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

Médésu Sogbohossou, Antoine Vianou

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. Des solutions sont présentées pour améliorer la compacité du préfixe obtenu.

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. Some solutions are presented to improve compactness of the prefix obtained.

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) [14] 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 [18, 19, 20] 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 [4] 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 [15, 2, 16] montrent que ces différentes techniques sont toujours en cours d'amélioration. Par exemples, l'article [15] propose un compromis entre rapidité [6] et moindre coût mémoire [10] du dépliage, et l'article [16] 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* [13, 7]. L'approche classique [7] (et ses généralisations dans [11, 2]) 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 [7] 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 [17] 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 : la méthode définie se rapproche du calcul en profondeur utilisé pour obtenir les séquences de tir composant un graphe d'état. Pour améliorer la compacité du préfixe obtenu, nous esquissons des solutions en envisageant d'une part l'élimination des auto-conflits apparaissant dans les réseaux non saufs, et d'autre part en fournissant une extension de la notion d'événement cut-off (renommé *événement de reprise* dans ce contexte) qui sert à arrêter la dérivation d'événements pour un RdP borné contenant des boucles d'exécution.

La section 2 rappelle les définitions sur les RdP et le dépliage, et propose une intégration du dépliage au calcul du graphe d'état pour mieux caractériser les concepts d'événement et de processus. Ensuite, la section 3 présente informellement le nouvel algorithme, son principe à travers de nombreuses illustrations, et les résultats de sa mise en œuvre. La section 4 se consacre à une présentation plus formelle et détaillée de l'algorithme. Puis, la section 5, en étendant le concept d'événement de reprise, permet une amélioration dans l'immédiat de la compacité d'un dépliage. Enfin, la section 6 présente la synthèse des résultats et énonce les perspectives à plus ou moins long terme.

2. Calcul de l'espace d'état des réseaux de Petri

2.1. Réseaux de Petri et graphe d'état

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 $m : P \rightarrow \mathbb{N}$: 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 $m \stackrel{t}{\rightarrow}$, si ${}^\bullet t \subseteq m$. Le *tir* de t conduisant au marquage $m' = m \backslash {}^\bullet t \cup t^\bullet$ est noté $m \stackrel{t}{\rightarrow} m'$. Soit $m_0 \stackrel{\sigma}{\rightarrow} m$ 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 : $A(N, m_0) \stackrel{\text{def}}{=} \{m \mid \exists \sigma \in T^*, m_0 \stackrel{\sigma}{\rightarrow} m\}$. $< N, m_0 >$ 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.

```
1  Entrée : réseau marqué < N, m0 >;
2  Sorties : ensembles Etats et Transitions représentant le graphe d'état;
3  Etats := {m0} ; Transitions := {};
4  Empiler l'état initial m0;
5  tant que la pile n'est pas vide faire
6      Dépiler (LIFO) un état m;
7      pour chaque transition t tirable de l'état m faire
8          Calculer l'état successeur m';
9          si m' ∉ Etats alors
10             Empiler m';
11             Etats := Etats ∪ {m'};
12         fin
13         Transitions := Transitions ∪ {(m, t, m')} ;
14     fin
15 fin
```

Algorithme 1. *Construction du graphe d'état (calcul en profondeur)*

Le calcul du graphe d'état repose sur la sémantique séquentielle : les états accessibles sont énumérés à travers les séquences de tir possibles à partir du marquage initial. La couverture de l'espace d'état est obtenue à l'aide de la procédure décrite par l'algorithme 1. La ligne 7 de l'algorithme permet d'obtenir tous les entrelacements possibles de tirs et justifie le problème de l'explosion combinatoire.

Proposition 1. *Deux principes émergent de l'algorithme 1 pour parvenir à la complétude de l'espace d'état du réseau :*

– Principe 1 *: Pour tout état accessible (en commençant par l'état initial m_0), expérimenter tous les tirs possibles pour atteindre les états successeurs correspondants.*

– Principe 2 *: Le développement de chaque exécution (ou séquence de tirs) est arrêté lorsqu'un état déjà vu (c.-à-d. mémorisé dans l'ensemble $Etats$) est à nouveau atteint, à moins qu'il n'y ait plus de franchissement possible dans un état mort.*

Pour un réseau borné, le *Principe 1* produit un ensemble d'exécutions, et le *Principe 2* assure la finitude de chaque exécution (quand une boucle infinie est détectée) et donc la finitude de l'ensemble des exécutions.

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 $\lambda_F : B_F \cup E_F \rightarrow P \cup T$ t.q. $\lambda(B_F) \subseteq P$ et $\lambda(E_F) \subseteq T$.

Définition 2. Le dépliage (exhaustif) *[17]* $Unf_F \stackrel{\text{def}}{=} < O_F, \lambda_F >$ *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 [4] et Esparza et al. [8] 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 [13, 8].

2.3. Préfixe complet de dépliage dans le contexte d'un graphe d'état

L'algorithme 2 montre mieux la relation entre un graphe d'état et le dépliage ordre partiel correspondant, et permet ici notamment de caractériser le concept d'*événement*. Il constitue juste une extension de l'algorithme 1 pour obtenir un préfixe complet pendant le calcul du graphe d'état : le préfixe est donné par les ensembles $\mathcal{B}$ et $\mathcal{E}$ (les informations concernant l'étiquetage et la relation de flux sont implicites).

```
1  Entrée : réseau marqué < N, m0 >;
2  Sorties : ensembles Etats et Transitions représentant le graphe d'état, ensembles E et B
   des nœuds étiquetés représentant le préfixe complet du dépliage sous-jacent;
3  Convertir les jetons de m0 en conditions minimales B0 ; B := B0 ; E := {};
4  Etats := {m0} ; Transitions := {};
5  Empiler l'état initial m0/B0;
6  tant que la pile n'est pas vide faire
7      Dépiler un état m/B;
8      pour chaque transition t franchissable de m faire
9          si ∄e ∈ E t.q. •e ⊆ B ∧ λF(e) = t alors
10             Produire e ∈ EF, •e ⊆ B ∧ λF(e) = t;
11             E := E ∪ {e} ; B := B ∪ e•;
12         fin
13         Calculer l'état successeur m' ; B' := B\•e ∪ e•;
14         si m' ∉ Etats alors
15             Empiler m'/B';
16             Etats := Etats ∪ {m'};
17         fin
18         Transitions := Transitions ∪ {(m, t, m')} ;
19     fin
20 fin
```

Algorithme 2. *Obtenir le préfixe complet en même temps que le graphe d'état*

Après la création des conditions minimales (ligne 3), le dépliage est obtenu par les lignes 9-13 : il existe au moins un événement pour chaque tir t et les lignes 10-11 sont exécutées seulement pour ajouter un nouvel événement dans $\mathcal{E}$ et ses post-conditions dans $\mathcal{B}$. Un tir donné t correspond à un événement avec ses pré-conditions spécifiques incluses dans l'état B. Ainsi, un événement se caractérise par le tir d'une transition t donnée en plus du contexte de sa manifestation, c'est-à-dire sa configuration locale. Un même événement peut avoir lieu à partir de différents marquages en cas de concurrence avec d'autres tirs. Par conséquent, c'est uniquement lorsque ce tir est réalisé pour la première fois durant une exécution de l'algorithme que l'événement correspondant est nouvellement généré dans $\mathcal{E}$. Pour un réseau sauf, un unique événement correspond à chaque tir t. Ainsi, un unique processus supporte chaque entrelacement de tirs, et ce processus prend en compte une multiplicité de séquences en cas de concurrence entre plusieurs tirs.

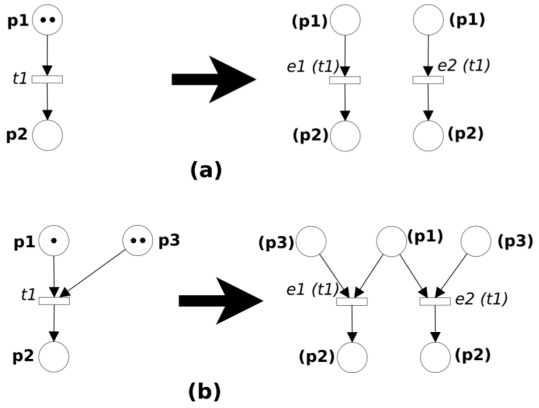

Figure 1. *Dépliage avec de multiples événements pour une même transition : multi-sensibilisation (a), auto-conflit (b)*

Pour un réseau non sauf, de multiples événements équivalents dans $\mathcal{E}_F$ pour une même transition sensibilisée t peuvent être possibles dans un état. Deux de tels événements sont mutuellement soit en concurrence (fig. 1(a)), soit en conflit (fig. 1(b)) dénommé ici *auto-conflit*. Le cas de la concurrence signifie la multisensibilisation de la transition t ; un unique processus, $\{e_1, e_2\}$, est généré pour l'exemple de la figure 1(b). Le cas de l'auto-conflit peut occasionner différents processus pour la même exécution à cause des choix multiples possibles : deux processus[1], $\{e_1\}$ et $\{e_2\}$, pour l'exemple de la figure 1(b). En pratique, il est souhaitable d'éviter le développement de plusieurs processus exprimant le même comportement, puisqu'il s'agit d'un facteur d'explosion pour générer l'espace d'état.

Les séquences de tir obtenues pendant le calcul du graphe d'état correspondent à la succession de marquages depuis m_0 se terminant lorsque m n'admet pas de franchissement (ligne 8 de l'algorithme 2), ou lorsqu'un franchissement renvoie à un état déjà mémorisé (cas $m' \in Etats$ à la ligne 14). Chacune de ces séquences constitue un entrelacement particulier d'un processus du dépliage obtenu. En ne considérant que les plus grands processus (dans le sens de la relation d'inclusion) formés à partir de ces séquences, nous définissons un ensemble des *processus alternatifs* du dépliage.

1. Mais l'algorithme 2 ne générera qu'un seul des deux événements possibles.

Evidemment, l'algorithme 2 est sans intérêt parce que l'espace d'état est complètement énuméré par entrelacements des tirs. Le concept d'événement *cut-off* basé uniquement sur les marquages propres des événements est plutôt utilisé dans une sémantique d'ordre partiel pour éviter l'écueil de l'explosion. Une conséquence de cette restriction est que le préfixe complet de dépliage peut être plus grand [13, 7], comparé à l'ensemble minimal d'événements obtenu avec l'algorithme 2.

2.4. 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 condition nécessaire pour identifier un cut-off.

L'algorithme de Esparza *et al.* [8], plus général et plus optimal que celui proposé par McMillan [13], se base sur le concept d'*ordre adéquat*. Une extension possible (événement potentiel) est choisie pour être intégrée au dépliage s'il est minimal selon l'ordre adéquat $<$. La relation d'ordre $<$ est définie selon le critère de comparaison des configurations locales des événements produits au cours du dépliage :

– 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[2] 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.

Bien entendu, le calcul du dépliage complet consiste à produire les événements possibles un à un (à partir du marquage initial) suivant l'ordre $<$ de leurs configurations locales, et une comparaison de leurs marquages propres est appliquée pour décider si l'événement le plus récemment ajouté à E est cut-off.

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. Les travaux subséquents [9, 10, 11, 15, 2] sur le dépliage reposent également sur ce concept. La prise en compte des RdP non saufs passe par une conversion en RdP sauf [7], avec pour conséquence une perte de concurrence qui peut être défavorable à la compacité du résultat.

3. Dépliage par processus : principe et tests

3.1. Esquisse de l'approche

Le nouvel algorithme proposé peut être rapproché du calcul de l'algorithme 2, dans le sens où chaque exécution est développée en profondeur, avant de passer à un processus suivant. Toutefois, ce dernier est développé sur la base du tir d'une transition sensibilisée dans un état intermédiaire mais en conflit avec un événement du processus précédent ;

2. 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 [8].

ainsi l'événement produit pour obtenir le nouveau processus ne peut être un tir concurrent (ou persistant dans un état) parce que dans ce cas, l'événement correspondant devrait appartenir à la même exécution ordre partiel. Ainsi, l'application du Principe 1 de la proposition 1 consistera à produire tous les événements visibles par un certain état dans chaque exécution, soit dans l'exécution elle-même, soit dans une nouvelle exécution en cas de conflit avec l'exécution courante.

Soit un dépliage avec son réseau d'occurrence $O \stackrel{\text{def}}{=} < B, E, F >$, et $\mathbb{P}(E)$, l'ensemble des parties E.

Définition 4. *$\overline{E} \subseteq \mathbb{P}(E)$ est un ensemble de processus tel que :*

– $E = \bigcup_{E_j \in \overline{E}} E_j$ et

– $\nexists E_k \in \overline{E}, \bigcup_{E_j \in \overline{E} \setminus \{E_k\}} E_j = E$: l'ensemble $\overline{E}$ est minimal.

Chaque élément de $\overline{E}$ est un processus alternatif *du dépliage.*

Chaque processus alternatif est l'une des exécutions ordre partiel obtenue. Il se distingue toujours des autres par l'apport d'un sous-ensemble unique d'événements dans le dépliage, ce qui implique que le nombre de processus alternatifs calculés est inférieur au nombre d'événements du dépliage E.

Définition 5. *L'ensemble $\{e \in E_F \setminus E_i \mid {}^\bullet e \subseteq B_i \setminus Max(C_i))\}$ constitue les* extensions externes *vues par un processus E_i, et seulement réalisables dans d'autres processus alternatifs : elles sont des conflits désactivés en exécutant E_i.*

Un réseau pouvant admettre des boucles infinies d'exécution, il est nécessaire de comparer des marquages pour garantir la terminaison de l'algorithme (Principe 2 de la proposition 1). Un dépliage s'appuie sur un nombre de marquages proportionnel au nombre d'événements pour éviter l'explosion. Classiquement, il s'agit des marquages propres des événements produits. Les marquages intermédaires traversés lors du développement des processus pourront être rajoutés à la base de comparaison, la taille mémoire totale nécesssaire étant ainsi moins du double du nombre d'événements du dépliage. Il peut être spécifiquement permis de rechercher un état équivalent dans le dépliage dans la mesure où le coût en temps de cette recherche reste d'ordre polynomial avec le nombre d'événements.

En considérant une exécution potentiellement infinie en développement, à quelle condition peut-on la stopper ? Plus précisément, suffit-il que le marquage admette un équivalent dans le dépliage, selon le Principe 2 ? Il est clair que si l'état final courant ($Max(B_i)$) est atteint en produisant un dernier événement sans concurrent, la réponse est positive : la suite de l'exécution sera prise en compte par un état équivalent au marquage propre de cet événement ; cela rejoint la signification classique d'un événement cut-off, dont le calcul de la succession est déjà (ou sera) pris en compte par un événement équivalent (c.-à-d. de même marquage propre).

Définition 6. *Un* événement de reprise *e est tel que $\exists e' \in E, Mark(E_e) = Mark(E_{e'}) \wedge (e' \prec e \vee e' \sharp e)$.*

Suivant la nouvelle approche, cette définition se substitue à celle d'événement cut-off. Bien entendu, elle exclut la concurrence qui traduit que le processus $E_e \cup E_{e'}$ induit un état global à représenter, avec potentiellement des dérivations qui peuvent être issues des événements e et e' à la fois.

Dans le cas où le dernier tir de E_i admet une extension concurrente sensibilisée par $Max(B_i)$, arrêter le développement du processus E_i parce qu'il admettrait un état équivalent, pourrait violer le Principe 1. Comparativement, selon la politique de l'algorithme

1, les entrelacements permettent de prendre en compte des tirs concurrents. La solution adoptée ici pour une sémantique d'ordre partiel est qu'une telle extension sensibilisée par $Max(B_i)$ soit rajoutée au processus pour l'étendre, tant qu'elle dérive directement au moins d'un événement parent qui n'est pas un événement de reprise. En effet, une extension quelconque e peut avoir plusieurs parents (c.-à-d. les éléments dans ${}^{\bullet\bullet}e$) concurrents dont un événement de reprise pourrait faire partie, ce qui traduit un contexte particulier créé par le processus qui permet la manifestation immédiate de l'événement e. Nous choisissons ici de permettre l'occurrence de e dans le processus, au lieu de se refuser la production d'une descendance pour un événement de reprise (à l'instar de la définition du cut-off dans un dépliage classique) et ainsi de ne l'envisager que par le biais d'un état équivalent à son marquage propre.

La règle appliquée pour le développement d'un processus est donc la suivante : tant qu'on peut y rajouter une extension ayant comme parent au moins un événement différent d'une reprise, le processus est étendu avec cet événement. C'est lorsqu'on ne peut plus développer le processus, même si son marquage final peut permettre encore un tir, qu'un état équivalent de l'état final est recherché dans le dépliage. Les exemples pratiques montrent que l'application de cette règle suffit pour identifier un état équivalent et extensible dans le dépliage. Ultérieurement, l'algorithme plus détaillé de la section 4 prendra en compte les cas d'échec d'identification de l'état équivalent et extensible.

Informellement, le calcul d'un préfixe complet de dépliage par processus s'effectue comme suit :

1) Le premier processus est une exécution du réseau à partir du marquage initial m_0 (B_0) ;

2) Les événements produits dans chaque processus sont collectés pour constituer l'ensemble E ;

3) Le développement de tout processus se termine soit dans un état mort, soit lorsque ses extensions ne peuvent que dériver d'événements de reprise et qu'un équivalent extensible de l'état final du processus est contenu dans le dépliage E courant ;

4) Le prochain processus alternatif est développé à partir d'une des extensions externes restantes (identifiée dans un processus passé quelconque) et non encore produites : l'exécution est démarrée à partir de la configuration locale de ce nouvel événement de E ;

5) Le dépliage est terminé lorsqu'il n'existe plus d'extension externe non encore produite.

3.2. Quelques exemples de dépliage

Quelques exemples de réseau illustrent ci-après la spécificité de l'algorithme. Quatre figures[3] sont prises dans la littérature pour comparer la méthode classique [7] (notée ERV) basée sur le concept d'ordre adéquat, avec l'algorithme proposé ici (noté DPP).

Il est montré que, selon les alternatives basées sur les stratégies *depth-first search* (DFS) [5] ou *breadth-first search* (BFS) [2, 8], 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 2 pour DFS et de la figure 3 pour BFS. L'approche proposée ici n'est complètement assimilable à aucune de ces deux stratégies.

Le tableau 1 résume les résultats obtenus.

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

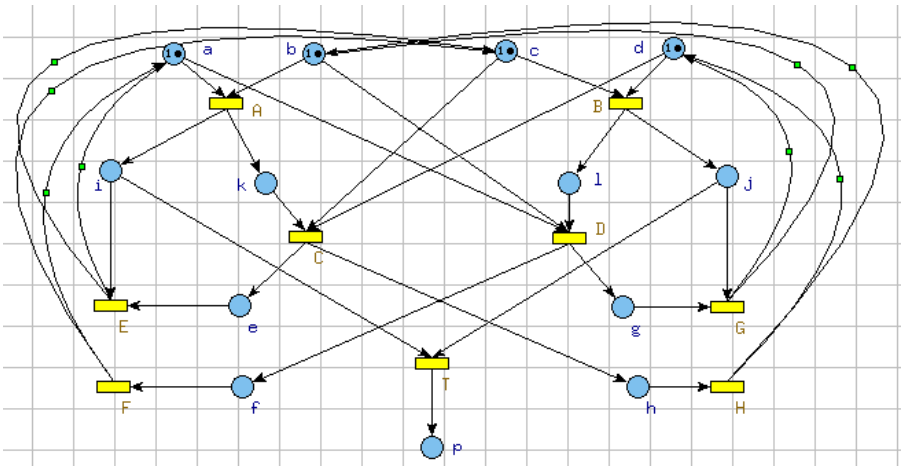

Figure 2. *Exemple 1 (fig. 1 de [5])*

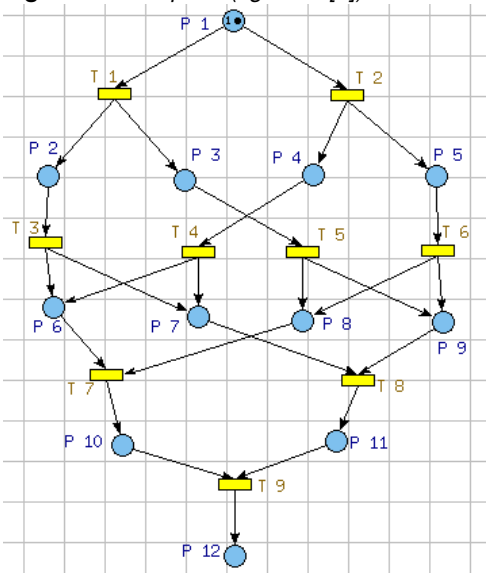

Figure 3. *Exemple 2 (fig. 3 de [7])*

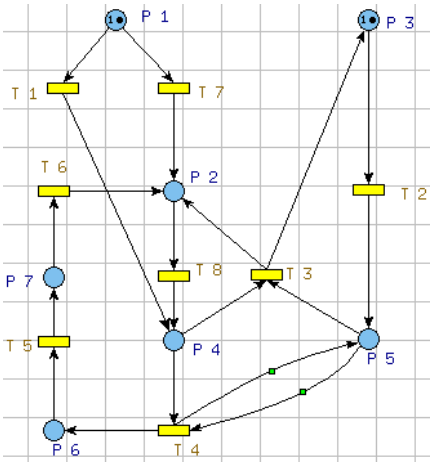

Figure 4. *Exemple 3 (fig. 4 de [2])*

Tableau 1. *Résultats selon les deux algorithmes.*

	ERV					DPP				
RdP	$\|E\|$	$\|B\|$	reprises	$\|BE\|$	$\|EB\|$	$\|E\|$	$\|B\|$	reprises	$\|BE\|$	$\|EB\|$
fig. 2	11	18	2	16	17	11	18	2	16	17
fig. 3	13	29	4	26	25	13	29	4	26	25
fig. 4	11	17	4	15	15	11	17	4	15	15
fig. 9	12	15	0	13	13	12	15	0	13	13

Le dépliage selon DPP [4] de l'exemple 1 (fig. 5) est obtenu des 3 processus : $E_1 = \{e_1, e_2, e_5\}$, $E_2 = \{e_1, e_3, e_6, e_7, e_8, e_9\}$ et $E_3 = \{e_2, e_4, e_{10}, e_{11}, e_{12}, e_{13}\}$. Les évé-

4. Les résultats selon l'implémentation de DPP sont convertis en fichier xml Romeo.

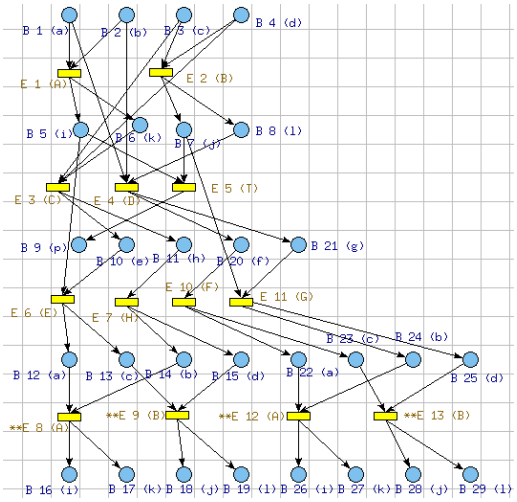

Figure 5. *Dépliage de la fig. 2*

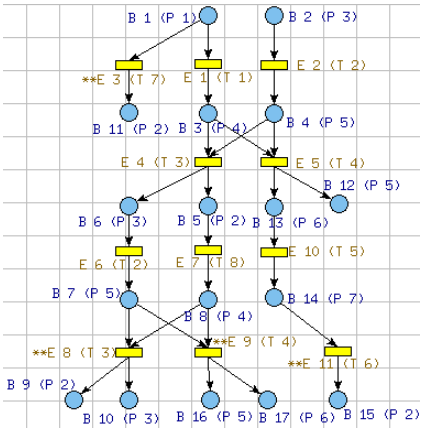

Figure 6. *Dépliage de la fig. 4*

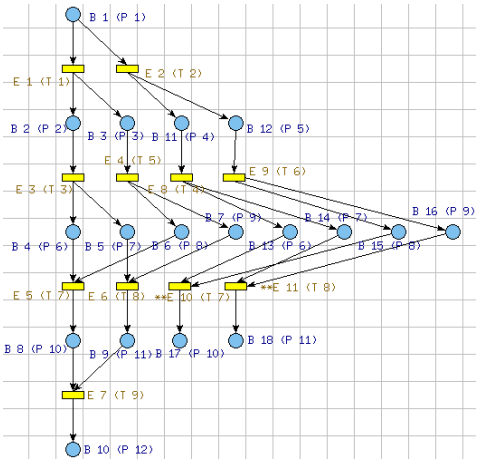

Figure 7. *Dépliage de la fig. 3*

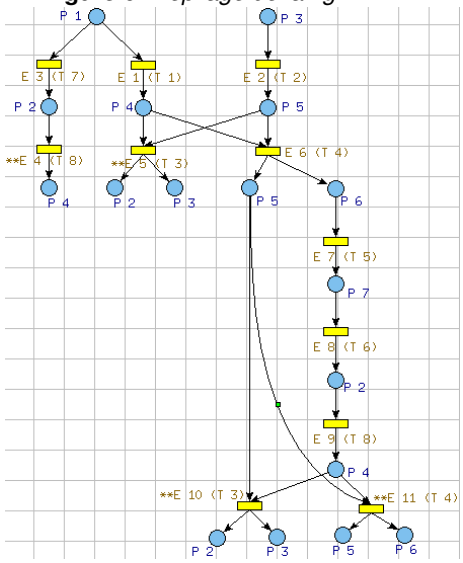

Figure 8. *Dépliage de la fig. 4 selon [7]*

nements de reprise sont e_8, e_9, e_{12} et e_{13}, en référence resp. à e_1, e_2, e_1 et e_2. L'état final E_1 est $\{b_6, b_8, b_9\}$ (marquage $\{k, l, p\}$), qui est un marquage mort. Les extensions externes sont $\{e_3, e_4\}$: le prochain processus E_2 est généré à partir de e_3. L'état final de E_2 (causé par les événements de reprise $\{e_8, e_9\}$) est $\{b_{16}, b_{17}, b_{18}, b_{19}\}$ (marquage $\{i, k, j, l\}$), qui est équivalent à l'état $\{b_5, b_6, b_7, b_8\}$ (causé par les événements concurrents $\{e_1, e_2\}$) dans E_1. L'extension externe restante est $\{e_4\}$: le dernier processus E_3 est généré à partir de e_4. L'état final de E_3 (causé par les événements de reprise $\{e_{12}, e_{13}\}$) est $\{b_{26}, b_{27}, b_{28}, b_{29}\}$ (marquage $\{i, k, j, l\}$ à nouveau, équivalent à l'état $\{b_5, b_6, b_7, b_8\}$ causé par les événements concurrents $\{e_1, e_2\}$ produits dans E_1).

Le dépliage de l'exemple 2 (fig. 7) est obtenu des 2 processus : $E_1 = \{e_1, e_3, e_4, e_5, e_6, e_7\}$ et $E_2 = \{e_2, e_8, e_9, e_{10}, e_{11}\}$. Les événements de reprise sont e_{10} et e_{11},

en référence resp. à e_5 et e_6. L'état final (et mort) de E_1 est $\{b_{10}\}$ (marquage $\{p_{10}\}$). L'unique extension externe est $\{e_2\}$: le dernier processus E_2 est généré à partir d'elle. L'état final de E_2 (causé par les événements de reprise $\{e_{10}, e_{11}\}$) est $\{b_{17}, b_{18}\}$ (marquage $\{p_{10}, p_{11}\}$), qui est équivalent à l'état $\{b_8, b_9\}$ (causé par les événements concurrents $\{e_5, e_6\}$) dans E_1.

Le dépliage de l'exemple 3 (fig. 6) est obtenu des 4 processus : $E_1 = \{e_1, e_2, e_4, e_6, e_7, e_8\}$, $E_2 = \{e_3\}$, $E_3 = \{e_1, e_2, e_5, e_{10}, e_{11}\}$ et $E_4 = \{e_1, e_2, e_4, e_6, e_7, e_9\}$. Les événements de reprise sont e_7, e_8, e_3, e_9 et e_{11}, en référence resp. à e_1, e_4, e_4, e_5 et e_6. L'état final de E_1 (causé par l'événement de reprise e_8) est $\{b_9, b_{10}\}$ (marquage propre $\{p_2, p_3\}$), qui est équivalent à l'état $\{b_5, b_6\}$ (causé par l'événement concurrent e_4) dans E_1. Il faut noter que l'événement de reprise e_7 a dû être étendu par e_8 dans E_1 (et par e_9 dans E_4) à cause de l'autre parent concurrent e_6 de e_8 (resp. e_9) qui n'est pas une reprise. Les extensions externes de E_1 sont $\{e_3, e_5, e_9\}$: le prochain processus E_2 est généré à partir de e_3. L'état final de E_2 (causé par l'événement de reprise e_3) est $\{b_2, b_{11}\}$ (marquage propre $\{p_3, p_2\}$), qui est équivalent à l'état $\{b_5, b_6\}$ (causé par l'événement e_4) dans E_1. Le processus E_3 est généré à partir de l'extension e_5. L'état final de E_3 (causé par l'événement de reprise e_{11}) est $\{b_{12}, b_{15}\}$ (marquage propre $\{p_5, p_2\}$), qui est équivalent à l'état $\{b_5, b_7\}$ (causé par l'événement e_6) dans E_1. L'extension externe restante est $\{e_9\}$: le dernier processus E_4 est généré à partir de e_9. L'état final de E_4 (causé par l'événement de reprise e_9) est $\{b_{16}, b_{17}\}$ (marquage propre $\{p_5, p_6\}$), qui est équivalent à l'état $\{b_{12}, b_{13}\}$ (causé par l'événement e_5) dans E_3.

Une comparaison entre les fig. 8 (ERV) et fig. 6 (DDP) montre que les préfixes produits sont généralement différents, du point de vue structurelle.

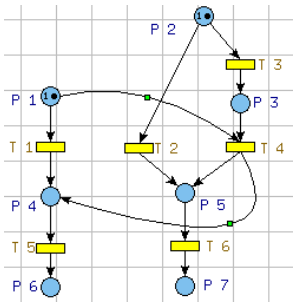

Figure 9. *Exemple 4*

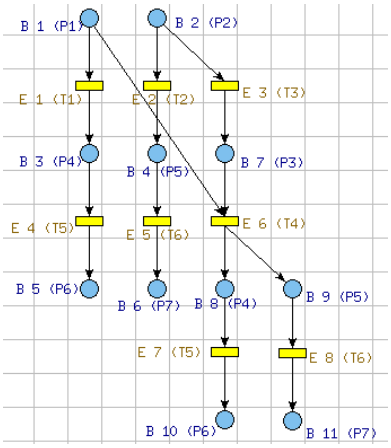

Figure 10. *Dépliage de l'exemple 4*

Le dépliage du réseau de la figure 9 est donné à la figure 10 (absence d'événement de reprise), soit la même structure que l'implémentation Mole (aux numéros des nœuds près). Il est occasionné par deux processus alternatifs, $E_1 = \{e_1, e_2, e_4, e_5\}$ et $E_2 = \{e_3, e_6, e_7, e_8\}$, tombant sur le même marquage final $\{P_6, P_7\}$. Cet exemple 4 illustre que des processus peuvent converger vers les mêmes états sans que les méthodes de dépliage ne permettent un arrêt plus tôt (donc une économie d'événements à produire) de l'un d'eux ; ceci est lié à la définition d'événement de reprise (ou de cut-off) uniquement basée sur les marquages propres.

En somme, bien que notre approche puisse produire parfois un nombre d'événements différent de celui de [8], le dépliage produit est complet. Le principe utilisé pour tester dans un premier temps 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.

3.3. Résultats expérimentaux

Des résultats de tests sur un bon nombre de RdP bornés, saufs ou non, sont présentés ci-après. 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.

3.3.1. 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.

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

	RdP				ERV			DPP		
	$\|T\|$	$\|P\|$	Marquages	Transitions	$\|E\|$	$\|B\|$	reprises	$\|E\|$	$\|B\|$	reprises
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

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).

5. `http://www.lsv.ens-cachan.fr/~schwoon/tools/mole/`

3.3.2. Réseaux non saufs

Nous considérons des réseaux non saufs des figures 11 et 12 qui modélisent resp. un système multiprocesseur et un système producteur-consommateur, et qui ont la particularité de contenir un grand nombre de conflits de transitions au cours de l'exécution.

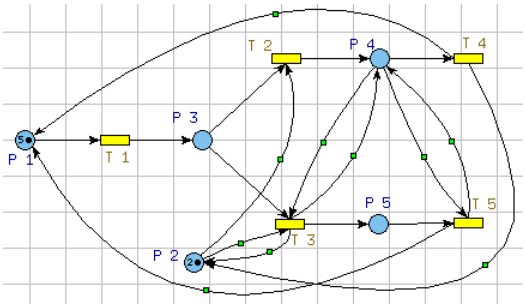

Figure 11. *Système multiprocesseur (5 processeurs, 2 bus)*

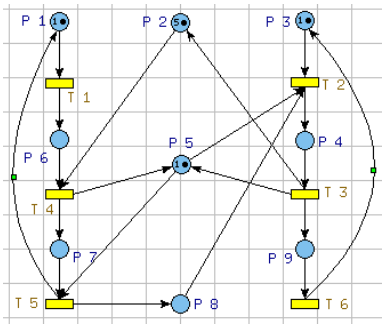

Figure 12. *Système producteur-consommateur (tampon de taille 5)*

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

	Original PN		DPP			
	Marquages	Transitions	$\|E\|$	$\|B\|$	reprises	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 de reprise), 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 préconditions en concurrence (cf. figure 1(b) de la sous-section 2.3) : 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 de reprise, afin que le nombre possible d'événements n'excède jamais le nombre de transitions du graphe d'état.

Les auteurs [7], 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 [7] pourra être corrigé en utilisant la solution préconisée au paragraphe précédent.

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'événe-

ments créés. En effet, un dépliage par processus est relativement similaire au calcul en profondeur d'un graphe d'état.

Dans la section suivante, l'algorithme détaillé présenté ne prend pas en compte les suggestions faites ici sur l'élimination des auto-conflits et le test du caractère borné.

4. Algorithme, complétude et finitude

4.1. Détail de l'algorithme

L'algorithme 3 est l'implémentation proposée.

$NewExt$ contient les nouvelles extensions possibles dues aux post-conditions créées après la production de chaque nouvel événement e (lignes 3 et 20). À la ligne 1, l'état initial[6] B_0 est créé par l'événement fictif e_0. E_i contient les événements du processus alternatif en cours de calcul, par ajout un à un des extensions possibles et compatibles de Ext (lignes 5 et 6). Le développement du processus peut être arrêté lorsque plus aucun ajout d'extension compatible n'est pas possible (ligne 7), une extension non compatible étant une extension externe.

Entrée : réseau $< N, m_0 >$
Sorties : ensemble E et B des nœuds étiquetés représentant le préfixe complet

1 $E \leftarrow \{e_0\}$; Créer B_0 à partir de m_0 ; $B \leftarrow B_0$;
2 $E_i \leftarrow \emptyset$; $\overline{E} \leftarrow \emptyset$;
3 $NewExt \leftarrow \{e \in E_F \mid {}^\bullet e \subseteq B_0\}$;$Ext \leftarrow NewExt$;
4 **tant que** $Ext \neq \emptyset$ **faire**
5 **si** $\exists e \in Ext \mid \forall e' \in E_i, \neg(e \sharp e')$ **alors**
6 $E_i \leftarrow E_i \cup \{e\}$;
7 **sinon si** `ValiderEtatFinal() =` *vrai* **alors**
8 $\overline{E} \leftarrow \overline{E} \cup \{E_i\}$;
9 Choisir $e \in Ext$;
10 $E_i \leftarrow E_e$;
11 **sinon**
12 $NewExt \leftarrow \{e \in E_F \mid {}^\bullet e \subseteq Max(C_i)\}$;
13 $Ext \leftarrow Ext \cup NewExt$;
14 Choisir $e \in NewExt$;
15 **fin**
16 $Ext \leftarrow Ext \backslash \{e\}$;
17 $E \leftarrow E \cup \{e\}$;
18 $Post\_e \leftarrow e^\bullet$; $B \leftarrow B \cup Post\_e$;
19 **si** *e n'est pas un événement de reprise* **alors**
20 $NewExt \leftarrow \{e \in E_F \mid ({}^\bullet e \subseteq B) \wedge ({}^\bullet e \cap Post\_e \neq \emptyset)\}$;
21 $Ext \leftarrow Ext \cup NewExt$;
22 **fin**
23 **fin**

Algorithme 3. *Préfixe complet de dépliage par processus*

6. On admet que m_0 est produit par l'événement fictif e_0, qui est pris en compte par les implémentations disponibles de [7] : en effet, e_0 peut constituer un événement de référence.

À l'ajout d'une extension compatible e au processus courant E_i, ce nouvel événement est évidemment extrait de l'ensemble Ext (ligne 16) et ajouté au dépliage (ligne 17). Ces post-conditions sont également créées et ajoutées au dépliage (ligne 18). Il est ensuite testé si l'événement est une reprise en conformité avec la définition 6. Seuls les événements qui ne sont pas des reprises peuvent occasionner une dérivation (lignes 20-21), en prenant toutefois en compte au besoin des post-conditions d'événements de reprise. La gestion de l'ensemble des événements de reprise n'est pas explicitée dans l'algorithme 3.

À la ligne 7, il est requis que la fonction `ValiderEtatFinal` (qui sera présentée de manière plus détaillée à la sous-section suivante) retourne $vrai$ pour arrêter effectivement un processus E_i en développement. Cette fonction recherche s'il est nécessaire de calculer les successeurs de l'état final du processus courant E_i. Pour un marquage mort, le processus E_i n'admet évidemment pas d'extension, et la fonction retourne $vrai$. Sinon, soit la fonction retourne $vrai$ si l'état final de E_i admet un équivalent (dans son propre passé ou dans un processus alternatif antérieurement calculé, ce qui signifie que la dérivation d'un tel marquage est déjà ou sera pris en compte), soit elle retourne $faux$ si un tel état n'est pas trouvé (ligne 11). Dans ce dernier cas, la dérivation des successeurs de l'état final de E_i s'avère indispensable (lignes 12-14).

Quand un processus alternatif est terminé (la fonction `ValiderEtatFinal` a retourné $vrai$), un nouveau processus alternatif est alors chargé en tant que E_i (lignes 9 et 10), après la production d'une des extensions dans Ext (qui sont toutes incompatibles avec les processus alternatifs précédents) et en intégrant sa configuration locale. Ainsi, les différents processus alternatifs sont dépliés un à un, et le dépliage est complet quand il n'existe plus d'extension possible (Ext devient vide).

En gros, l'algorithme 3 diffère essentiellement des précédents 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. De plus, l'ensemble des marquages déterminant la terminaison ne repose pas que sur les marquages propres des événements, et un événement de reprise (correspondant à un cut-off) peut faire l'objet d'une dérivation.

4.2. Recherche d'état

La recherche d'état équivalent à l'état final du processus courant est obtenue par la fonction `ValiderEtatFinal` de l'algorithme 3.

La recherche d'état ne se justifie que pour un processus qui n'est pas dans un état inextensible, comme avec les processus E_1 de l'exemple 1 (fig. 5) et E_1 de l'exemple 2 (fig. 7).

En général, une recherche d'état (test d'accessibilité) sur un réseau est de coût potentiellement exponentiel. Ici, le réseau est le dépliage courant E. Appliquer l'algorithme de calcul en profondeur implique une énumération des états, ce qui a priori n'est pas souhaitable dans notre contexte. Une recherche de couverture d'un marquage peut également reposer sur un dépliage, en ajoutant une transition spéciale au réseau et ayant comme pré-conditions les places (un multi-ensemble de places pour un marquage non sauf) constituant l'état recherché : le principe est décrit par K. L. McMillan dans [13]. Dans notre cas[7], c'est le coût du calcul des extensions (reposant sur une combinatoire des jetons) qui peut être défavorable en temps de calcul. La méthode de recherche proposée ci-après évite ces deux écueils, en garantissant un coût d'ordre polynomial.

7. Il pourrait donc s'agir de faire un autre dépliage du dépliage courant.

Une recherche d'état est directe (donc la moins coûteuse en temps de recherche) lorsque l'état final du processus correspond à un état antérieur et connu :

– un état propre d'événement,

– un état traversé et mémorisé au cours du développement des différents processus alternatifs : à chaque ajout d'extension à un processus, l'état final du processus ainsi étendu peut être mémorisé.

Une telle base de comparaison est constituée d'au plus $2\times|E|$ états. Les exemples suivants illustrent une telle recherche concluante : les processus E_2 et E_3 de l'exemple 1 (fig. 5), E_2 de l'exemple 2 (fig. 7), E_1 à E_4 de l'exemple 3 (fig. 6).

Lorsque cette phase de la recherche échoue, cela signifie que l'état équivalent recherché pourrait être intermédiaire à un entrelacement non pris en compte par l'ordre dans lequel les événements concurrents ont été produits pour le processus.

Ici, le contexte de mise en œuvre sur un dépliage permet de limiter le coût supplémentaire de cette exploration. En effet, la recherche s'appliquera à un nombre de processus (constituant $\overline{E} \cup \{E_i\}$) inférieur au nombre $|E|$ d'événements du dépliage courant, avec au plus un nombre de $|E|$ réductions d'événements par processus, soit au pire suivant un ordre inférieur à $O(|E|^2)$ itérations.

La solution adoptée est d'appliquer une réduction structurelle [1, 12] à chacun des processus, via le réseau causal correspondant. Une réduction structurelle permet de réduire la taille du réseau, tout en préservant ses propriétés pertinentes. Le principal obstacle à une réduction est la possibilité qu'une place ait plusieurs transitions d'entrée ou plusieurs transitions de sortie. Un réseau causal ne contenant que des places avec au plus une transition suivante ou et au plus une transition précédente, sa réduction peut être donc particulièrement efficace, en diminuant ainsi drastiquement l'ensemble d'accessibilité.

Ici, la réduction doit juste préserver le marquage intermédiaire recherché pour l'unique exécution ordre partiel représentée par le réseau causal. Ceci implique qu'il faille préserver a priori toutes les places marquées par un tel marquage. Soit m_f, le marquage final du processus courant E_i. L'algorithme de la réduction sur un réseau causal C_j d'un processus $E_j \in \overline{E} \cup \{E_i\}$ s'opérera comme suit.

Soit $C_{j,0} \stackrel{\text{def}}{=} C_j$, le réseau initial avant une réduction. Après l'application d'une des règles énumérées ci-après dans une phase, un réseau $C_{j,k}$ est réduit en un nouveau réseau $C_{j,k+1}$ à considérer en lieu et place de $C_{j,k}$ pour l'application de la prochaine règle. Quatre cas de figure sont possibles pour une place p (condition b) candidate à l'élimination (c.-à-d. non marquée par m_f) :

1) la place est source dans un réseau $C_{j,k}$: pour l'événement e t.q. $b \in {}^\bullet e$, on ne gardera du réseau causal C_e que les conditions $Max(C_e)$, tous les autres nœuds de C_e étant supprimés. En effet, le marquage de la place p persistera tant que l'événement e n'est pas produit ;

2) la place est puits dans un réseau $C_{j,k}$: pour l'événement e t.q. $b \in e^\bullet$, on supprimera e et tous les nœuds qui en dérive. En effet, le marquage de la place p persistera dès que l'événement e est produit ;

3) la place est intermédiaire entre deux événements dans un réseau $C_{j,k}$: soient les transitions e_1 t.q. $b \in e_1{}^\bullet$ et e_2 t.q. $b \in {}^\bullet e_2$, et soit l'ensemble $B' = \{b' \in B_{j,k} \mid b' \in e_1{}^\bullet \wedge b' \in {}^\bullet e_2\}$. Supprimer tout élément de B' (dont b fait partie), et substituer les événements e_1 et e_2 par un nouvel événement e t.q. ${}^\bullet e \stackrel{\text{def}}{=} {}^\bullet e_1 \cup {}^\bullet e_2 \backslash B'$ et $e^\bullet \stackrel{\text{def}}{=} e_2{}^\bullet \cup e_1{}^\bullet \backslash B'$. Il s'agit ainsi d'une règle de fusion sérielle (semblable à une pré-agglomération de transitions [1]) de e_1 et e_2 ;

4) la place est isolée (c.-à-d. à la fois source et puits) dans un réseau $C_{j,k}$: le marquage de cette place étant persistant, tout marquage de $C_{j,k}$ est différent de m_f. La fonction `ValiderEtatFinal` retournera alors immédiatement le résultat $faux$ s'il ne reste plus d'autres réseaux à traiter pour les processus $\overline{E} \cup \{E_i\}$.

Après une application successive des règles de réduction sur un réseau C_j, la résultante finale $C_{j,n}$ (si le dernier cas de figure n'est pas rencontré) sans davantage de réduction possible permet :

– d'identifier que $Min(C_{j,n})$ correspond à m_f ; la fonction `ValiderEtatFinal` retourne alors $vrai$;

– sinon, de passer à la réduction d'un autre réseau C_j tiré de l'ensemble $E_j \in \overline{E} \cup \{E_i\}$, selon la même procédure.

Dans le cas du dépliage d'un réseau $< N, m_0 >$ sauf, les places source $Min(C_{j,n})$ correspondent nécessairement au marquage m_f. En effet, soit le marquage initial m_{init} correspondant à $Min(C_{j,n})$ et supposons que m_f soit différent de m_{init} (donc en aval de m_{init}). La relation $m_{init} \subset m_f$ est impossible car cela traduirait un réseau non borné. De même, les relations $m_f \subset m_{init}$ ou $m_{init} \neq m_f$ sont impossibles car cela signifierait que m_{init} possèderait au moins une place non marquée par m_f. Ainsi, $m_{init} = m_f$ lorsque C_j contient m_f pour un réseau $< N, m_0 >$ sauf.

Il faut noter qu'une transition obtenue par fusion sérielle a le même marquage propre que la transition en aval (e_2). Si le marquage m_f était contenu dans $C_{j,n}$ mais distinct de m_{init} et que $C_{j,n}$ n'est pas constituée uniquement de places, c'est qu'elle contient de la concurrence entre plusieurs transitions. En effet, si toutes les transitions étaient en relation de causalité, m_f serait un marquage propre d'une des transitions, ce qui aurait été décelé avant de recourir aux réductions structurelles. La concurrence des tirs est le facteur d'explosion que nous avons choisi d'éviter dans la définition de la fonction `ValiderEtatFinal`. Une alternative serait de continuer la recherche d'état dans $C_{j,n}$ soit par dépliage ordre partiel, soit par entrelacements des tirs (en limitant éventuellement le nombre d'itérations avant un abandon de la recherche).

```
1  fonction ValiderEtatFinal()
2  début
3      si m_f ↛^t alors retourner vrai;
4      si m_f équivalent à un marquage propre d'événement ou à un marquage traversé par un
       processus alors retourner vrai;
5      pour chaque E_j ∈ Ē ∪ {E_i} faire
6          C_{j,k} ← C_j;
7          tant que ∃b ∈ B_{j,k}, m_f(λ(b)) = 0 et b n'est pas isolée faire
8              Appliquer la règle de réduction appropriée ; C_{j,k} ← C_{j,k+1};
9          fin
10         si Min(C_{j,k}) correspond à m_f et Min(C_{j,k}) ≠ Max(C_i) alors retourner vrai;
11     fin
12     retourner faux;
13 fin
```

Algorithme 4. *La fonction de recherche d'état*

Un échec de la recherche d'état dans toutes les phases décrites précédemment ne signifie pas nécessairement que cet état n'existe pas dans le dépliage actuel E, puisque

la recherche est restreinte aux processus explicitement identifiés et formant l'ensemble $\overline{E} \cup \{E_i\}$, et que nous choisissons de renoncer à l'exploration d'un réseau causal en cas de tirs concurrents qui persistent après la réduction. Un échec d'identification de l'état extensible, alors qu'il sera malgré tout déjà contenu dans le dépliage courant, impliquerait juste un résultat final de dépliage potentiellement plus volumineux qu'en opérant une recherche plus approfondie mais de nature explosive. Cette faiblesse est bien sûr inhérente à toute technique de dépliage ordre partiel, qui par essence se base sur un nombre d'états proportionnel à celui des événements constituant l'espace d'état. En particulier, cela n'empêcherait pas la finitude de l'algorithme 3 parce que le réseau déplié étant borné passe par un nombre fini d'états accessibles.

La fonction `ValiderEtatFinal` donnée par l'algorithme 4 est déduite directement des trois phases décrites précédemment : le test d'extensibilité de m_f (ligne 3), la recherche parmi les marquages propres et traversés (ligne 4), et la recherche dans les processus alternatifs (lignes 5-11). La condition $Min(C_{j,k}) \neq Max(C_i)$ de la ligne 10 permet de ne pas prendre en compte l'état final de E_i dans la recherche. La fonction retourne $faux$ quand la recherche d'état échoue pour tous les processus alternatifs (ligne 12). Les ensembles et structures manipulés par `ValiderEtatFinal` sont supposés des variables globales.

4.3. Complexité

En matière de complexité, l'algorithme 3 ne remet pas en cause l'affirmation selon laquelle le facteur dominant est le calcul des extensions possibles [8] (cf. lignes 3, 12 et 20). En effet, les spécificités de l'algorithme sont d'une complexité qui reste d'ordre polynomial. Ainsi, le choix d'une extension (ligne 5) nécessite un test de conflit avec chacun des événements du processus courant, comparé au choix d'un élément minimal ($<$) de [8]. Et le test d'événement de reprise (ligne 19) semble globalement plus simple que celui de [8], qui peut nécessiter de comparer des configurations locales en plusieurs étapes (cf. sous-section 2.4). Aussi, le chargement d'un nouveau processus alternatif (la configuration locale d'une extension externe ajoutée) à la ligne 10 est une opération spécifique à l'algorithme, de même que la recherche d'état par la fonction `ValiderEtatFinal` (algorithme 4). Cette dernière s'effectue en un temps au pire quadratique avec le nombre des événements E.

4.4. Complétude et finitude

La preuve de la complétude peut être basée sur la proposition 1. L'algorithme 3 produit un ensemble $\overline{E}$ d'exécutions ordre partiel, à l'instar de l'algorithme 1 pour un graphe d'état qui produit un ensemble de séquences de tir couvrant l'espace d'état d'un réseau borné.

Comme la sous-section 2.3 le montre, un processus prend en compte plusieurs séquences de tir en cas de concurrence. Le principe 1 est satisfait parce que chaque tir possible est vu comme une extension correspondante (systématiquement mise dans l'ensemble Ext par l'algorithme de dépliage), et lorsque la fin de l'algorithme 3 est atteinte, il ne reste plus d'extension dans Ext.

Le principe 2 est également satisfait parce que chaque exécution se termine soit dans un état mort, soit dans un état déjà représenté dans l'espace d'état (B). En effet, avec la fonction `ValiderEtatFinal`, il est garanti qu'un équivalent identifié pour l'état final de chaque processus encore extensible existe. Puisque le réseau $< N, m_0 >$ est borné,

toute exécution potentiellement infinie sera arrêtée après la production d'un nombre fini d'événements dans le processus courant.

Puisqu'un tel processus contient un nombre fini d'événements de même qu'un nombre fini d'extensions créées dans Ext pour un réseau N fini, la finitude et la complétude de l'espace d'état obtenu sont ainsi déduites des principes 1 et 2.

5. Réduction de la taille du préfixe complet

La taille d'un préfixe complet peut être plus grande que nécessaire, éventuellement dans un rapport d'ordre exponentiel pour certains modèles de RdP. Ainsi, l'algorithme de Esparza et al. [8] favorise une moindre taille comparativement à la solution originelle de McMillan [13], et les travaux de Heljanko [9] (toujours sur la base du concept d'ordre adéquat) ont visé la minimisation du préfixe complet de dépliage. L'auteur [9] ne se limite pas seulement, pour identifier un événement cut-off, à la comparaison entre son marquage propre et ceux des événements précédemment ajoutés au dépliage : il prend en compte tout marquage global contenu dans le dépliage courant, ce qui a généralement bien sûr des inconvénients de coût en temps de recherche (en général exponentiel).

La méthode de dépliage proposée ici pourrait prendre en compte la possibilité de réduire le préfixe complet pendant la mise en œuvre de l'algorithme, avec le souci d'éviter les facteurs de coût exponentiel. Outre l'élimination des auto-conflits évoquée à la sous-section 3.3.2 pour les réseaux non saufs, il est montré ici qu'un processus alternatif pourra parfois être raccourci si certains de ses états et événements sont pris en compte par un autre processus alternatif.

Le dépliage de l'exemple 4 (figure 9) donné à la figure 10 occasionne deux processus ayant un suffixe commun constitué des tirs des transitions T_5 et T_6. 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, alors que la réduction à $\{e_1, e_2, e_5, e_6, e_7, e_8\}$ serait un dépliage incomplet. Nous proposons donc une extension de la définition 6 pour prendre en compte le scénario de la figure 10.

Définition 7. *Un* événement de reprise e *est tel que* $\exists E' \subset E$*, un processus vérifiant :* $Mark(E') = Mark(E_e)$*, avec* $E_e \neq E'$.

Ce qui signifie que le marquage propre de l'événement e admet un état équivalent dans le dépliage courant. En pratique, l'état recherché est soit intermédiaire dans le processus E_e, soit situé dans un des processus alternatifs précédemment calculés.

La définition 7 permet donc de recourir à la fonction `ValiderEtatFinal` (dans ses lignes 4 à 12, et en remplaçant E_i par E_e) pour rechercher l'état équivalent au marquage propre de l'événement e. La complexité ainsi rajoutée reste d'ordre polynomial.

Pour illustrer la mise en œuvre sur l'exemple 4, les marquages propres des événements e_6, e_7 et e_8 du processus E_2 admettent comme équivalents resp. les états intermédiaires $\{b_3, b_4\}$, $\{b_4, b_5\}$ et $\{b_3, b_6\}$ du processus E_1 : par conséquent, dès la production de e_6, le développement du processus E_2 aurait été stoppé. Si le RdP avait permis le développement du processus E_2 avant E_1, les marquages propres des événements e_4 (marquage $\{P_2, P_6\}$) et e_5 (marquage $\{P_1, P_7\}$) de E_1 n'existent pas dans E_2, et n'auraient donc pas permis une réduction de E_1.

L'application de la définition 7 sur l'exemple 4 suggère que l'efficacité du résultat peut dépendre de l'ordre de production des processus alternatifs. Une perspective pour maximiser cette efficacité serait de permettre une reprise du dépliage lorsque deux processus convergent vers le même état, avec seulement la possibilité de réduire le premier des deux produits. La réduction a posteriori du premier aurait bien entendue une incidence sur les processus alternatifs postérieurs qui en dépendraient (par des événements uniques et communs qui pouraient disparaître). La contrepartie est bien sûr une augmentation du coût du dépliage.

6. Conclusion

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 [7], il produit un espace d'état complet. Aussi, il offre la possibilité d'une meilleure réduction du préfixe grâce à une extension peu coûteuse de l'algorithme.

Une première perspective serait une comparaison des résultats des variantes de l'algorithme proposé avec les autres algorithmes de dépliage [6, 10, 9] en se basant sur un jeu plus fourni de modèles de RdP, et en ressortant des critères de comparaison pertinents pour mieux justifier l'intérêt de la nouvelle approche. Une autre suite immédiate des travaux consisterait à s'intéresser à l'optimisation du coût du dépliage [15], comme évoqué dans [2] avec la stratégie du calcul en profondeur.

Plus généralement, une autre perspective serait le développement d'algorithmes de vérification des propriétés sur les RdP basée sur les processus alternatifs générés. 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 dernière perspective serait de se servir du dépliage pour obtenir une réduction d'ordre partiel.

7. Bibliographie

[1] Gérard Berthelot. Checking properties of nets using transformation. In *Advances in Petri Nets 1985, covers the 6th European Workshop on Applications and Theory in Petri Nets-selected papers*, pages 19–40, London, UK, 1986. Springer-Verlag.

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

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

[5] 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.

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

[7] Javier Esparza, Stefan Römer, and Walter Vogler. An improvement of McMillan's unfolding algorithm. In Tiziana Margaria and Bernhard Steffen, editors, *TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 87–106. Springer, 1996.

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

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

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

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

[12] K. H. Lee, J. Favrel, and P. Baptiste. Generalized Petri net reduction method. *IEEE Transactions on Systems Man and Cybernetics*, 17(2) :297–303, 1987.

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

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

[15] 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.

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

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

[18] 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.

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

[20] 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.