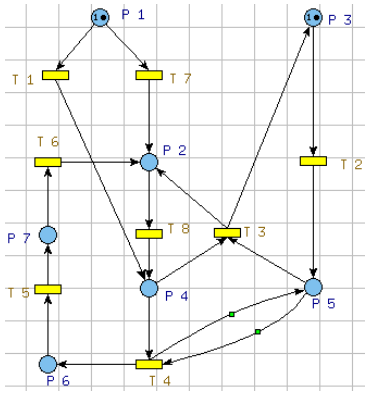

Figure 2. *Exemple 3 (fig. 4 de [1])*

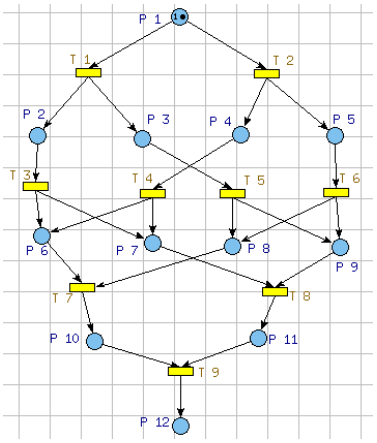

Figure 3. *Exemple 2 (fig. 3 de [6])*

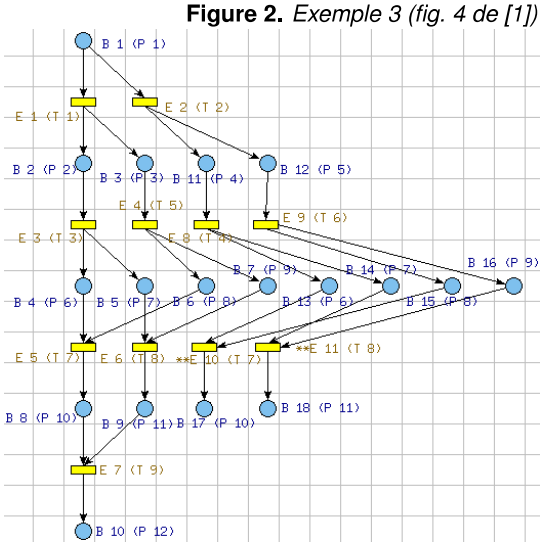

Figure 4. *Dépliage de l'exemple 2*

Il est montré que, selon les alternatives basées sur les stratégies *depth-first search* (DFS) [4] ou *breadth-first search* (BFS) [1, 6], sans associer un concept d'ordre adéquat (ou de relation bien fondée), le dépliage généré n'est pas toujours complet. C'est les cas par exemples de la figure 1 pour DFS et de la figure 3 pour BFS. Notre approche n'est complètement assimilable à aucune de ces deux stratégies.

Selon [6], une extension possible est choisie pour l'intégrer au dépliage s'il est mini-

2. Les réseaux sont édités avec le logiciel Romeo : `http://romeo.rts-software.org`

mal selon l'ordre adéquat $<$. La relation d'ordre $<$ est définie selon le critère de comparaison :

– de la taille de la configuration (locale), et en cas d'égalité,

– de l'ordre lexicographique des transitions associées du RdP N à la configuration locale, et en cas d'égalité,

– de l'ordre basé sur la forme normale de Foata[3] d'une configuration locale.

L'adéquation de la relation $<$ devra être préservée par toute extension en événement d'une configuration : en appliquant la forme normale de Foata, ceci n'est garanti que pour les réseaux saufs.

Le tableau 1 résume les résultats obtenus.

Tableau 1. *Résultats selon les deux algorithmes*

	ERV					DPP				
Petri nets	$\|E\|$	$\|B\|$	cut-offs	$\|BE\|$	$\|EB\|$	$\|E\|$	$\|B\|$	cut-offs	$\|BE\|$	$\|EB\|$
fig. 1	11	18	2	16	17	11	18	2	16	17
fig. 3	13	29	4	26	25	13	29	4	26	25
fig. 2	11	17	4	15	15	9	13	3	11	11

Pour illustration, le dépliage[4] de l'exemple 2 (fig. 4) est obtenu à partir de 2 processus : $\{e_1, e_3, e_4, e_5, e_6, e_7\}$ et $\{e_2, e_8, e_9, e_{10}, e_{11}\}$. Les événements cut-off sont e_{10} et e_{11}, en référence resp. à e_5 et e_6.

Bien que notre approche produise parfois un nombre d'événements différent de celui de [6], le dépliage produit est complet. Le principe utilisé pour tester la complétude sur le dépliage obtenu consiste à énumérer tous les marquages couverts et toutes les transitions entre ses marquages : à partir des labels des nœuds du dépliage, on traduit les éléments (marquages et transitions) de son graphe d'état en éléments du graphe d'état du RdP initial, puis on compare le résultat avec le graphe d'état obtenu directement à partir du RdP initial.

B. Présentation des résultats expérimentaux

B.1. Réseaux saufs

Les exemples de réseaux testés proviennent du logiciel de dépliage Mole[5]. Le tableau 2 permet de comparer les résultats de notre implémentation et ceux du dépliage classique.

Les résultats coïncident souvent, mais il existe des cas de dépliage pour lesquels ERV est plus favorable (gasnq3, over3-5) et d'autres pour lesquels notre algorithme est plus favorable (gasnq2, mmgt3, ring3-7).

3. Elle prend la forme d'une partition des événements de la configuration, partition obtenue en détachant itérativement l'ensemble des événements minimaux de la configuration locale [6].
4. Nos résultats (DPP) sont convertis en fichiers Romeo.
5. `http://www.lsv.ens-cachan.fr/~schwoon/tools/mole/`

Tableau 2. *Résultats du dépliage de réseaux saufs.*

	RdP				ERV			DPP		
	$\|T\|$	$\|P\|$	Marquages	Transitions	$\|E\|$	$\|B\|$	cut-offs	$\|E\|$	$\|B\|$	cut-offs
cyclic6	35	47	638	2176	50	112	7	50	112	7
cyclic9	53	71	7422	36608	77	172	10	77	172	10
cyclic12	71	95	77822	501760	104	232	13	104	232	13
dac6	34	42	640	2144	53	92	0	53	92	0
dac9	52	63	7424	35968	95	167	0	95	167	0
dac12	70	84	77824	493568	146	260	0	146	260	0
dme2	98	135	538	1036	122	487	4	122	487	4
dme3	147	202	6795	18312	321	1210	9	321	1210	9
dme4	196	269	76468	265868	652	2381	16	652	2381	16
dpfm2	5	7	4	5	5	12	2	5	12	5
dpfm5	41	27	12	31	31	67	20	31	67	20
gasnq2	85	71	192	373	169	338	46	165	330	46
gasnq3	223	143	1769	4587	1205	2409	401	1219	2437	401
gasq1	21	28	18	23	21	43	4	21	43	4
gasq2	97	78	180	357	173	346	54	173	346	54
mmgt1	58	50	72	144	58	118	20	58	118	20
mmgt2	114	86	816	2047	645	1280	260	641	1272	260
mmgt3	172	122	7702	22449	5841	11575	2529	5800	11493	2515
over2	32	33	64	133	41	83	10	41	83	10
over3	53	52	518	1563	187	369	53	286	566	83
over4	74	71	4174	16502	783	1536	237	1208	2378	410
over5	95	90	33506	163618	3697	7266	1232	5829	11490	2136
ring3	33	39	86	191	47	97	11	46	95	11
ring5	55	65	1289	4299	167	339	37	145	295	37
ring7	77	91	16999	75919	403	813	79	325	657	80

B.2. Réseaux non saufs

Nous considérons des réseaux non saufs qui modélisent un système multiprocesseur, puis un système producteur-consommateur : ils ont la particularité de contenir un grand nombre de conflits de transitions au cours de l'exécution.

Tableau 3. *Résultats du dépliage des réseaux non saufs.*

	RdP originel		DPP			
	Marquages	Transitions	$\|E\|$	$\|B\|$	cut-offs	Processus
Multiproc (5 proc, 2 bus)	45	107	91	172	67	49
Multiproc (10 proc, 5 bus)	250	845	1600	4036	1516	874
ProdCons (5 buf)	48	90	90	169	49	43
ProdCons (20 buf)	183	360	810	1429	649	613

Les résultats de dépliage (tableau 3) sont confirmés par le calcul des graphes d'état. On notera, en comparaison avec le nombre de transitions des graphes d'état, le plus grand nombre d'événements (dominés par les événements cut-off), et également le grand nombre de processus alternatifs nécessaires. Ceci est dû manifestement aux auto-conflits, i.e. le fait qu'un conflit concerne deux mêmes transitions du réseau originel avec des pré-conditions en concurrence : dans un graphe d'état, on ne considère jamais plusieurs instances de la même transition à partir d'un marquage. La solution évidente serait donc d'éliminer au cours du dépliage les auto-conflits qui sont des événements cut-off, afin que le nombre possible d'événements n'excède jamais le nombre de transitions du graphe d'état.

Les auteurs [6], dans la section 7.2 de leur article, comparent les résultats de la sémantique d'ordre partiel et de la sémantique d'exécution (i.e. par conversion préalable en réseau sauf) pour un certain nombre de réseaux non saufs. Notre implémentation donne des résultats toujours au moins aussi compacts que ceux prévus par la sémantique d'ordre partiel. Comparativement à la sémantique d'exécution, le résultat défavorable de la figure 9(c) de [6] pourra être corrigé en utilisant la solution préconisée au paragraphe précédent.

C. Détection de redondance et perspective de réduction du préfixe complet de dépliage

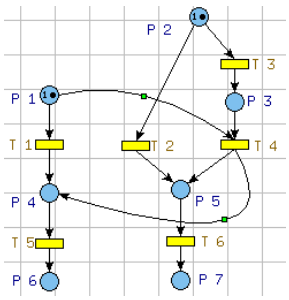

Figure 5. *Exemple 1 avec redondance*

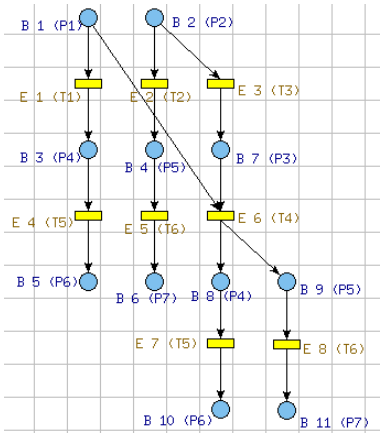

Figure 6. *Dépliage de l'exemple 1*

Le dépliage du RdP de la figure 5 est donné à la figure 6 (absence d'événement cut-off), soit la même structure que l'implémentation Mole (aux numéros des nœuds près). Il est occasionné par deux processus maximaux, $E_1 = \{e_1, e_2, e_4, e_5\}$ et $E_2 = \{e_3, e_6, e_7, e_8\}$, tombant sur le même marquage final $\{P_6, P_7\}$. Les deux processus ont ainsi un suffixe commun, les tirs des transitions T_5 et T_6, ce qui signifie des expressions de redondances dans le dépliage.

La question est de savoir si ses redondances peuvent être toujours supprimées, afin de réduire le préfixe complet. L'exemple nous suggère qu'en supprimant les descendants de b_8 et b_9 dans le processus E_2, la complétude est maintenue. Par contre, l'alternative de supprimer les descendants de b_3 et b_4 dans le processus E_1 ne conviendrait pas, parce qu'on perdrait la représentation des marquages intermédiaires P_2P_6 (b_2b_5) et P_1P_7 (b_1b_6), et des transitions qui en résultent. En somme, un préfixe réduit à $\{e_1, e_2, e_3, e_4, e_5, e_6\}$ est envisageable, mais la réduction à $\{e_1, e_2, e_5, e_6, e_7, e_8\}$ serait un dépliage incomplet.

De ce qui précède, pour qu'un des processus maximaux convergents vers le même marquage final soit réductible, l'événement (dit charnière) immédiatement en amont du suffixe commun doit être un point de convergence pour tous les événements plus antérieurs, et doit être une origine pour tous les événements constituants le suffixe commun : cela évite ainsi que des jetons (conditions) constituant un état intermédiaire soient dispersés sur le suffixe et le préfixe dont l'événement charnière constitue le point de ralliement.

Les travaux de Heljanko [7] ne traitent pas explicitement la question des redondances nécessaires, qui affectent la complétude du préfixe calculé.

L'extension de notre algorithme consistera à comparer l'état final du dernier processus maximal généré avec ceux des processus maximaux antérieurs, et en cas d'identité de marquage final, de voir si une réduction est possible sur l'un d'eux (par exemple, par une co-simulation inverse des deux processus) avant de poursuivre le dépliage.