Algorithmique des diagrammes de séquence hiérarchiques
Un premier pas pour les Nested Message Sequence Charts (nMSC)

Blaise GENEST sous la direction de Anca Muscholl

LIAFA

Introduction

Les Message Sequence Charts (MSC) sont des spécifications, sous forme de schéma, des communications entre plusieurs processeurs. Les MSC sont importants car ils sont très utilisés, notamment pour spécifier des protocoles de communications. L'enjeu de travailler directement sur les MSC est important car ils se placent à un niveau d'implémentation très bas. Donc trouver des erreurs à ce niveau évite de nombreuses étapes inutiles d'implémentation.

On va définir dans ce rapport les nMSC (Nested MSC), qui sont une représentation hiérarchique (ou grammaticale) des MSC. C'est à dire que l'on peut définir des sous-MSC (écrit une seule fois entièrement) et les utiliser à plusieurs endroits, sans avoir besoin de les redéfinir. Deux raisons expliquent que l'on s'intéresse à ce formalisme.

- Premièrement, il est naturel d'écrire les MSC en réutilisant à plusieurs endroits les mêmes «macros» (MSC de hiérarchie plus basse).
- Deuxièmement, cela permet de « compresser » l'entrée (au mieux de manière logarithmique).

Pour que cette compression soit efficace, il faut évidemment que l'on puisse exécuter les algorithmes classiques des MSC sans avoir à déplier les nMSC, et en gardant une complexité raisonnable. Nous allons montrer que dans la plupart des cas, la complexité est équivalente aux MSC, ce qui nous amène à penser que les nMSC sont une bonne structure à utiliser.

Les nMSC sont riches car ils regroupent différents domaines très divers de l'informatique : compression, grammaire, informatique distribuée, langage de trace. Il est bon de noter que nous aurons besoin tantôt d'une approche, et tantôt d'une autre pour résoudre les différents problèmes que nous serons amener à considérer.

1 Les nMSC - Syntaxe et Sémantique

1.1 Les MSC

Les MSC décrivent des échanges de messages entre processus. Ils doivent donc décrire les envois, réceptions, et l'ordre d'arrivée de chaque message.

Les définitions que nous allons donner maintenant essaient de respecter la norme [itu96].

Définition 1 *Un MSC M est un 5-uplet $(E, <, P, l, t, m)$, contenant*
- *L'ensemble fini E des événements.*
- *Une relation d'ordre acyclique $<\subseteq E^2$.*
- *Un ensemble fini P de processus (ou d'instances).*
- *Une fonction de localisation $l : E \longrightarrow P$, qui à tout événement associe le processus où il a lieu.*
- *Une fonction de type $t : E \longrightarrow \{s, r\}$, qui à un événement associe son type (s pour un envoi, r pour une réception).*
- *Une fonction totale et bijective de message $m : t^{-1}(s) \longrightarrow t^{-1}(r)$ qui à un envoi associe une réception.*

De plus, la relation $<$ est décrite par

1. *Si (e,f) est un message, e précède f : $e < f$ si $m(e) = f$.*
2. *Si e et f sont sur le même processus, $l(e) = l(f)$, alors $e < f$ ou $f < e$.*

Nous supposerons, comme dans la définition habituelle des MSC, que chaque canal de communication de $p_1 \in P$ vers $p_2 \in P$ est *FIFO*.

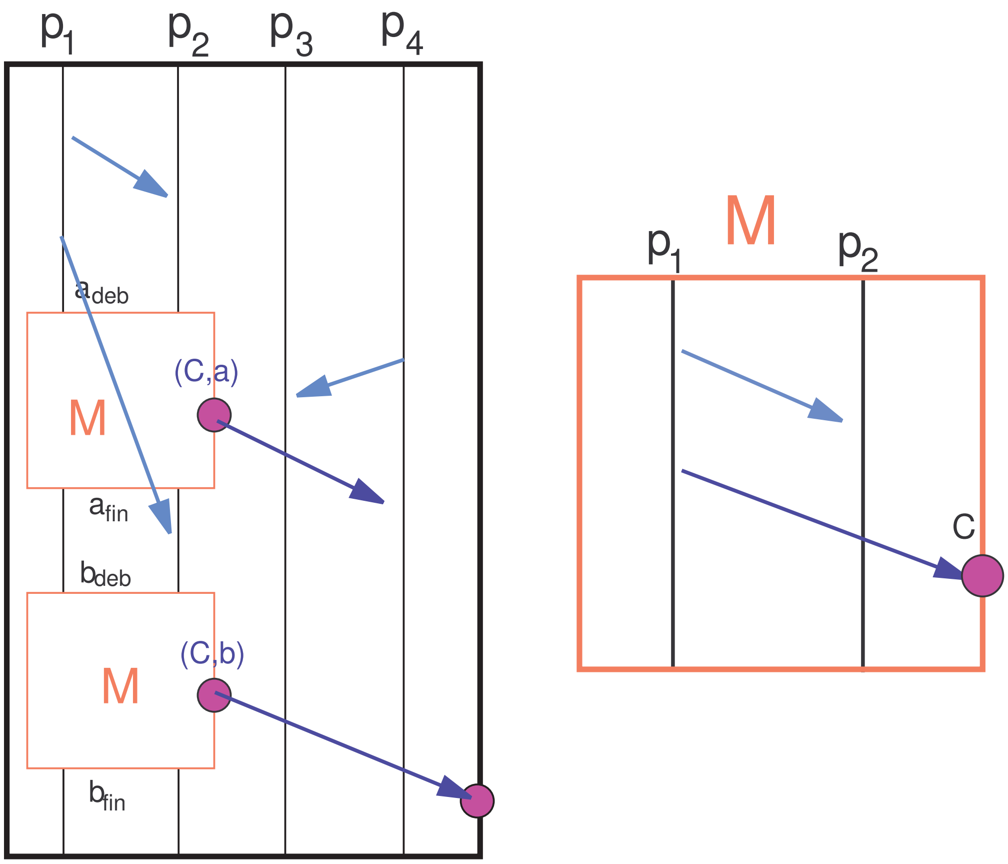
Définition 2 *La clôture transitive de $<$ est appelée ordre visuel.*

Il faut noter que cet ordre est plus restrictif que l'ordre causal, utilisé habituellement en algorithmique distribuée : l'ordre visuel est l'ordre causal où l'on empêche les réceptions d'avoir lieu plus tôt que là où elles sont spécifiées.

Définition 3 *On définit l'ordre causal $\prec$ comme la plus petite relation d'ordre contenant :*
- *l'ordre fifo*
- $\{(e, f) \mid e < f \text{ et } t(f) = s\}$.
- $\{(e, f) \mid m(e) = f\}$

Définition 4 *M est bien formé si pour tout préfixe de M, $\#send(i, j) \geq \#receive(i, j)$ pour tout i, j. Intuitivement, cela signifie qu'à toute réception correspond un envoi.*

Définition 5 *M est complet si $\#send(i, j) = \#receive(i, j)$ pour tout i, j. Intuitivement, cela signifie qu'à tout envoi correspond une réception.*



Exemple de nMSC utilisant des ports

1.2 Les nMSC

Un nMSC est un MSC où on accepte qu'un événement soit un nMSC déjà défini (on n'accepte pas les définitions croisées). De plus, nous allons toujours avoir comme objectif de définir des MSC dessinables (on peut dessiner l'émission et la réception d'un message).

Ainsi, nous nous servirons des ports, liberté apporté par [itu96], afin de passer des messages d'un niveau à l'autre de la hiérarchie.

En effet, si on perd la représentativité des MSC, leur intérêt diminue.

Définition 6 *Un nMSC* $M = (M_i)_{i=1}^n$ *est une suite finie de modules de la forme* $M_i = (E_i, P_i, B_i, \varphi_i, CI_i, CO_i, l_i, t_i, m_i, <_i)$.

Chaque module M_i *consiste en :*

- *L'ensemble fini* E_i *des événements de* M_i,
- *L'ensemble fini* P_i *des processus (ou instances) utilisés par* M_i,
- *L'ensemble fini* B_i *des boîtes utilisées par* M_i,
- *La fonction* φ_i *qui à toute boîte de* B_i *associe* $j > i$, *pour spécifier le module* M_j *que la boîte représente.*
- *L'ensemble fini* CI_i *des ports d'entrée (depuis le niveau supérieur), et l'ensemble fini* CO_i *des ports de sortie (vers le niveau supérieur). On pose* $C_i = CI_i \cup CO_i \cup \bigcup_{b \in B_i} (CO_{\varphi(b)} \cup CI_{\varphi(b)}) \times \{b\}$ *(où* $(CO_{\varphi(b)} \cup CI_{\varphi(b)}) \times \{b$ *désigne l'ensemble des ports de la sous-boîte* b*).*
- *La fonction de localisation* $l_i : E_i \longrightarrow P_i \cup C_i$, *qui à un événement associe le processeur (ou le port) où se déroule l'événement,*
- *La fonction de type* $t_i : E_i \longrightarrow \{s, r\}$, *qui à un événement associe sa fonction (envoi ou réception),*
- *La fonction de message* $m_i : t_i^{-1}(s) \longrightarrow t_i^{-1}(r)$, *bijection totale qui forme les messages complets (envoi puis réception),*
- *Un relation acyclique* $<_i$ *sur* $E_i \cup \hat{B}_i$, *où* $\hat{B}_i = \{b_{deb}, b_{fin} \mid b \in B_i\}$ *désigne les débuts et les fins des sous-boîtes de* M_i.

Les restrictions imposées à M_i *sont les suivantes :*

1. *Au plus haut niveau de hiérarchie, il n'y a pas de sur-boîte :* $CO_1 = CI_1 = \emptyset$.
2. *Les ports d'entrée ne peuvent pas recevoir de messages et les ports de sortie ne peuvent en envoyer :* $t_i(CI_i) = \{s\}$ *et* $t_i(CO_i) = \{r\}$.
3. *Pour chaque processeur* $p \in P_i$, *la relation* $<_i$ *doit être totale sur les ensembles* $l_i^{-1}(p) \cup \{b_{deb}, b_{fin} \mid b \in B_i, p \in P_b\}$.
4. *Pour chaque message* (e, f), *c'est à dire* $f = m_i(e)$,*on a* $e <_i f$.
5. *On impose que pour toute sous-boîte* $b \in B_i$, *pour tout événement* e *associé à un port d'entrée de* b *et tout événement* f *associé à un port de sortie , c'est à dire* $l_i(e) \in (CI_{\varphi(b)} \times \{b\})$ *et* $l_i(f) \in (CO_{\varphi(b)} \times \{b\})$, *que* $b_{deb} <_i e <_i f <_i b_{fin}$.
6. *Pour tout port* $c \in C_i, \#(l_i^{-1}(c)) = 1$ *: on utilise un port pour un et un seul message.*

On appelle linéarisation d'un nMSC tout ordre total, compatible avec la suite des $(<_i)$.

Remarque 1 On suivra [AEY01] en assimilant un MSC à la projection de ses évènements sur chaque processeur. On peut alors assimiler un nMSC à une grammaire algébrique (par processus) engendrant un unique mot (par processus).

Remarque 2 On emploie deux termes : sous-boîte et nMSC pour distinguer l'utilisation dans un MSC d'un sous module (la sous-boîte), et la définition de ce sous-module (le nMSC).

- Il est utile de remarquer que cette définition peut en quelque sorte contredire le caractère fifo du canal grâce à la hiérarchie et au passage de message via les ports.
- La condition 5) sur les ports d'entrée apparaissent avant les ports de sortie fait que la relation sur le MSC représenté par un nMSC est aussi acyclique.
- Les demandes sur l'ordonnancement des boîtes s'expliquent ainsi : un ordre total sur chaque processus, en considérant que le début et la fin d'une boîte travaillant sur un processus sont des événements de ce processus. Ceci ne signifie pas que l'on impose que tous les messages avant une boîte soient arrivés avant de commencer la boîte, c'est la sémantique qui réglera ces problèmes. Cependant, le dessin impose une telle propriété, qu'on retranscrit dans la définition.

Définition 7 *La taille de M, notée $|M|$, est la somme des tailles de ses événements. La taille d'un envoi ou d'une réception est 1, celle d'une sous-boîte représentant N est $Card(P_N)$ (il faut écrire dans M l'ordre de N pour les processus de P_N). On notera $p = |P|$ le nombre de processus, $c = \sum |C_i|$ celui des ports.*

1.3 Sémantique des nMSC

Pour définir la sémantique d'un nMSC, le plus simple est de la donner sous forme de MSC : Un nMSC M représente le MSC obtenu après dépliage de M. Plus formellement, on a :

$M = (M_i)$ représente $(E'_1, P'_1, l'_1, t'_1, m'_1, <'_1)$, en associant à chaque nMSC M_i le MSC $M'_i = (E'_i, P'_i, l'_i, t'_i, m'_i, <'_i)$ qu'il représente, comme décrit ci-dessous :

- Pour les événements, on a $E'_i = E_i \cup \bigcup_{b \in B_i} (E'_{\varphi_i(b)} \times \{b\}) \setminus l_i^{-1}(C_i)$ (On enlève les évènements liés au ports).
- Les processus $P'_i = P_i$ ne changent pas.
- Pour la fonction de localisation $l'_i = [l_i \cup \bigcup_{b \in B_i} l'_b]|_{E'_i}$.
- La fonction de type $t'_i = [t_i \cup \bigcup_{b \in b_i} t'_b]|_{E'_i}$ est définie de manière analogue.
- Pour la fonction de message m'_i,
  D'abord, $\forall e \in E_i, m'_i(e) = m_i(e)$.
  Pour toute sous-boîte b, pour les événements non situés sur les ports , $\forall (e,b) \in E'_i$, $m'_{\varphi_i(b)}(e) \notin CO_{\varphi_i(b)}$, on pose $m'_i((e,b)) = (m'_{\varphi_i(b)}(e), b)$: on recopie juste.
  Pour les messages associés au port c de b, on associe le seul $e = send(i,c)$ au seul $receive(c,j)$: $m'_i((e,b)) = m_i(l_i^{-1}(c))$.
- On ajoute à la relation d'ordre l'ordre dû aux messages, et l'ordre concernant un élément d'une sous-boîte et un élément de M_i situés sur le même processeur.

$$<'_i = <_i \cup \bigcup_{b \in B_i} \begin{array}{l} <_b \cup \{(e,f) \in (E'_i)^2 \mid m'_i(e) = f\} \\ \cup \{(e,(f,b)) \in (E'_i)^2 \mid l'_i(e,b) = l'_i(f), e <_i b_{deb}\} \\ \cup \{((e,b),f) \in (E'_i)^2 \mid l'_i(e,b) = l'_i(f), b_{fin} <_i f\}. \end{array}$$

2 Les nHMSC - Syntaxe et Sémantique

2.1 Les MSC-graphes

Les MSC ne permettent que de définir des spécifications finies. Pour permettre des spécifications infinies, il est naturel de considérer des graphes de MSC, qui sont appelés MSC-graphes (où encore High-Level MSC, de symbole HMSC).

Définition 8 *Un MSC-graphe G est un quadruplet (V, E, v^0, f^0), contenant*
- *Un ensemble fini V de noeuds, étiquetés par des MSC.*
- *Un ensemble $E \subseteq V^2$ d'arêtes.*
- *$v^0 \in V$ et $f^0 \in V$ les noeuds initiaux et finaux.*

2.2 Les nHMSC

On peut définir de même les nHMSC, qui sont des graphes de nMSC. On attire l'attention du lecteur sur le fait que la définition de HMSC diffère suivant les auteurs entre High-Level MSC (c'est à dire des MSC-graphes) et Hiérarchical MSC. Il nous semble préférable de bien faire la distinction, et donc de n'utiliser que l'appellation MSC-graphe dans le cas plat et **nested** High-Level MSC (nHMSC) dans le cas hiérarchique. On interdit donc l'appellation HMSC seule.

Définition 9 *Un nHMSC est une suite finie $G = (G_i)_i$, où chaque G_i est un 5-uplet $(V_i, E_i, \phi_i, v_i^0, f_i^0)$ (voir figure 1) qui consiste en*
- *L'ensemble fini V_i des noeuds.*
- *L'ensemble fini $E_i \subseteq V_i^2$ des arrêtes.*
- *v_i^0 est le noeud initial, f_i^0 est le noeud final.*
- *La fonction ϕ_i qui associe à chaque noeud soit un nHMSC $G_{\phi(v)}$, soit un nMSC M_v dont les sous-modules peuvent être des nHMSC, ie $\varphi_v(b)$ peut être un G_j.*

Remarque 3 On peut remarquer que le nHMSC de la figure 1 n'est pas représentable par un MSC-graphe.

En revanche, tout MSC-graphe est un nHMSC.

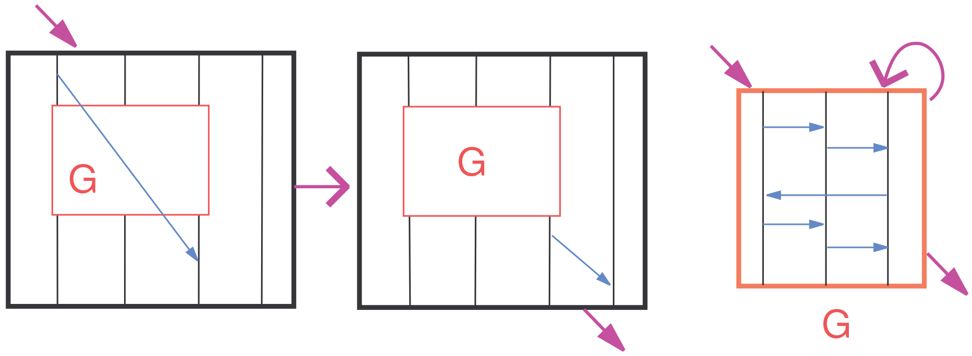


FIG.1 – Exemple de nHMSC

2.3 Sémantique des nHMSC

Il reste maintenant à définir le sens que l'on veut donner à la définition (la définition est là pour représenter au plus près le dessin, mais n'est pas suffisante pour donner un sens aux nHMSC).

Définissons la composition séquentielle faible de deux nMSC : $N_1 \circ N_2$ est définie comme $(E_1 \cup E_2, P, l_1 \cup l_2, t_1 \cup t_2, m_1 \cup m_2, <)$ avec

$<=<_1 \cup <_2 \cup \bigcup_{p \in P} l_1^{-1}(P) \times l_2^{-1}(P)$

Intuitivement, on ne demande pas que tous les processus se synchronisent avant de commencer un nouveau MSC. Un processus peut être en train d'exécuter une action d'un noeud N alors qu'un autre exécutera une action d'un successeur de N. La sémantique de G est l'ensemble des MSC donnés par la composition faible de la suite des MSC vus en suivant un chemin de G.

Définition 10 *Un chemin $(\rho_i)_i$ de G est une suite de noeuds tels que*

- *Pour tout i, $(\rho_i, \rho_{i+1}) \in E_G$,*
- *$\rho_0 = v_G^0$,*
- *Si $(\rho_i)_i$ est finie, et à $n+1$ éléments, $\rho_n = f_G^0$.*

Définition 11 *Une exécution d'un nHMSC $(G_j)_j$ est un MSC M obtenu en remplaçant, dans un chemin de G_0, chaque noeud ρ_i par une exécution de $G_{\phi(\rho_i)}$. De même, chaque boîte b associée à un nHMSC $G_{\varphi(b)}$ est remplacée par une exécution de $G_{\varphi(b)}$.*

Remarque 4 Ainsi, toute exécution d'un MSC est ce MSC.

Définition 12 $L(G) = \{ M \mid M$ *est une exécution de* $G\}$.

Définition 13 *On notera $X|_i$ la restriction de X sur le processeur i.*

Les travaux de [AEY01] font état de la clôture faible de G, sémantique sous laquelle certains problèmes sont plus faciles (voir section 9.1).

Définition 14 $L^w(G) = \{ M \mid$ *pour tout* $i, M|_i$ *est une exécution de* $G|_i$ $\}$.

3 Les problèmes à résoudre

La question est de savoir si on peut réaliser des algorithmes sans avoir à déplier les nHMSC, donc en se servant de la structure hiérarchique. Les problèmes pour lesquels il nous faut des algorithmes efficaces sont nombreux. Nous avons donc fait un choix de problèmes, les plus variés possibles.

1. Comme nous l'avons dit précédemment, il est indispensable de posséder un algorithme nous indiquant si un nMSC respecte l'ordre fifo.
2. Ensuite, le problème de l'égalité de deux nMSC (est-ce que deux nMSC représentent le même MSC ?) et de ses variantes (M est-il motif de N ?, existe il un plongement de M à N ?) sont très importants.
3. Puis vient le problème de réalisation : un ensemble fini de nMSC étant donné, existe-t-il un réseau d'automates communicants dont l'ensemble des exécutions est exactement cet ensemble de nMSC ?
4. Enfin vient le problème d'ordonnancement : les MSC imposent par définition un ordre total (l'ordre visuel) sur chaque processus. Pratiquement, on ne peut forcer cet ordre dans le sens où on ne peut pas imposer à une réception d'avoir lieu à tel ou tel moment (ordre causal). Cependant, le MSC peut imposer un ordre total sur chaque processus. C'est l'égalité de ces deux ordres que nous voulons tester.
5. Pour les nHMSC, le problème d'appartenance sert à déterminer si un scénario donné par une nMSC fait partie d'une spécification donnée par un nHMSC. Cela permet de faire du Model Checking.
6. De manière plus générale pour le Model Checking, étant donnés deux nHMSC M et N, on peut se demander si $M \cap N = \emptyset$.

4 L'ordre fifo

Les ports (et dans une moindre mesure les passages de messages par-dessus une sous-boîte) peuvent engendrer des messages qui ne respectent pas l'ordre fifo. Il est donc nécessaire d'avoir un algorithme pour vérifier que le nMSC respecte bien l'ordre fifo.

Ce que l'on a à vérifier est simplement que pour tout couple (m_1, m_2) de messages avec $s_1 < s_2$ et $r_2 < r_1$, m_1 et m_2 ne concernent pas le même couple de processeurs. Le problème est qu'il y a trop de couples de messages (un nombre exponentiel). L'idée est ici qu'on peut vérifier en temps linéaire que (m, m') ne contredit pas l'ordre fifo, où m' est un message contenu dans une sous-boîte traversée totalement par m.

Pour tout nMSC M_i, on pose $G_{M_i} = \{(p_1, p_2) \mid \text{il existe un message de } p_1 \text{ à } p_2 \text{ dans } M_i\}$.

Algorithme pour un nMSC M :

On va procéder de manière récursive, du plus bas au plus haut niveau de hiérarchie.

A chaque niveau, on va calculer, pour toutes les sous-boîtes b, le graphe de communication $G_{\phi(b)}$, ainsi que pour tout port de b,

- le processus vers lequel le message associé va, si c'est un port d'entrée.
- le processus d'où le message associé vient, si c'est un port de sortie.

On pose $G_M = \emptyset$.

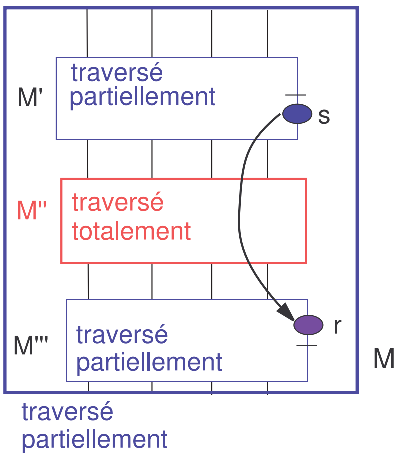
Pour tout sous-nMSC N de M :

- On lance l'algorithme sur N, on rajoute G_N à G_M.
- On parcourt les ports de N : Si l'envoi (sur p_1) et la réception (sur p_2) du message associé au port sont dans M, on rajoute (p_1, p_2) à G_M. Ce n'est pas le cas si le message va ou vient d'un port de M.

On parcourt les ports de M : On note dans le port sa provenance ou sa destination (on peut avoir besoin des informations des sous-boîtes).

Pour tout message (s, r) de p_1 vers p_2 dans M, y compris ceux dépendant des ports,

- Vérifier que les sous-boîtes b que (s, r) traverse complètement satisfont $(p_1, p_2) \notin G_b$.
- Pour les sous-boîtes b non traversées totalement (en particulier M) par (s, r),
  - Pour tout message (s', r') de b de p_1 vers p_2 , y compris ceux venant des ports,
    - vérifier que $s < s'$ et $r < r'$ ou l'inverse, sinon répondre faux.



Complexité : Pour tout M, l'algorithme nous coûte c (parcours des ports), puis une boucle de $|M|$ sur $(|N| + \sum_{b \text{ boîte}} |b|)$.

Soit au total $O(\sum_M [c + |M|(|N| + |N|)]) = O(|N|^2)$, ce qui est très bon car identique au cas sans hiérarchie.

5 Le futur d'un événement

Un MSC représente un ordre partiel sur les événements. Ainsi, nous avons souvent besoin d'engendrer le futur d'un événement. Nous allons montrer qu'on peut engendrer $\{f \mid e <_M f\}$ en temps $O((p+c)^2|M|)$, et qu'on peut le représenter en espace quadratique. De même, on peut engendrer le passé d'un événement en échangeant les émissions avec les réceptions et en lisant en sens inverse.

Pour ce faire, nous allons, pour tout processus i, produire le premier événement de i, noté m_i (notons que c'est forcément une réception), après e pour l'ordre visuel : f(processus j) est après e si et seulement si $f \geq m_j$.

On précalcule, pour tout sous nMSC M (dans l'ordre croissant de hiérarchie), un $(p+c)^2$-uplet $H(M,i,j)$. Cet ensemble représente les événements minimaux des $M|_j$ que l'on peut atteindre à partir du premier évènement du processus i. Pour tout i, $H(M,i,i) = 1$.

On va définir un ordre θ sur $E_M \cup B_M$: A chaque instant, les premiers événements à traiter sont les e_i, où $e_i \in M|_i$. Supposons être sur le processus i. Si $e_i = \mathit{send}(i,k)$, on le traite. Si $e_i = \mathit{receive}(k,i)$, on pose s l'envoi associé. On traite e_i si $s < e_k$, et on continue sur le même processus avec $e_i = e_i + 1$. Sinon, on passe au processus $(i+1 \bmod p)$ sans traiter e_i. Si e_i est une sous-boîte, on la traite si tous les processus qu'elle concerne ont exécuté toutes leurs actions précédant cette sous-boîte. Sinon, on passe au processeur $(i+1 \bmod p)$.

Il n'y a pas de blocage, sinon tous les e_i seraient des réceptions bloquées, donc $e_1 < e_2 < \cdots < e_i = e_1$, ce qui est faux ! Notons que θ est un ordre total sur $E_M \cup B_M$, compatible avec $<_M$, c'est à dire un linéarisation de M.

Pour tout $e \in E_M \cup B_M$ de M, lu dans l'ordre θ :

1. Si $e = \mathit{send}(i,k)$, associé à la réception r, on pose

$$\text{Pour tout } j \text{ tel que } H(M,j,i) < e, H(M,j,k) = \mathit{min}(H(M,j,k), r).$$

2. Si $e = M'$ un sous-MSC, on pose pour tout $i \in P_{M'}$,

$$\forall j, \text{ si } H(M,j,i) \leq e, \text{ alors } \forall k, H(M,j,k) = \mathit{min}(H(M,j,k), H(M',i,k)).$$

On traite les envois depuis les ports de M', comme des envois normaux

Ainsi, pour calculer tous les $H(M,i,j)$, on a besoin d'un temps en $O((p+c)^2|M|) = O(|M|^3)$, où p est le nombre de processus, et c le nombre de ports. Rappelons que la taille de l'événement M' est $|P_{M'}|$.

Une fois ce précalcul effectué, supposons vouloir calculer le futur de $e \in N|_i$. On lance l'algorithme avec $H(N,i,i) = e$, et $H(N,k,k) = \emptyset$. Les événements minimaux après e sont alors les $H(N,i,k)$.

démonstration

Soit M un nMSC, i,j deux processeurs. On note $e_j = \mathit{receive}(k,j)$ le premier événement de $M|_j$ après le premier événement de $M|_i$, et on pose s l'envoi associé. On peut remarquer tout d'abord que l'algorithme impose que $H(M,i,j)$ soit dans le futur du premier événement de $M|_i$: $e_j \leq H(M,i,j)$.

Procédons par récurrence sur le nombre minimum de processus (et de ports) à traverser à partir de i pour voir e_j. On veut montrer que $H(M,i,j) \leq_M e_j$ juste après avoir lu e_j. On sait que e_j est le premier événement vu, donc l'envoi s est utilisé, et le processus k est traversé.

S'il n'y a aucun processus traversé, alors $k = i$.

Si e_j se trouve sur M (et pas dans une sous-boîte), quand on lira s (situé sur le processus i), on aura $H(M,i,i) = 1 <_M s$. D'où $H(M,i,j) = min(H(M,i,j), e_j)$, donc on a bien $H(M,i,j) \leq_M e_j$.

Si e_j est un port de M, quand on lira s (situé sur le processus i), on aura $H(M,i,i) = 1 <_M s$. D'où $H(M,i,j) = min(H(M,i,j), e_j)$, donc on a bien $H(M,i,j) \leq_M e_j$.

Si e_j se trouve dans la sous-boîte M', comme s ne vient pas d'un port, il est dans M', et avant e_j : e_j est atteignable à partir du premier événement de i, donc $H(M',i,j) \leq e_j$. De plus, quand on lira cette sous-boîte M', on aura $H(M,i,i) = 1 \leq M'$, donc $H(M,i,j) = min(H(M,i,j), H(M',i,j)) \leq H(M',i,j) \leq e_j$. CQFD.

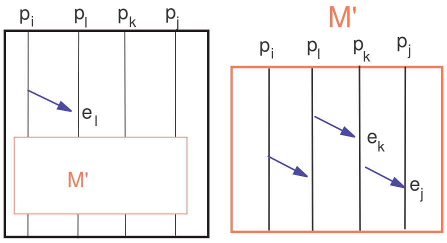
Supposons que le nombre minimal de processus (et de ports) à traverser soit $n > 0$.

Si k est un port de M, il n'est associé qu'à une unique réception, donc c'est e_k. On note s' l'envoi associé. On peut appliquer l'hypothèse de récurrence au processus k : $H(M,i,k) \leq_M e_k$, quand on lit e_k. Si k est port d'entrée de M', $H(M',k,j) \leq e_j$, donc quand on lira l'événement M', on aura $H(M,i,j) = min(H(M,i,j), H(M',k,j)) \leq_M e_j$. Si c'est un port de sortie de M', après avoir lu M', on traite l'envoi de ce port, qui mène à e_j. Donc comme $H(M,i,k) = e_k < s$, on a bien $H(M,i,j) = min(H(M,i,j), e_j) \leq e_j$.

Si e_k se trouve sur M, alors on peut appliquer l'hypothèse de récurrence au processus k : $H(M,i,k) \leq_M e_k$, quand on lit e_k. Or on a $e_k \leq_M s$, donc quand on lit e_j, on a déjà lu e_k, ie $H(M,i,k) \leq_M s$. Mais alors on applique l'algorithme qui donne $H(M,i,j) = min(H(M,i,j), r) \leq_M r = e_j$.

Sinon, e_k se trouve dans M'. Il existe un chemin de i à e_k passant par e_l, où $e_l \in M$, $e_l < M'$ et $H(M',l,k) \leq e_k$) (l peut être égale à 1 ou k : s'il est égale à k , on retrouve le cas précédent).

Supposons qu' e_j se trouve dans M', Donc $H(M',l,j) \leq e_j$). Après avoir lu M', comme $l \in P_{M'}$, et que $H(M,i,l) \leq e_l < M'$), $H(M,i,j) \leq H(M',l,j) \leq e_j$.



Si e_j n'est pas dans M', s est après M'. Quand on lira s, $H(M,i,k) \leq e_k \leq s$, donc $H(M,i,j) = min(H(M,i,j), r = e_j) \leq e_j$.

□

6 Egalité et motifs de nMSC

6.1 Egalité de nMSC

Soit M et N deux nMSC pour lesquelles on se demande si M et N représentent le même MSC. Puisque les canaux respectent l'ordre FIFO, on sait que

Proposition 1 *$M = N$ si et seulement si pour tout i, $M|_i = N|_i$, où $X|_i$ désigne la projection de X sur le i-ème processus.*

Remarque 5 Ceci n'est pas généralisable. La propriété M est facteur de N n'est pas équivalent à pour tout i, $M|_i$ est facteur de $N|_i$. Les projections peuvent être des facteurs à différents endroits de N.

Examinons tout d'abord la difficulté de cette question : un algorithme trivial engendrerait les deux MSC représentés par M et N, puis testerait l'égalité structurelle des projections de M et de N. Evidemment, cet algorithme est dans EXPTIME. Il semble alors assez difficile de tester en temps polynomial l'égalité, et pourtant c'est ce qui va être fait.

Nous prendrons ici l'approche grammaticale des nMSC : Les M_i sont des non-terminaux d'une grammaire hors contexte. Le problème que l'on se pose est le suivant :

- Entrée : A et B deux non-terminaux d'une grammaire G hors contexte, vérifiant pour tout non-terminal E, si $E \longrightarrow \alpha$ et $E \longrightarrow \beta$, alors $\alpha = \beta$. Donc tous les non-terminaux engendrent un unique mot : on peut alors à tout E associer $||E||$, la longueur du mot engendré par E.
- Sortie : oui si A et B engendrent le même mot, non sinon.

Il suffirait alors de citer le résultat de [Pla94] qui nous dit :

Théorème 1 *Tester l'égalité de deux non-terminaux d'une grammaire algébrique engendrant chacun un seul mot peut se faire en temps polynomial.*

Cependant, ce résultat est en même temps très utile et étonnant pour que nous donnions au moins une idée de la preuve.

démonstration

Soit G une grammaire et A et B deux non-terminaux engendrant un unique mot.

Calculer $||E||$ et tester si $||E|| = ||F||$ peut se faire en temps linéaire en la taille de la grammaire. On vérifie que $||A|| = ||B||$. De même, on peut calculer la taille de chaque non-terminal , et donc ranger les non-terminaux par taille décroissante, et on note $E^1 > \cdots > E^n$ la suite triée.

Pour un non-terminal A, on note $A_1 \ldots A_{||A||}$ le mot engendré par A.

On va définir deux relations sur les triplets de NT x NT x $\mathbb{N}$:

$$(A, B, i) \in \mathcal{F} \Leftrightarrow B \text{ est un préfixe de } A_i \ldots A_{||A||}$$

$$(A, B, i) \in \mathcal{S} \Leftrightarrow A_i \ldots A_{||A||} \text{ est un préfixe de } B$$

On va calculer **rel**, un ensemble dynamique de triplets, tel que à tout moment

$$A \text{ et } B \text{ engendrent le même mot } \Leftrightarrow \mathbf{rel} \subseteq \mathcal{S} \cup \mathcal{F} \text{ et } ||A|| = ||B||$$

Or A et B engendrent le même mot $\Leftrightarrow ||A|| = ||B||$ et $(A, B, 0) \in S \cup \mathcal{F}$.
Donc on pose pour commencer $\mathbf{rel} = \{(A, B, 0)\}$.
On va définir Split(**rel** ,E) par rapport à un non-terminal E tel que E n'apparaisse plus dans **rel** et que

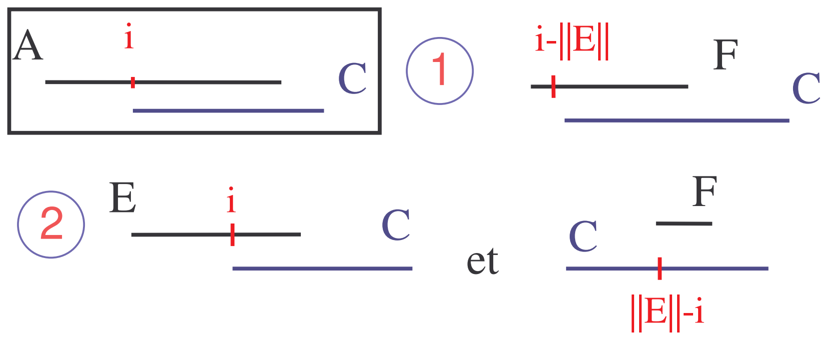
$$\mathbf{rel} \subseteq S \cup \mathcal{F} \Leftrightarrow Split(\mathbf{rel}, E) \subseteq \mathcal{S} \cup \mathcal{F}.$$

Choisissant les non-terminaux du plus grand au plus petit, après avoir effectué, pour $i = 1 \ldots N$, $\mathbf{rel} =$ Split(**rel**,E^i), il n'y aura plus de non-terminaux , et il reste seulement à tester des égalités structurelles.

Pour simplifier, on va appliquer le lemme suivant :

Lemma 1. *Quitte à changer la grammaire en une grammaire équivalente, on peut supposer qu'elle est de la forme $A \longrightarrow BC$ / a (où A, B, C sont des non-terminaux et a un terminal), c'est à dire sous forme de Chomsky.*

Split(**rel**,E) se définit formellement en rajoutant de nouveaux triplets à **rel**. Par exemple, Split((A, C, i),A) avec $A \longrightarrow EF$ et $||A|| - i < ||C||$ nous donne un des deux cas suivants :$(F, C, i - ||E||)$, où (E, C, i) et $(C, F, ||E|| - i)$.



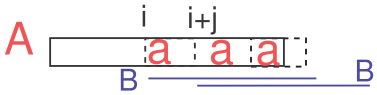
Les autres cas se définissent de manière similaire.
On trouve ainsi

$$|Split(\mathbf{rel}, E) \cap \mathcal{S}| \leq 3|\mathbf{rel} \cap \mathcal{S}|$$

$$|Split(\mathbf{rel}, E) \cap \mathcal{F}| \leq |\mathbf{rel} \cap \mathcal{F}| + 3|\mathbf{rel} \cap \mathcal{S}|$$

Donc a priori, on engendre un nombre *exponentiel* de formules atomiques à tester à la fin.

Pour régler ce problème, *Plandowski* remarque tout d'abord que si (A, B, i), $(A, B, i + j) \in \mathcal{S}$, alors $A_{i+l} = B_l = A_{i+j+l}$: j est période de $A_i \cdots A_{||A||}$.



Puis il fait appel au lemme combinatoire de [FH65] :

Lemma 2. *Si x et y sont deux périodes d'un mot w avec $x + y \leq ||w||$, alors $pgcd(x, y)$ est aussi période de w.*

Ainsi, $(A, B, i), (A, B, i + j)$ et $(A, B, i + k) \in \mathcal{S}$ avec $j + k \leq ||A|| - i$

$$\Leftrightarrow (A, B, i) \text{ et } (A, B, pgcd(j, k)) \in \mathcal{S}$$

On peut appliquer cette compression après chaque appel de la fonction Split jusqu'à ce qu'il n'existe plus de tel triplet dans **rel**.

On montre que pour tout (A, B), comme $i \leq ||A|| = 2^{|M|}$ on a au plus $(2|N|+1)$ triplets $(A, B, *)$ possibles dans $\mathbf{rel} \cap \mathcal{S}$, et donc $|\mathbf{rel} \cap \mathcal{S}| \leq |N|^2(2|N| + 1)$.

D'où $|\mathbf{rel} \cap \mathcal{F}| \leq (3|N| + 1)|N|^2(2|N| + 1)$.

Complexité : on a un algorithme en $O(|M| + |N|)^5 \log(|M| + |N|))$ pour tester l'égalité de deux nMSC.

□

Remarque 6 On peut également tester en temps polynomial si un non-terminal d'une grammaire quelconque engendre un unique mot (et donc simplifier la grammaire) : si $A \longrightarrow BC$ et $A \longrightarrow DE$ avec BC et DE engendrant le même mot, alors on peut éliminer une des règles. En effet, on applique juste l'algorithme de façon récursive grâce à la propriété BC et DE engendrent le même mot implique que tout non-terminal apparaissant dans A ou B engendre un unique mot.

On a donc un théorème plus fort

Théorème 2 *Le problème suivant est dans P :*
- *entrée : G hors contexte, A et B deux non terminaux de G.*
- *sortie : Oui si A et B engendrent chacun un unique mot, et le même.*

Remarque 7 Pour tout sous-MSC A, on note ses ports $c_{(A,1)} \cdots c_{(A,c)}$. On peut calculer en temps linéaire $S_A = \{(p_1, \cdots, p_c) \mid$ il existe une sur boîte B de A où pour tout port $c_{(A,i)}$ de A, le message associé à c est localisé sur $p_i\}$. $\sum_A Card(S_A) \leq |N|$, donc on peut faire $\sum_A Card(S_A)$ copies de la grammaire (qui sera donc au plus de taille quadratique). Ainsi, on peut supposer que tout les ports ont un unique destinataire et un unique envoyeur. On peut alors exécuter l'algorithme pour tester l'égalité.

6.2 Motifs de nMSC

L'algorithme trivial pour savoir si le mot généré par un non-terminal B est facteur du mot généré par un non-terminal A teste s'il existe $i < ||A|| - ||B||$ tq $B = A_i \cdots A_{i+||B||}$. Mais cet algorithme est exponentiel car $||A|| - ||B||$ l'est.

Ici, c'est une idée venue de [MST97] (voir aussi [Ryt99]) dont on va se servir, les arguments de base étant toujours la description sous forme grammaticale ainsi que le lemme de périodicité. Cependant, il est intéressant de remarquer que de telles préoccupations sont venues du monde d'Internet : rechercher des mots dans des grandes bases de données compressées. Cela explique que ce soit les gens s'intéressant à la compression qui aient trouvé des techniques efficaces. Ainsi, le caractère «Nested» de nos MSC s'interprète en compression dite «straight-line program».

De plus, on peut noter que l'algorithme que l'on va décrire pour les mots compressés est en temps $O(|N|^2|M|^2)$, donc, bien que plus général que l'égalité, meilleur que l'algorithme initial de [Pla94].

Concernant les nMSC, on peut toutefois noter une différence avec l'algorithme d'égalité : on a $N = M$ si et seulement si chacune des projections $N|_i = M|_i$, mais ceci est faux pour les facteurs (chacune des projections de M peut être facteur de N, mais à des positions différentes). De plus, on impose que $G_B = \{\{p,q\} \mid \text{il existe un message entre } p \text{ et } q\}$ ait une seule composante connexe.

On va ainsi raisonner en deux étapes : "Générer" en un certain sens tous les endroits où les projections de M peuvent être facteurs des projections de N, puis vérifier qu'il existe (ou non) de telles projections qui peuvent se rassembler en une MSC (c'est à dire tel que les envois de messages de p_1 à p_2 correspondent aux réceptions de messages de p_1 à p_2).

Le résultat de [MST97] étant bien expliqué et utilisant les mêmes idées que [Pla94], nous ne donnerons qu'une idée de l'algorithme et de la preuve, juste suffisante pour notre utilisation.

Théorème 3 *Tester si M est facteur de N peut se faire en temps $O(|N|^2|M|^2)$, et espace $O(|N|^2|M|)$, où N et M sont des mots compressés en straight-line-program.*

démonstration

On notera X les non-terminaux de M et Y ceux de N, ainsi que $X_1 \cdots X_{||X||}$ le mot engendré par X.

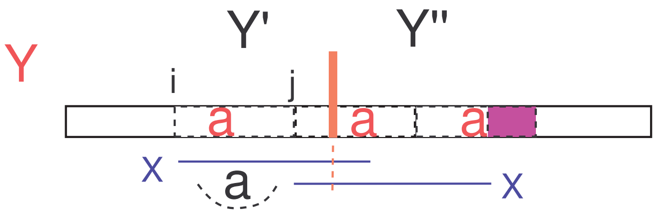
L'idée est ici de dire que si X est facteur de N, alors soit il est inclus dans un Y du plus bas niveau de hiérarchie (ce qui est facile à traiter car Y est de taille polynomiale), soit il est inclus dans un $Y \longrightarrow Y'Y''$, et à cheval entre Y' et Y'' (donc non inclus dans Y' ni Y'', ou en tout cas pas cette occurrence de X).

On notera $Occ^\star(Y,X)$ l'ensemble des indices i tels que $Y_i \cdots Y_{i+||X||-1} = X$, avec $i < ||Y'|| < i + ||X|| - 1$.

L'algorithme repose sur le fait que cet ensemble $Occ^\star(Y,X)$ est soit un singleton, soit une unique progression arithmétique : la démonstration s'appuie sur

[FH65]. Il a donc une représentation de taille $O(|N|)$ (début, fin et le pas de la suite).

En fait, si $i, j \in Occ^\star(Y, X)$, alors $Y_{i+k} = X_k = Y_{j+k} = X_{j-i+k}$. De plus, comme ces occurrences de X sont à cheval entre Y' et Y'', elles ont une intersection non vide. On a donc une période sur X : en effet $X = (a_1 \cdots a_{j-i})^d(a_1 \cdots a_l)$, avec $l < j-i$ et $d \geq 1$. De plus, $Y_i \cdots Y_{j+||X||} = (a_1 \cdots a_{j-i})^e(a_1 \cdots a_l)$, où $e > d$.



Enfin, [MST97] nous montre qu'on peut calculer en temps $O(|N||M|)$ l'ensemble $Occ^\star(Y, X)$ si on connaît déjà $Occ^\star(Y', X)$, $Occ^\star(Y'', X)$, $Occ^\star(Y, X')$ et $Occ^\star(Y, X'')$, où $X \longrightarrow X'X''$ et $Y \longrightarrow Y'Y''$.

Ainsi, l'algorithme devient évident : calculer, par programmation dynamique, un tableau de taille $|N| \times |M|$ représentant les $Occ^\star(Y, X)$. Il prend une place $O(|N|^2|M|)$.

Une fois le tableau calculé, on peut tester en temps linéaire si $M = N_i \cdots N_{i+||M||}$. $\square$

Notons qu'on peut changer en temps linéaire la grammaire de B pour que B engendre uniquement $B_1 \cdots B_{||A||}$. Si on teste ensuite si pour tout i, $A|_i$ est facteur de $B|_i$, on a

Corollaire 1 *Tester si A est préfixe de B peut se faire en temps $O(|N|^2|M|^2)$.*

Passons maintenant au cas plus difficile où on rajoute la concurrence.

Définition 15 *Pour deux non-terminaux X et Y, on pose $X \in Y$ quand X est dans un niveau hiérarchique plus bas que Y, c'est à dire X étiquette une sous-boîte de Y, ou bien X étiquette une sous-boîte de Z, avec $Z \in Y$.*

On pose de même $X \subseteq Y$ quand le mot engendré par X est facteur du mot engendré par Y.

Théorème 4 *Tester si M est facteur de N peut se faire en temps $O(|N|^2|M|^2)$, et espace $O(p|N|^2|M|)$, où M et N sont des nMSC, N est **sans port**, et M satisfait G_M a une seule composante connexe.*

démonstration

Nous indiquons au lecteur que la démonstration qui va suivre est excessivement technique, donc longue et fastidieuse, même si elle n'est pas dénuée d'intérêt (lemmes 3 et 4 par exemple).

On doit tout d'abord remarquer que l'algorithme du théorème 3 est toujours valable même si la grammaire n'est pas sous forme de Chomsky, c'est à dire $Y \longrightarrow Y'Y''$, mais par exemple $Y \longrightarrow Y'Y''Y'''Y^{(4)}$.

Cela nous permet de garder la forme initiale du nMSC : puisque N n'a pas de port, un *receive*$(1, 2)$ et le *send*$(1, 2)$ associé apparaissent littéralement sur le même non-terminal de N.

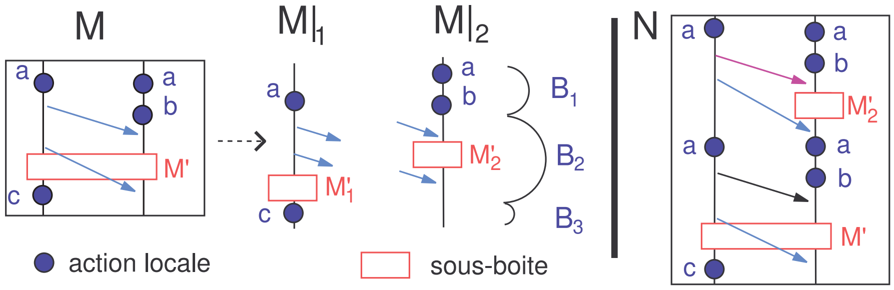
On va d'abord s'intéresser au cas où il n 'y a que *deux processus qui communiquent entre eux* , pour illustrer la preuve générale.

On note $M|_1$ et $M|_2$ la projection de M sur le premier et le second processus respectivement. On calcule les ensembles $Occ^\star(Y, X)$ pour tous les non-terminaux de $M|_1 \times M|_2$.

On va supprimer tous les indices de $Occ^\star(Y, M|_1)$ tels qu'il n'existe pas un $M|_2^0$ associé, c'est à dire une occurrence de $M|_2$ dont le premier message de 1 vers 2 correspond (dans Y) au premier message de 1 vers 2 de $M|_1$, car le canal est **fifo**.

On peut noter que pour tout indice de $Occ^\star(Y, M|_1)$, il existe au plus une occurrence de $M|_2$ dans Y associée, pas forcément à cheval entre Y' et Y''. Attention, il peut exister plusieurs occurrences de $M|_2$ associées à cette occurrence de $M|_1$, si ces occurrences de $M|_2$ apparaissent dans un non-terminal de niveau plus haut que Y.

Effectivement, le premier *send*$(1, 2)$ de l'occurrence de $M|_1$ doit être associé dans Y au premier *receive*$(1, 2)$ de l'occurrence de $M|_2$, ce qui fixe l'indice de cette occurrence de $M|_2$.



Prenons la figure ci-dessus, où $M|_1$ est suffixe de $N|_1$, et $N|_2 = M|_2 M|_2$. Cependant, pour le seul $M|_1$ possible, il faut prendre le message en noir sur N, et donc on est obligé de prendre la seconde occurrence de $M|_2$.

Pour toute occurrence M^0 de M, il existe Y tq $M^0 \subseteq Y$ et M^0 à cheval entre Y' et Y'', où $Y \longrightarrow Y'Y''$. On a soit $M^0|_1 \not\subseteq Y'$, soit $M^0|_2 \not\subseteq Y'$, sinon $M^0 \subseteq Y'$ (de même pour Y''). De plus $M^0|_1$ et $M^0|_2$ communiquent, et il n'y a *pas de port* : il existe un non-terminal $Z \subseteq Y$ qui contient un envoi entre $M^0|_1$ et $M^0|_2$ et la réception associée. Donc on ne peut pas avoir $M^0|_1 \subseteq Y'$ (resp. Y'') et $M^0|_2 \subseteq Y''$ (resp. Y''). Ceci montre que $M^0|_1$ ou $M^0|_2$ est à cheval entre Y' et Y''.

Prenons le cas où $M^0|_1$ est à cheval entre Y' et Y''. $M^0|_1 \subseteq M^0 \subseteq Y$, donc $M^0|_1$ est associé à un indice de $Occ^\star(Y, M|_1)$. Comme $M^0|_2 \subseteq M^0 \subseteq Y$, $M^0|_2$ est associé à un indice de $Occ^\star(W, M|_2)$, avec $W \in Y$, c'est à dire $M^0|_1$ sur un niveau de hiérarchie plus haut ou égal à celui de $M^0|_2$. Donc $M^0|_2$ est déterminé de manière unique par la donnée de $M^0|_1$.

On traite tout d'abord les $Occ^\star(Y, M|_i)$ *qui sont des singletons* (donc pas les progressions arithmétiques). Il y en a un nombre linéaire (au plus autant que de Y, donc au plus $|N|$).

Pour tout Y tq $Occ^\star(Y, M|_1)$ est un singleton, on détermine si le seul M_2 associé possible est présent dans les $Occ^\star(W, M|_2)$ calculés, avec $W \in Y$. Si oui, on marque $Occ^\star(Y, M|_1)$, et l'occurrence de $Occ^\star(W, M|_2)$ associée, comme à conserver (on n'efface pas encore les mauvaises occurrences). On fait de même pour les Y tq $Occ^\star(Y, M|_2)$ soit un singleton. Nous verrons après le lemme 3 que nous avons traité tous les cas possibles où des singletons étaient concernés.

Le problème des progressions arithmétiques est qu'elles représentent un nombre potentiellement exponentiel d'indices. Donc on ne peut pas regarder tous les $M|_2$ associés. Cependant, un lemme va nous tirer d'affaire, utilisant quelques nouveaux outils.

Supposons qu'il y a au moins deux messages entre 1 et 2 dans M (on verra que s'il n'y a qu'un seul message, alors il ne peut pas y avoir de progression arithmétique).

On décompose le mot engendré par $M|_1$ en $A_1A_2A_3$, où A_2 est le mot commençant par le premier message entre 1 et 2, et finissant par le dernier message entre 1 et 2. De même, on décompose le mot engendré par $M|_2$ en $B_1B_2B_3$, où B_2 commence par le premier message entre 1 et 2, et finit par le dernier message entre 1 et 2.

Lemma 3. *Soit une progression arithmétique d'au moins de* 3 *termes de* $Occ^\star(Y, M|_1)$. *Alors les* A_2 *associés aux occurrences de* $M|_1$ *de cette progression sont tous dans* $Occ^\star(Y, A_2)$ *(c'est le même* Y*), sauf peut-être le premier et le dernier.*

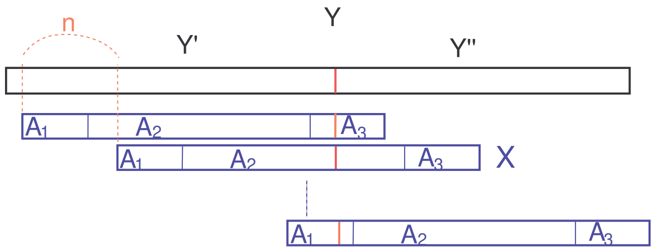
démonstration

On a déjà vu que M_1 est de la forme $(u_1 \cdots u_n)^d(u_1 \cdots u_i)$, $d \geq 2$ et $i < n$ car $Occ^\star(Y, M|_1)$ forme une progression arithmétique d'au moins trois termes.

Or il y a un message de 1 à 2, donc c'est un u_k (au passage, on voit que s'il y a une progression arithmétique, il y a au moins deux messages de 1 à 2 s'il y en a un).

Donc comme A_1 et A_3 ne contiennent aucun message de 1 à 2, on peut écrire : $A_1 = u_1 \cdots u_{k-1}$, $A_3 = u_{l+1} \cdots u_n u_1 \cdots u_i$, avec $l + 1 > i$.

Donc en particulier $|A_1| < n$ et $|A_3| < n$. Or M_1 est à cheval entre Y' et Y'', donc à moins que ce ne soit A_1 ou A_3 (cas les plus à gauche et à droite de M_1), A_2 est à cheval entre Y' et Y''.



□

Regardons pourquoi nous avons introduit A_2 et B_2 :

Le premier message de A_2^0 et son occurrence B_2^0 associée appartiennent au même non-terminal de N (car il n'a *pas de port*). De même pour le dernier message. Donc A_2^0 et une occurrence associée B_2^0 sont à cheval entre les mêmes non-terminaux Y', Y'', avec $Y \longrightarrow Y'Y''$.

On peut se débarrasser des termes extrémaux de la progression arithmétique en les considérant un à un (puisque on en a au plus $2|N| + |M|$).

Ainsi, il ne nous reste plus qu'à considérer les progressions arithmétiques de $Occ^\star(Y, M|_1)$, sans termes extrémaux.

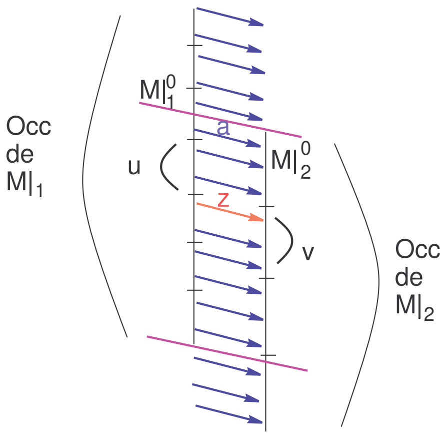
S'il y a un $M|_2^0$ associé, il peut être un singleton. Mais alors $M|_1^0$ est dans le même non terminal Y que le A_2^0 associé (d'après le lemme 3). Donc le seul B_2^0 possible associé est aussi dans $Occ^\star(Y, B_2)$. Donc le seul $M|_2^0$ possible est plus haut dans la hiérarchie. Donc ce cas a déjà été traité.

Sinon, c'est un élément d'une progression arithmétique de $Occ^\star(Z, M|_2)$.

$M|_1^0$ est dans le même non-terminal Y que A_2^0 qui est dans le même non-terminal que B_2^0 qui est dans le même non-terminal que $M|_2^0$ (d'après le lemme 3). Il faut remarquer que nous n'avons pas eu besoin de changer la grammaire, car l'algorithme ne se sert pas des A_2 et B_2. Ils nous servent juste à prouver que $M|_1^0$ et $M|_2^0$ sont sur le même niveau quand les deux appartiennent à des progressions arithmétiques.

Donc il suffit de considérer pour tout Y les ensembles $Occ^\star(Y, M|_1)$ et $Occ^\star(Y, M|_2)$ qui sont tous les deux des progressions arithmétiques (de périodes r et s).

On a alors $M_1 = u^e y$ et $M_2 = v^f z$. Supposons que u envoie m messages à 2 et que v en reçoit n de 1. On pose $ppcm(n, m) = pm = qn$ et $pgcd(n, m) = d$.



$(z_1 \equiv 1 \mod 2)$ et $(z_2 \equiv 0 \mod 3)$

On pose a le premier message de 1 vers 2 qui appartient à la fois à une occurrence de $Occ^\star(Y, M|_1)$, notée $M|_1^0$, et à une occurrence de $Occ^\star(Y, M|_2)$, notée $M|_2^0$. Il ne peut pas y avoir d'occurrence de M dans Y avant ce message a. On pose $z_1 \in [0 \cdots m-1]$ le nombre de message de 1 vers 2 entre le premier message de $M|_1^0$ et a. De même, on pose $z_2 \in [0 \cdots n-1]$ le nombre de message de 1 vers 2 entre le premier message de $M|_2^0$ et a.

On veut trouver un message de Y qui soit en même temps le premier message de 1 vers 2 d'une occurrence de $M|_1$ et le premier de $M|_2$. Ainsi, on va poser z le premier message depuis a tel qu'il existe une occurrence de $M|_1$ et de $M|_2$ commençant par ce message (en rouge sur le dessin).

On a ainsi à résoudre le système $z \equiv -z_1 \mod m$ et $z \equiv -z_2 \mod n$. En particulier, $z \equiv -z_1 \mod d \equiv -z_2 \mod d$. Si ça n'est pas le cas, il n'y a pas de solution.

Supposons donc que $z_1 \equiv z_2 \mod d$. Quitte à prendre $z = z - z_1$, on peut supposer $z_1 = 0$. Donc $z \equiv z_1 \equiv z_2 \equiv 0 \mod d$. Ainsi, quitte à tout diviser par d (puisque tout est divisible par d), on peut supposer $d = 1$.

On est donc dans les hypothèses du lemme des restes chinois :

Lemma 4. *Soit m_i deux à deux premiers entre eux, avec $n = \prod m_i$. On pose $M_i = \prod_{j \neq i} m_j$, premier avec m_i . La fonction de $[0 \ldots n-1]$ dans $\prod [0 \ldots m_i - 1]$ à qui x associe $(x \mod m_i)_i$ est bijective, et sa réciproque est $(x_i) \mapsto (\sum x_i M_i r_i) \mod n$, avec r_i l'inverse de M_i modulo m_i.*

Ainsi, il existe un unique z (modulo $ppcm(m, n)$) qui convient. De plus, il est calculable par l'algorithme d'Euclide étendu (pour obtenir l'inverse d'un nombre). On détermine donc le début de la progression arithmétique. Pour $M|_1$, la raison est rp, et pour $M|_2$, la raison est sq.

On supprime alors les occurrences qui sont marquées comme à supprimer.

Les nouveaux ensembles $Occ^\star(Y, M_j)$ sont au plus de taille $(|N|)$.

p processeurs :

On rappelle que par hypothèse, tous les processus communiquent entre eux (mais pas forcément directement).

On remplace les premiers et derniers indices de $Occ^{\star}(Y, M|_j)$ pour tout Y par des singletons. (on a au plus deux éléments et une progression arithmétique par $Occ^{\star}(Y, M|_j)$).

On va traiter le cas de tous les singletons et ensuite il ne restera plus que les occurrences appartenant aux produits des progressions arithmétiques.

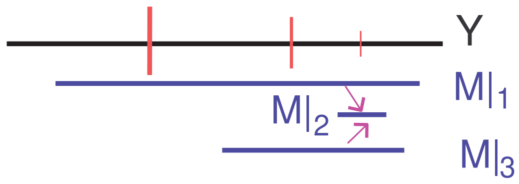
Pour toute occurrence de $(M|_1 \cdots M|_k)$ dans N, il existe un processus qui se trouve à un niveau de hiérarchie de N supérieur ou égal aux autres. En effet, supposons que l'occurrence de $(M|_1 \cdots M|_k)$ soit à cheval entre Y' et Y''. Comme tous les $M|_i$ ont une intersection non vide avec les $(M|_j)_{j \neq i}$, il existe un $M|_i$ a cheval entre Y' et Y''. Ce $M|_i$ se trouve donc à un niveau supérieur ou égal à tous les autres. Ca ne serait pas le cas si il y avait un processus qui ne communiquait avec les autres.

Une fois ce contexte fixé, il va exister un seul p-uplet d'occurrences possibles :

Pour tout les indices de $Occ^{\star}(Y, M|_j)$, l'occurrence de $(M|_1 \cdots M|_k)$ possible, avec j le processus sur le plus haut niveau de hiérarchie, va être déterminée de la façon suivante :

Comme sont déjà fixées les occurrences de tous les processus avec lesquels j communique, sont fixés aussi tous les processus avec lesquels ces processus communiquent... Donc toutes les occurrences des projections sont fixées. On s'assure qu'aucun choix n'est contradictoire entre deux processus (sinon cette occurrence de $M|_j$ ne peut pas être prolongée en une occurrence de $(M|_1 \cdots M|_k)$ (avec j sur le plus haut niveau)).

Il semble y avoir un problème : 1 peut communiquer juste avec 2, et 2 juste avec 3. De plus, 3 peut être sur un niveau plus haut de hiérarchie que 2, et alors l'occurrence n'est plus a priori unique. Le message entre 2 et 3 est néanmoins unique dans le contexte Y (car 1 est au plus haut niveau de la hiérarchie).



Ainsi, cette phase nous demande au plus $O(p^2)$ calculs par non-terminal et processus au plus haut niveau. Donc $O(p^3|N|)$ calculs.

Ainsi, il ne reste plus que le cas où toutes les projections d'une occurrence de M sont dans des progressions arithmétiques. Alors, elles sont toutes sur le même niveau (comme dans le cas avec 2 processus).

On suppose qu'on a $M|_i = u_i^{e_i} z_i$.

On pose $p_{i,j}$ le nombre d'envois de i à j (ou de réceptions) dans u_i.

Il suffit ensuite de calculer $m = ppcm(p_{i,j}) = p_{i,j}q_{i,j}$. Cela nous demande un temps $k\log(k)N^2$ (les nombres sont inférieurs à $2^{|N|}$, sinon ça ne sert à rien de calculer : il ne peut pas y avoir deux occurrences dans N ! !). De même on pose $d = pgcd(p_{i,j})$

Ensuite, il ne reste plus qu'à déterminer la première occurrence possible : Comme dans le cas à deux processus, on peut appliquer le lemme chinois. Donc on sait calculer le premier X à convenir en temps $|N|^3$ (temps pour calculer l'inverse d'un nombre $\mod 2^{|N|}$).

On va conserver ces informations sous forme de p tableaux, chacun représentant l'ensemble des occurrences avec j le processus au niveau le plus haut. Pour chaque non-terminal , on se souvient de la position de Y, et de toutes les positions (indices dans Y) des autres non-terminaux plus petits que Y (cela peut éventuellement être une p-uplet de progressions arithmétiques).

Enfin, on n'a pas besoin de plus de calculs que la première phase de l'algorithme sur les mots (en $O(|N|^4)$), donc la complexité ne change pas.

□

6.3 Plongement d'un nMSC dans un autre

Nous suivrons le papier de [MPS98] pour cette partie, qui traite le cas sans hiérarchie.

Définition 16 *Soit M, N deux MSC, et h une injection de E_M dans E_N. h est un plongement de M à N si et seulement si pour tout $e, f \in E_M$,*

- *Si e est un envoi (resp. une réception) de i à j, alors $h(e)$ aussi*
- *Si $m_M(e) = f$, alors $m_N(h(e)) = h(f)$. C'est à dire que si f est la réception associée à e, alors $h(f)$ est la réception associée à $h(e)$.*
- *Si $e \prec_M f$ (ordre causal), et sont sur le même processus, alors $h(e) <_N h(f)$ (ordre visuel).*

Ainsi, les auteurs en déduisent un algorithme en temps linéaire pour savoir si il existe un plongement h de M dans N :

Plongement(M,N)

Si $M = \emptyset$ renvoyer h.

Si $N = \emptyset$ renvoyer *faux*.

Calculer les envois de M minimaux pour $\prec_M$, dont on notera A l'ensemble. Pour tout $e = send(i, j) \in A$, calculer le plus petit événement $\mu(e) \in N$ tel que $\mu(e) = send(i, j)$. On pose $B = \{\mu(e) \mid e \in A\}$.

Calculer le minimum pour $<_N$ de B, $\mu(e^0)$. Calculer f^0 (resp. g^0) la réception associée à e^0 (resp. $\mu(e^0)$).

Ecrire $h(e^0) = \mu(e^0)$ et $h(f^0) = g^0$.

Lancer Plongement($M\backslash\{e^0, f^0\}$,$N\backslash\{g \mid g <_N g^0\}$).

Fin

Evidemment, on peut appliquer le même algorithme pour le cas où M et N sont des nMSC. Mais alors M et N étant de taille exponentielle, on a besoin d'un espace polynomial pour se souvenir de l'endroit où on est. On ne retient pas la fonction h, sinon cela prendrait un espace exponentiel, mais on va juste dire si cette fonction h existe. Les taches de base sont un calcul de minima et un calcul du passé d'un événement : nous savons faire ces taches en temps polynomial, donc en espace polynomial.

Théorème 5 *Tester si M se plonge dans N est PSPACE, pour M et N des nMSC.*

Cependant, nous ne connaissons pas exactement la classe de complexité dans lequel ce problème se trouve. D'ailleurs, même pour des mots (sans concurrence) compressés en «straight-line-program», le problème reste ouvert. Nous conjecturons néanmoins que notre problème est NP-complet (nous avons essayé de réduire le problème du sac à dos à ce problème, sans succès).

Nous pouvons tout de même essayer d'avoir des algorithmes meilleurs dans le cas où l'un ou l'autre de M et N ne serait pas sous forme hiérarchique.

Si N n'est pas hiérarchique, on calcule $|M|$et $|N|$. Si $|M| > |N|$,alors M ne peut pas être plongé dans N. Sinon, on peut déplier M puisqu'il a une taille polynomiale. On lance enfin l'algorithme sur les MSC sans hiérarchie sur M et N.

Si M n'est pas hiérarchique, on peut précalculer pour tout X non-terminal de N, pour tout (i, j), si X contient un $send(i, j)$. On peut alors appliquer le même algorithme récursif que dans le cas plat, en se servant des précalculs pour trouver les $\mu(e)$ en temps linéaire (on n'explore une sous-boîte que si elle contient un tel $\mu(e)$). De plus, le calcul des minima et du passé d'un évènement se fait en temps $O((p + c)^2|N|)$ (cf. section Le futur d'un événement).

Ainsi l'algorithme est en temps $O(((p + c)^2|N||M|)$.

7 La Réalisation Standard (safely realization)

Le but est de savoir si on peut implémenter sans blocage un ensemble de (n)MSC par des automates communicants (familles d'automates détérministes (A_i), pour chaque instance i, qui communiquent par des canaux) .

Définition 17 *Soit $\mathcal{M}$ un ensemble de (n)MSC. Le langage de $\mathcal{M}$, noté $L(\mathcal{M}) = \bigcup_{M \in \mathcal{M}} L(M)$.*

Définition 18 *Un automate communicant est sans blocage si toutes ses configurations accessibles sont coaccessibles.*

Définition 19 *$\mathcal{M}$ est S-réalisable s'il existe un famille (A_i) d'automates communicants sur $\{send(i,j); receiv(i,j) \mid i,j \in P\}$ telle que $L(\prod A_i) = L(\mathcal{M})$, et (A_i) sans blocage.*

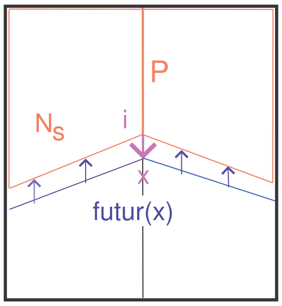
Nous suivrons le travail de [AEY00] pour cette partie :

Théorème 6 *Soit $M^1, \ldots, M^n$ des nMSC. $\{H^1 \cdots H^n\}$ est S-réalisable ssi $\forall w$ bien formé tq $\forall i, \exists l_i$ tq $w|_i$ est préfixe de $M^{l_i}|_i$, alors $\exists k$ tq w est préfixe de M^k.*

Si une telle condition est vérifiée, A_i accepte précisement l'union sur k des projections des H^k sur i, pour tout i.

Suivant [AEY00], on trouve une autre caractérisation d'un ensemble de MSC non S-réalisable, exploitable d'un point de vue algorithmique.

Condition (P) : Soit $F = \{M_1 \cdots M_n\}$ un ensemble de MSC. F satisfait (P) s'il existe un processus i et $s \neq t$ tq si N_s, N_t sont les préfixes maximaux de M_s et M_t qui satisfont $N_s|_i = N_t|_i$ et que x' est l'événement de i qui suit N_t, et tq $N_s.x'$ est bien formé et n'est préfixe d'aucun MSC M_r.



Théorème 7 *$F = \{M_1 \cdots M_n\}$ est S-réalisable ssi F satisfait (P).*

démonstration Si un tel triplet (M_s, M_t, i) existe, on prend $w = N_s.x'$, dont les projections sont préfixes de M_s (sauf pour la projection sur i qui est préfixe de M_t). Or w n'est pas préfixe d'un H^k : $\{M_1 \cdots M_n\}$ n'est pas S-réalisable.

Réciproquement, supposons que $\{M_1 \cdots M_n\}$ ne soit pas S-réalisable : il existe w bien formé tel que $\forall i, \exists l_i$ tq $w|_i$ est préfixe de $H^{l_i}|_i$. On va montrer qu'il existe un triplet (M_s, M_t, i) satisfaisant (P) par récurrence sur la taille de w.

Si $||w|| = 2$, on pose b un événement maximal de $w.w = a.b$. On remarque que w concerne exactement deux processus, i et j, avec b sur i. Soit s, t tels que $w|_i \leq M_s|_i$ et $w|_j \leq M_t|_j$. Le préfixe commun de M_s et M_t sur i est vide sinon w serait préfixe de M_t. Ainsi, $N_s|_i = N_t|_i = \emptyset$. Donc $x = b$. Comme b est maximal, N_t contient au moins a, donc $N_t.b$ contient w. Par conséquent, il est bien formé. Il n'est pas préfixe d'un M_r sinon w le serait. CQFD

Supposons $||w|| = n + 1$:w est de la forme $w = w'.b$, et w' est bien formé. Si w' est un contre-exemple qui démontre $(M_1 \cdots M_n)$ n'est pas S-réalisable, on applique la propriété de récurrence à $||w'|| = n$ et on a notre triplet (M_s, M_t, i).

Sinon, w' est préfixe d'un M_t. On note i le processus sur lequel se déroule b. w concerne au moins deux processus : on a déjà i, on notera un autre j. On a $w|_j = w'|_j$ est un préfixe de M_t, $w|_i = w'|_i b$ est préfixe d'un M_s mais pas de M_t. Donc $N_s|_i = w'|_i$, et $x = b$. Comme b est maximal, N_t contient w' : $N_t.b$ contient w et est bien formé. Si $N_t.b$ était préfixe d'un M_r, alors w serait aussi préfixe de M_r, ce qui n'est pas. D'où la démonstration par récurrence.

$\square$

On en déduit un algorithme rapide : on cherche si un tel triplet est présent.

L'algorithme :

On précalcule pour tout processus et tout nMSC M le dernier évènement sur le processus i dans M (pour ne pas avoir à le refaire un nombre exponentiel de fois) : on regarde au plus $|N|$ fois ce qu'il y a dans chaque boîte.

Pour tout (M_s, M_t, i) :

Générer le plus long préfixe commun de M_s et M_t sur i, noté P (on fait de la recherche dichotomique).

Posant x (resp. x') l'événement arrivant sur i (resp. j) après P dans M_s (resp. M_t), on calcule les événements minimaux après x : les événements juste avant (on se sert des précalculs) ces minimaux nous donnent le N_s de la condition (P).

Vérifier que $N_s.x'$ est bien formé.

Vérifier que $N_s.x'$ est préfixe d'un des M_r.

Complexité :

Pour un processus i, on note $|M_s|_i$ la taille de la projection sur i de M_s ($\sum |M_s|_i = |M_s|$), et $\sum_s |M_s| = |M|$, taille de l'entrée.

Les préfixes étant de taille au plus $2^{|M|_i}$, la recherche dichotomique prend un temps $|M|_i$. De plus, à chaque pas de la recherche on exécute l'algorithme pour le préfixe de nMSC qui prend un temps $O(|M_s|_i^2 |M_t|_i^2)$ (voir corollaire 1). C'est à dire qu'à la fin de l'algorithme, cette étape aura pris $O(\sum_t \sum_s \sum_i (|M_s|_i^2 |M_t|_i^2 |M|_i) = O(|M|^5)$ au total.

On utilise l'algorithme de calcul du futur de x, qui prend un temps $O((p + c)^2 |M_s|)$. Vérifier que $N_s.x'$ est bien formé se fait en temps linéaire (il suffit de compter le nombre d'envoi/de réception).

Vérifier que $N_s.x'$ est préfixe d'un des M_r se fait en temps $\sum_r |M_s|^2 |M_r|^2 = O(|M_s|^2 |M|^2)$. Donc au total $\sum_i \sum_t \sum_s |M_s|_i^2 |M|_i^2 = O(p|M|^5)$.

Ainsi, nous disposons d'un algorithme en $O(p|M|^5)$ pour décider si l'ensemble considéré est S-réalisable ou non.

8 Le problème de concurrence

On suppose ici qu'on ne peut pas forcer (dans l'ordre causal) a priori les réceptions à arriver à un moment donné, donc on ne peut pas considérer automatiquement qu'il y a un ordre total (l'ordre visuel) sur chaque processeur. Le problème est de savoir si l'ordre causal engendre malgré tout l'ordre visuel.

Nous nous référerons ici à [AHP96].

Ce problème est intéressant car on ne peut pas a priori utiliser l'idée de Plandowski car on ne peut pas générer les réponses par processeurs, on a besoin des dépendances entre les messages.

Caractérisation (C) : M n'a pas de concurrence $\Leftrightarrow \forall receive(i,j)$, l'événement de j précédant cette réception est un envoi, et cet envoi est causalement avant le $send(i,j)$ associé à cette reception.

Pour obtenir un algorithme polynomial, on imagine un algorithme récursif et en programmation dynamique pour éviter de recalculer un nombre exponentiel de fois la même chose.

La question est : pour chaque boîte, qu'est ce qui est utile ?

On va avoir besoin de regarder, pour chaque réception, ce qu'il y avait avant (pour trouver un envoi correspondant). Il faudra donc se souvenir du premier événement et du dernier événement de chaque nMSC, pour tout processus i.

Algorithme :

On lance l'algorithme sur les nMSC de hiérarchie inférieure.

Ensuite, pour tout processus i, pour tout événement e sur i,

- si e est le premier/dernier événement du MSC sur le processus i, on enregistre cet événement. Si l'événement est un nMSC, on enregistre le premier/dernier événement de i de ce nMSC, puisque ce nMSC a déjà été traité.
- si e est un envoi, on ne fait rien.
- si e est un réception, d'envoi associé s, on regarde le dernier événement f sur le processus i avant e.
  - Si f est une réception on renvoie faux
  - Si f est un envoi, on vérifie que $f \prec s$, sinon on renvoie faux.
  - Si f est un nMSC, on vérifie que le dernier élément de f(il a déjà été enregistré) est un envoi précédent causalement s. Sinon on renvoie faux.
- si e est un nMSC, on regarde f, le premier événement sur i de e.
  - si f est un envoi, on ne fait rien
  - si f est une réception (d'envoi associé s), on s'assure que le dernier événement de i avant f est un envoi, ou un nMSC finissant par un envoi, précédent causalement s.

renvoyer Vrai.

Complexité : l'algorithme est lancé une fois par nMSC. A chaque appel récursif sur M, il demande $O(\sum_i(|M_i|.\text{ complexité(futur)}))$. Or la complexité du futur se fait en temps $O(|M|(p+c)^2)$. Ainsi l'algorithme prend un temps $O(\sum_M(|M|^2(p+c)^2) = O((p+c)^2|N|^2)$.

Rappelant que l'algorithme de [AHP96] sur les MSC était en temps N^2, cela ne semble pas mauvais.

9 Appartenance d'un MSC à un nHMSC

Comme tous les problèmes considérés pour les nMSC (sauf le plongement) ont une complexité comparable pour les nMSC à celles des MSC, on s'intéresse maintenant aux problèmes pour les nHMSC.

Il est à noter que d'après [Pap94], on sait que certains problèmes sur les graphes donnés de manière hiérarchique sont NEXPSPACE-complet. On s'attend donc à trouver des problèmes difficiles du point de vue de la complexité.

Définition 20 *Problème d'appartenance hiérarchique : étant donnés un nMSC M et un nHMSC G, est ce que $M \in L(G)$*

Rappelons que [AEY01] montre que ce problème est NP-complet dans le cas de MSC non hiérarchiques, mais dans P à nombre de processus fixés.

Ici, les choses sont moins favorables pour nous.

Eliminons tout d'abord le cas facile où on teste

Définition 21 *Problème d'appartenance semi-hiérarchique : étant donnés un MSC M et un nHMSC G, est ce que $M \in L(G)$?*

La différence avec le problème précédent est que M n'est pas hiérarchique. Donc le chemin pour lire M dans G (si M est dans G) est de taille polynomiale. On peut le deviner de manière non déterministe puis vérifier qu'il convient. Le problème d'appartenance faible est donc dans NP, et NP-complet grâce à [AEY01].

Cependant, dès que le MSC est hiérarchique, on a :

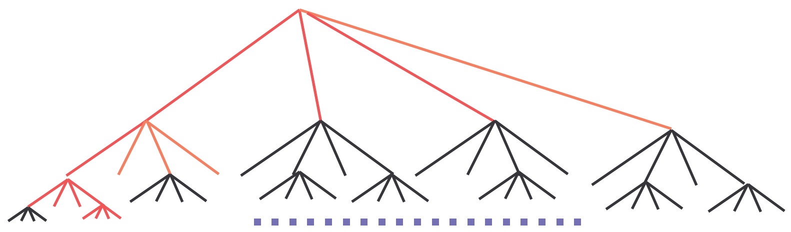
Théorème 8 *Le problème d'appartenance hiérarchique, même si le graphe est donné de manière non hiérarchique, est PSPACE-complet.*

démonstration

Montrons tout d'abord le caractère (N)PSPACE du problème.

On va commencer par définir une preuve du fait qu' un nMSC M appartient à un nHMSC G. Ensuite, on va montrer que l'on a besoin de se rappeler simplement un nombre polynomial de symboles pour tester si cette preuve est correcte.

La preuve se présente sous forme d'arbre



Chaque sous arbre représente la preuve entière de la proposition associée à sa racine.

On pose C l'ensemble des configurations de G. Une configuration est la position exacte dans le graphe : on indique pour tout processus i la dernière action (a_i) à avoir été effectuée (ou aucune si aucune n'a été exécutée dans le nHMSC). Si cette action est le début d'une sous-boîte, il faut rajouter la configuration de la sous-boîte dans laquelle on se trouve. Ainsi, une configuration peut être décrite de manière polynomiale. De plus, on impose que $\{e_i \leq a_i \mid i \in P, e_i \in G|_i\}$ soit un préfixe bien formé.

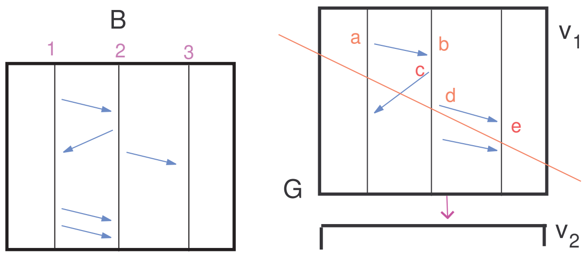
Pour qu'une configuration soit exploitable, il faut considérer uniquement les configurations pour lesquelles tous les processus se trouvent dans le même noeud de G(et ce pour tout niveau de hiérarchie). Sinon, on risque de suivre un chemin différent pour deux processus différents, ce qui n'est pas permis. On note l'ensemble de ces configurations CC.

Si on a un chemin dans G, on peut le représenter par une linéarisation du MSC qu'il représente. Une fois qu'un processus i a fini de lire un noeud, il ne peut pas empêcher les autres processus d'avancer dans ce noeud. Sinon, il y aurait une action dans le futur de i situé dans le même noeud, ce qui est impossible car il n'y a plus d'action sur i sur ce noeud, et qu'on interdit à un noeud futur d'envoyer un message dans un noeud passé. Cela serait possible en considérant les messages compositionnels de [GMP01], ce que l'on exclut ici. Ainsi, on peut toujours choisir une linéarisation qui ne commence pas un nouveau noeud avant d'avoir complètement fini celui traité. Cela revient en fait à considérer les synchronisations fortes sur le graphe, ce qui n'est pas restrictif si on garde la liberté du côté de M.

Ainsi, à chaque noeud de l'arbre, on associe un label $(v_1, (A_i, a_i, b_i)_i, v_2) \in CC \times (NT \times \mathbb{N}^2)^p \times CC$. Cette proposition $(v_1, (A_i, a_i, b_i)_i, v_2)$ est vraie si et seulement si il existe un chemin ρ dans G depuis l'arrête v_1 et jusqu'à l'arrête v_2 avec $\forall i, \rho|_i$ est le facteur engendré par la projection sur i de A_i (symbole non terminal de M), compris entre le a_i^{eme} caractère et le b_i^{eme}.

On pose $A_i = B_i C_i$ dans le nMSC.

La preuve de $(v_1, (A_i, a_i, b_i)_i, v_2)$ est constituée par $p + 1$ labels (au plus) $l_0 \cdots l_p$. On crée un label à chaque fois que l'on passe de la sous nMSC B_i à C_i pour un i quelconque.



$$l_0 = \begin{pmatrix} (v_1, \emptyset) & (B, 0, 1) & (v_1, a) \\ (v_1, \emptyset) & (B, 0, 3) & (v_1, d) \\ (v_1, \emptyset) & (B, 0, fin) & (v_1, e) \end{pmatrix}$$

Si on est au plus bas niveau de la hiérarchie, on devine le chemin qui réalise les (A_i, a_i, b_i) entre v_1 et v_2. La taille est polynomiale en $|M|$ car il n'y a pas de hiérarchie !) .

Un tel arbre de preuve existe si et seulement si $M \in G$.

Le fait que le graphe soit ou non hiérarchique ne rajoute pas de complexité car dans les deux cas on peut décrire de manière polynomiale l'endroit où l'on se trouve dans le graphe.

Le seul problème qui reste est que l'arbre possède p^n noeuds, donc un nombre exponentiel.

Cependant, on a besoin au maximum d'un seul ensemble de noeuds par niveau de hiérarchie (voir les noeuds rouges sur la figure).

Effectivement, quand on passe à la vérification de la preuve d'un frère, on peut oublier la preuve du noeud qu'on a traité. De même, on n'engendre les preuves de manière non déterministe que lorsqu'on en a vraiment besoin (on ne va pas engendrer la preuve d'un frère avant d'avoir traité le noeud courant). Ainsi, on a besoin d'un espace polynomial.

Conclusion : Le problème d'appartenance hiérarchique est dans PSPACE. $\square$

démonstration Passons maintenant au caractère PSPACE-dur de l'appartenance hiérarchique. Pour ça, on va montrer que le problème est PSPACE-dur même si le graphe n'est pas hiérarchique (ce qui impliquera que le problème pour un nHMSC est aussi PSPACE-dur. Il est à noter que la preuve directe du caractère PSPACE-dur pour un nHMSC est plus facile).

On va réduire TQ(1-in-3)BF (true quantified one-in-three boolean formula) à notre problème. Il est montré dans [Sch78] que TQ(1-in-3)BF est PSPACE-complet (on applique le théorème 6.1 à R=$\{$(0,0,1),(0,1,0),(1,0,0)$\}$, qui n'est pas dans les cas c),d),e) ou f) d'après les lemmes 3.1A, 3.1B et 3.1W). Nous donnons également en annexe une démonstration directe que TQ(1-in-3)BF est PSPACE-complet

Soit $F = \forall x_n \exists x_{n-1} \cdots \exists x_1 \phi$ une instance de TQ(1-in-3)BF. La formule ϕ est de la forme $\wedge_i R(\alpha_{i,1}, \alpha_{i,2}, \alpha_{i,3})$, avec $R(x, y, z)$ vraie si et seulement si une et une seule de ses variables est vraie.

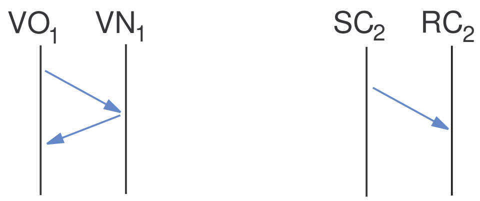
On pose $F_{2i+1} = \forall x_{2i+1} \exists x_{2i} \cdots \exists x_1(\phi)$ et $F_{2i} = \exists x_{2i} \forall x_{2i-1} \cdots \exists x_1(\phi)$.

ϕ comporte m clauses $(C_1 \cdots C_j \cdots C_m)$ sur les variables $x_1 \cdots x_n$.

On va raisonner sur des MSC-graphes G_i et un nMSC M_i mettant en oeuvre $2n + 2m$ processus. Ces processus sont SC_j, RC_j, VO_i, VN_i (S désigne un émetteur, R un destinataire, C pour les clauses, V pour les variables, O pour oui, N pour non).

On pose pour tout i, xo_i le MSC constitué de l'envoi d'un message de VO_i à VN_i puis de VN_i à VO_i, et de l'envoi des messages de SC_j à RC_j pour tout j tel que x_i apparaît dans C_j : Si x_i est vrai, C_j a un littéral de plus à vrai.

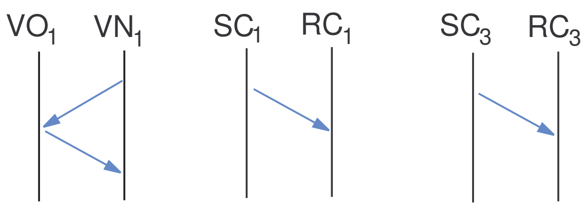
Exemple : xo_1



$[\neg x_1 \vee x_2 \vee x_3] \wedge [x_1 \vee x_2 \vee x_3] \wedge [\neg x_1 \vee \neg x_2 \vee x_3]$

On pose, pour tout i, xn_i le MSC constitué de l'envoi d'un message de VN_i à VO_i puis de VO_i à VN_i, et de l'envoi des messages de SC_j à RC_j pour tout j tel que $\neg(x_i)$ apparaît dans C_j : Si x_i est faux, C_j a un littéral de plus à vrai.

Exemple : xn_1

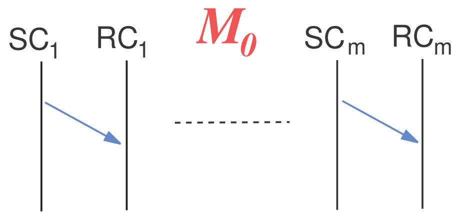


$[\neg x_1 \vee x_2 \vee x_3] \wedge [x_1 \vee x_2 \vee x_3] \wedge [\neg x_1 \vee \neg x_2 \vee x_3]$

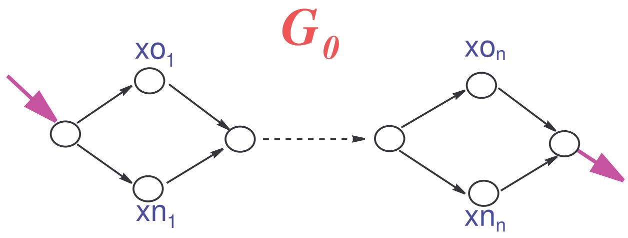
Notons que les seuls messages ne pouvant pas commuter sont ceux concernant VO_i et VN_i.

On définit (G_i, M_i) de manière récursive, suivant le nombre de variables n de ϕ. On va associer (G_i, M_i) à F_i.

M_0 est un MSC constitué d'un unique message de SC_j à RC_j pour tout j.



G_0 est un MSC-graphe, composé de $4n$ noeuds, associées aux xo_i, xn_i (ou vides). En outre, il y a un noeud initial et un noeud final. Le graphe consiste à choisir pour tout i un noeud xo_i ou xn_i.



Intuitivement, si M appartient à G, c'est qu'il y a un chemin ρ de G représentant M. Le chemin ρ correspond à choisir si x_i vaut 0 (cas xn_i) ou 1 (cas xo_i). Alors, on regarde si toutes les clauses contiennent exactement un littéral assigné à vrai. En effet, le nombre de messages de SC_j à RC_j de ρ est le nombre de littéraux à vrai pour C_j. Or M_0 comporte un et un seul message de SC_j à RC_j.

Ne reste plus qu'à s'occuper des quantificateurs.

On pose a_i le message de VO_i à VN_i, et b_i le message de VN_i à VO_i. Le doublet $a_i b_i$ signifie que x_i est vrai, $b_i a_i$ signifie qu'il est faux.

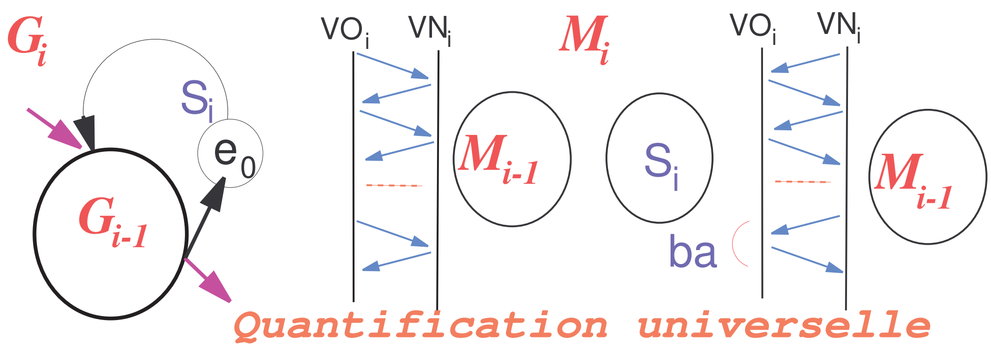
Supposons avoir défini (G_{i-1}, M_{i-1}).

Si $F_i = \forall x_i F_{i-1}$, on pose $M_i = (ab)^{2^{i/2}} M_{i-1} S_i (ba)^{2^{i/2}} M_{i-1}$ (grâce à la hiérarchie, la taille reste linéaire). On a besoin d'autant de messages «ab» ou «ba» car il faut qu'ils se distribuent sur toutes les sous boucles (dues aux quantificateurs universelles des variables déjà traitées) : il en faut un par passage de boucle.

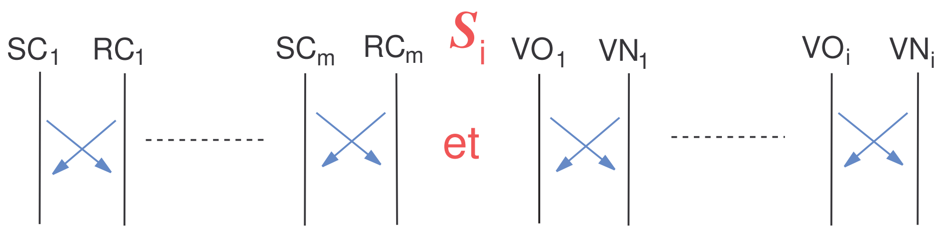
S_i est un MSC "synchronisant", que l'on définira plus tard.

On pose $G_i = (V_i, E_i)$, avec le noeud in initial et f final identiques à ceux de G_{i-1}. De plus, $V_i = V_{i-1} \cup \{e_0\}$.

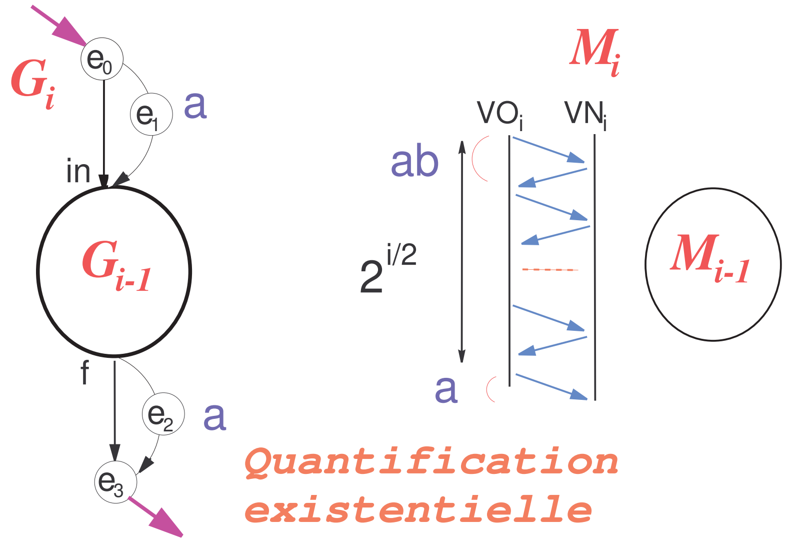
En outre, $E_i = E_{i-1} \cup \{(f, e_0), (e_0, in)\}$ (on fait une boucle sur G_i en passant par e_0). On associe le MSC S_i au noeud e_0.



S_i consiste en une série de messages croisés entre RC_j et SC_j pour tout j, et entre VO_k et VN_k pour $k \leq i$. Ces messages associés à e_0 permettent de «synchroniser» d'une certaine manière les messages. S'il n'y avait pas ce S_i, on pourrait utiliser de manière trop libre le quota dû aux deux M_{i-1}, alors qu'il faut qu'à chaque passage de boucle, il y ait exactement une variable à vrai par clause.



Comme il y a une synchronisation S_i, sur le premier M_{i-1}, le graphe doit passer par $2^{i/2}$ $(ab)_i$. Donc à chaque fois il passera par xo_i, ie il considérera x_i comme étant vrai. De plus, après la synchronisation, il va considérer la même chose, mais avec x_i à faux. Donc F_i est vérifiée si et seulement si pour $x_i = 1, F_{i-1}$ est vraie et pour $x_i = 0, F_{i-1}$ est vraie. D'où la conclusion.



Sinon, $F_i = \exists x_i F_{i-1}$, et on pose $M_i = (a_i b_i)^{2^{i/2}}(a_i)M_{i-1}$

En outre, $G_i = (V_i, E_i)$, où $V_i = V_{i-1} \cup \{e_o, e_1, e_2, e_3\}$.

$E_i = E_{i-1} \cup \{(e_o, in), (e_f, e_3), (e_0, e_1), (e_1, e_i), (f, e_2), (e_2, e_3)\}$, où in désigne le noeud initial de G_{i-1}, et f le noeud final de G_{i-1}. Les nouveaux noeuds initiaux et finaux de G_i sont e_0 et e_3. On associe a_i à e_1 et à e_2, le vide pour e_0 et e_3.

Intuitivement, on a besoin de $2^{i/2}a_i$ et $2^{i/2}b_i$ (dus aux tours de boucles des quantificateurs universels). Or M_i donne un a de plus. Donc on peut et on doit passer par e_1 ou (exclusif) par e_2, pour consommer le dernier a_i. Si on choisit de passer par e_1, on va lire des ba à chaque fois, donc on sera obligé de choisir xn_i ($x_i = 0$) pour toute valuation. Si on choisit de passer par e_2, on aura à chaque fois ab, et donc xo_i ($x_i = 1$). Comme le graphe nous donne le choix, $M_i \in G_i$ si et seulement s'il existe un x_i tq F_{i-1}. CQFD.

□

Remarque 8 On peut remarquer que dans la preuve, on utilise un nombre polynomial de processus, donc on a peut être une chance de simplifier l'algorithme si le nombre de processus est fixé.

9.1 Clôture faible de nHMSC

Reprenant les travaux de [AEY01], certains problèmes (celui de l'appartenance par exemple), ont l'air d'être plus simples en considérant la clôture faible de L(G), notée $L^w(G)$. Ce problème revient à la reconnaissance d'un mot par un automate, et est donc intéressant.

Le problème est l'appartenance hiérarchique :

Définition 22 *Etant donné M un nMSC, G un nHMSC, est ce que* $\forall i \in P, M|_i \in G|_i$, *où* $X|_i$ *désigne la projection sur le processus p.*

Dans le cas sans hiérarchie, la facilité vient du fait que l'on ne considère plus un système concurrent.

Ici encore, le problème est plus facile : on peut entre autres appliquer la représentation grammaticale des $M|_i$.

Traitons le cas où M est un mot hiérarchique et G un automate.

Algorithme : Pour tout A non-terminal de M, pris par ordre croissant de taille, on peut calculer tous les couples de noeuds (a, b) tq il existe un chemin de a à b étiqueté par A. Pour les calculer, c'est très facile. Supposons que $A \longrightarrow BC$. Or on a déjà calculé les ensembles pour B et C. Il suffit de calculer pour tout (a, b) tq il existe un chemin de a à b étiqueté par B, l'ensemble des (b, c) tq il existe un chemin étiqueté par C. Ainsi l'ensemble correspondant à A est l'ensemble des (a, c). Il ne reste plus qu'à regarder s'il y a un (a_0, b_0) associé à M, avec a_0 initial et b_0 final.

Complexité : On regarde pour tout non-terminal (il y en a $|M|$), $|G|^2$ couples (a, b) puis $|G|$ couples (b, c) correspondant. D'où une complexité en $O(|M||G|^3)$.

Considérons maintenant que le mot $M = M_1 \cdots M_{||M||}$ est non hiérarchique mais que l'automate G l'est.

Algorithme : Pour tout sous-automate A de G, on calcule

$$D_A = \{(i,j) \mid M_i \cdots M_j \text{ est accepté pas } A\}.$$

- Si A est au plus bas niveau de la hiérarchie, c'est évident.
- Sinon, pour tout $k = 0, \ldots, |M|$, pour tout $i = 0, \ldots, |M| - k$, calculer $C_{(k,i)} = \{(a,b) \mid$ il existe un chemin de a à b étiquetté par $M_i \cdots M_{i+|k|}\}$:
  - $C_{(k,i)} = \emptyset$
  - Pour tout $l < k$, rajouter $\{(a,c) \mid \exists b \text{ tq } (a,b) \in C_{(l,i)} \& (b,c) \in C_{(k-l-1,i+l-1)}\}$ à $C_{(k,i)}$ (on est amené à se servir des D_A pour A de niveau hiérarchique plus bas).
- $D_A = \{(i, i+j) \mid (a_i, a_f) \in C_{(j,i)}\}$, où a_i et a_f sont les noeuds initiaux et finaux de A.

Complexité : Pour tout A, on fait des calculs pour tout (k,i,l), et tout (a,b,c) de A. Donc $\mathrm{O}(|M|^3|A|^3)$ calculs.

Soit au total $O(|M|^3|G|^3)$ calculs.

Cependant, on peut montrer que le problème

Théorème 9 *Si M est un mot hiérarchique et G un automate non déterministe hiérarchique, décider si $M \in L(G)$ est PSPACE-complet, même si G est cyclique*

démonstration

Pour le caractère PSPACE-facile, on peut appliquer l'algorithme de la section précédente.

Passons au caractère PSPACE-dur : le problème est qu'il n'y a plus de concurrence : on n'a plus des actions qui peuvent commuter les unes avec les autres, mais uniquement des mots sur l'alphabet $\{a,b\}$.

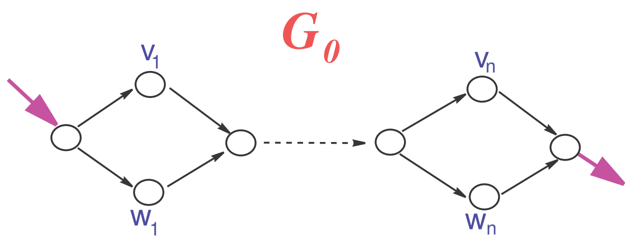
Cependant on va pouvoir adapter la réduction de TQ(1-in-3)BF grâce à la remarque suivante : deux mots de $\{a\}^\star$ commutent . La lettre b servira à synchroniser les a. En revanche, nous ne pouvons pas choisir les messages à synchroniser : tout est synchronisé ou rien. C'est une astuce qui va nous tirer d'affaire.

La hiérarchie va nous permettre de faire suffisamment de messages pour avoir un codage *injectif*.

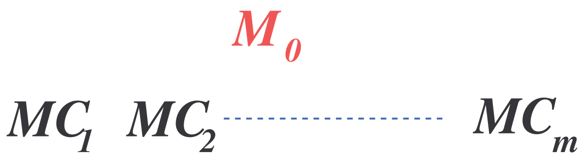
On a une injection (et même une bijection) f de $\{0,1,2,3\}^n$ dans $\{0 \ldots 4^{n+1} - 1\}$, qui est $f((y_i)) = \sum_{i \leq n}(y_i(4)^i)$, la bijection réciproque étant $f^{-1}(x) = (y_i)_i$, avec $y_i = (x \mod 4^{i+1})/4^i$.

Commençons par réduire le problème (1-in-3)SAT au problème d'appartenance à un automate sur un alphabet à une lettre (on montre donc le problème NP-dur). Soit φ une instance de ce problème, avec n variables et m clauses.

L'automate G_0 conserve la même fonction que pour le cas de l'inclusion hiérarchique : pour tout i, choisir entre vi (pour x_i vrai) et w_i (pour x_i faux).



M_0 quant à lui garde le même rôle (vérifier que toutes les clauses possèdent une et une seule variable à vrai), mais change de définition : On pose $M_0 = MC_1 \cdots MC_m$.



Les $MC_1 \cdots MC_m$ font référence aux clauses $C_1 \dots C_m$ de φ, et seront définis plus tard.

On pose pour tout i, v_i le mot constitué des MC_j pour tout j tel que x_i apparaît dans C_j.

On pose pour tout i, w_i le MSC constitué des MSC MC_j pour tout j tel que $\neg x_i$ apparaît dans C_j.

Il ne reste plus qu'à regarder ce dont on a besoin comme MC_j.

Quand on parcourt le graphe, on lit au plus 3 fois la même clause (si on assigne la valeur vraie aux trois littéraux de la clause).

Ainsi, il suffit de prendre $MC_j \in \{a\}^\star$ de longueur 4^j, ce qui est facile avec la hiérarchie :

$$MC_1 = aaaa, \cdots, MC_m = MC_{m-1}.MC_{m-1}.MC_{m-1}.MC_{m-1}$$

La seule chose que fait l'automate est compter le nombre de a lus. Ainsi, voir MC_i puis MC_j revient à ajouter 4^i a à 4^j a. Ainsi, le nombre de a lus quand on arrive à la fin d'un chemin est $\sum_{i \leq m}(y_i(4)^i)$, où y_i est le nombre de fois où on a vu MC_i, donc le nombre de littéraux à vrai pour la clause C_i.

Or le chemin accepte M_0 si et seulement si on a lu $\sum_{i \leq m}(4)^i$ a. Comme cette fonction est injective, cette condition équivaut à $\forall i, y_i = 1$.

Ainsi, $M_0 \in L(G_0) \Leftrightarrow \exists v \mid$ toutes les clauses de $\varphi(v)$ possèdent un unique littéral à vrai sur le chemin de G_0 reconnaissant M_0. **CQFD**.

Passons à la réduction de TQ(1-in-3)BF au problème d'appartenance à un automate sur deux lettres.

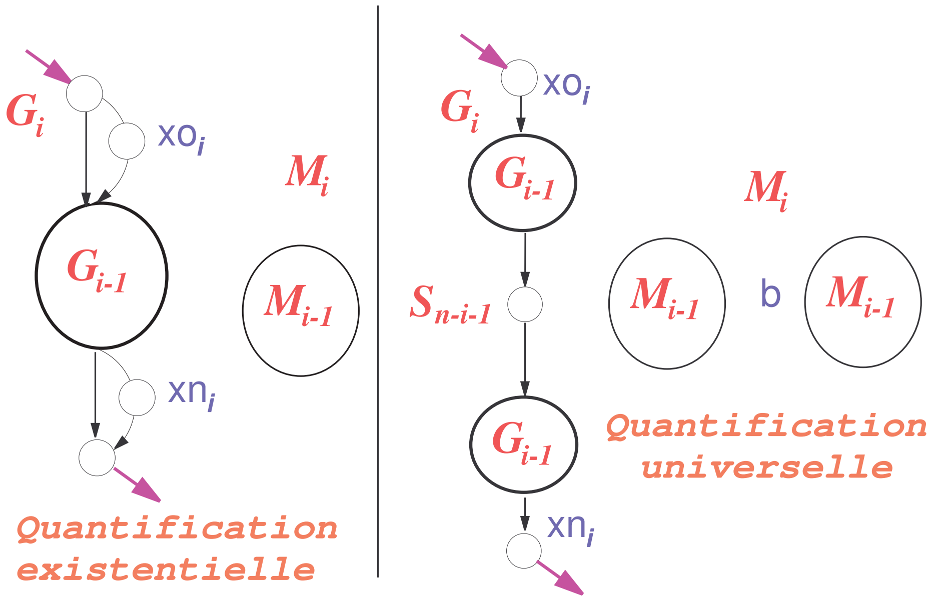
On pose $MV_i = 4^{m+i}\ a$ relatif aux variables (on commence à $4^m\ a$ pour ne pas interférer avec les clauses : le codage sera toujours bijectif).

On pose $xo_i = (v_i.MV_i)$, et $xn_i = (w_i.MV_i)$.

On va poser $M_0 = \sum_j (MC_j) + \sum_i (MV_i) = \sum_{k \leq m+n} 4^k\ a$, et G_0 l'automate avec juste un noeud initial et un noeud final.

Pour la quantification existentielle, on fait de même que pour la preuve précédente : on laisse le choix de prendre soit xo_i (on le fait au début pour que ce choix se répercute dans toutes les boucles grâce à S), soit xn_i (à la fin pour les mêmes raisons).

De même, pour la quantification universelle, on enchaîne la prise de xo_i au début, le graphe (avec $x_1 = 1$ donc), une synchronisation S_{n-i}, le graphe (mais avec $x_i = 0$ car on n'a pas lu xo_i, donc on n'a pas encore vu de MV_i), puis la prise de xn_i à la fin. Ainsi, il existe un chemin acceptant si et seulement si M est reconnu pour $x_i = 0$ et $x_i = 1$.

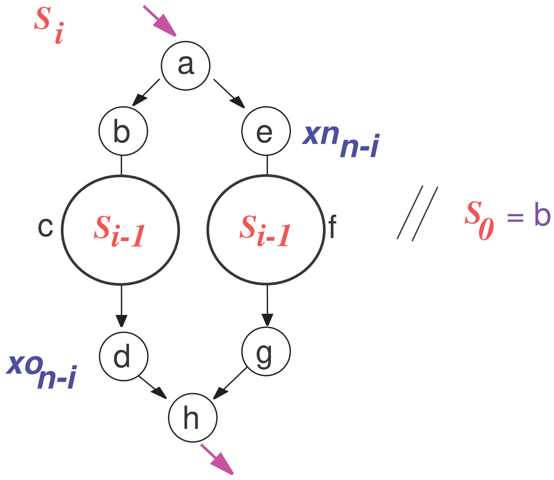


Les sous-automates (S_i) synchronisant sont définis : $S_0 = b$.

$S_i = (V, E)$, avec $V = \{a, b, c, d, e, f, g, h\}$ et

$E = \{(a, b), (b, c), (c, d), (d, h), (a, e), (e, f), (f, g), (g, h)\}$.

On étiquette a, b, g et h par le vide, c et f par S_{i-1}, d par xo_{n-i} et e par xn_{n-i}.



Le but de S_i est double : premièrement, quand on voit un quantificateur universel, on est obligé de lancer deux fois le test. Donc il faut régénérer les mêmes valeurs des x_j une deuxième fois.

Deuxièmement, il ne faut pas que les deux tests interférent l'un avec l'autre, donc on pose $S_0 = b$ pour qu'il y ait une synchronisation des messages lors du passage entre les deux tests.

De même que précédemment, si le chemin reconnaît M_n, c'est que entre chaque b (donc entre chaque synchronisation), il a reconnu un seul MV_i.

Donc soit un MV_i a été lu avant d'arriver à S, et alors il prend la première branche, qui le force à prendre xo_i. Sinon il ne l'a pas lu, et alors il prend la seconde branche (xn_i), il ne le lira pas après le b, donc sera obligé de le lire par la suite en reprenant un xn_i la fois d'après. Ainsi, dès qu'une valeur à été affectée à x_i, elle n'en change pas, jusqu'à ce que l'on revienne au niveau i. CQFD

□

Corollaire 2 *On peut simuler les lettres a et b de l'automate par un message de 1 à 2 et un message de 1 à 3. Ainsi, rien qu'avec la projection sur le premier processus (le graphe accepte n'importe quoi sur les deux autres composantes), on peut simuler l'appartenance d'un mot dans un automate avec le problème d'appartenance hiérarchique faible. Ce problème est donc PSPACE-complet.*

Remarque 9 Même à nombre de processus fixé, le problème d'appartenance hiérarchique est PSPACE-complet (puisqu'il l'est avec trois).

La hiérarchie étant analogue à écrire en binaire un nombre, nous pouvons adapter notre preuve pour dire qu'il est NP-dur de dire si :

Etant donné S un nombre écrit en binaire,

A un automate où les états sont des nombres écrits en binaire

Question : Existe-il un chemin dans A tq la somme des nombres sur ce chemin soit S ?

M	G	automate	MSC concurrent
Plat	Plat	P	NP-complet
Plat	Nested	P	NP-complet
Nested	Plat	P	PSPACE-complet
Nested	Nested	PSPACE-complet	PSPACE-complet

FIG.2 – M appartient à G ?

10 Intersection

L'intersection de deux nHMSC est intéressante car elle permet de faire du model Checking : on peut spécifier par un nHMSC N les mauvais comportements, et demander si M admet un de ces mauvais comportements, ie si $M \cap N \neq \emptyset$.

De plus, ce problème oblige à réduire la classe des nHMSC : en effet, pour des MSC-graphes quelconques, la vacuité de leur intersection est indécidable.

Ce paragraphe s'appuiera sur les papiers de [MP99] et de [AY99].

Pour contourner le problème, les deux articles font référence aux MSC-graphes bornés (ou bounded).

Définition 23 *Soit M un MSC-graphe, b un chemin de M. Le graphe de communication de b est $\{(p, q) \mid p$ envoie un message à q dans $b\}$.*

Définition 24 *Soit M un MSC-graphe. M est borné si et seulement si pour toute boucle b de M, le graphe de communication de b est connexe.*

Les deux articles concluent :

Théorème 10 *Savoir si $M \cap N = \emptyset$ est PSPACE-complet pour M et N des MSC-Graphes, M borné. De plus, ce problème est indécidable pour M et N quelconques.*

Nous ne donnerons ici qu'une idée de la preuve pour la borne supérieure.

démonstration

Il existe un automate A de taille exponentiel reconnaissant toutes les linéarisations des chemins de M, car M est *borné*. On fait ensuite de l'accessibilité (NLOGSPACE) dans l'automate produit, de taille exponentielle, pour savoir s'il existe un chemin ρ de N possédant une linéarisation reconnue par A.

Il est à noter que si une linéarisation de ρ est reconnue par A, toutes sont reconnues par A (car A est fermé pour les linéarisation).

□

Cependant, [AY99] montre qu'on ne peut pas faire mieux que déplier les nHMSC et appliquer l'algorithme du cas sans hiérarchie (pour lequel l'algorithme est PSPACE-complet) :

Théorème 11 *Savoir si $M \cap N = \emptyset$ est EXPSPACE-complet pour M et N des nHMSC, avec M borné.*

Il reste tout de même à considérer le cas où M n'est pas hiérarchique mais N l'est.

Il existe un automate A de taille exponentiel reconnaissant toutes les linéarisations des chemins de M, car M est *borné*. Le nHMSC N déplié est lui aussi de taille exponentielle (on ne le déplie pas !). On fait ensuite de l'accessibilité (NLOGSPACE) sur l'automate produit, de taille exponentielle, pour savoir s'il existe un chemin ρ de N possédant une linéarisation reconnue par A.

On en conclut

Théorème 12 *Savoir si $M \cap N = \emptyset$ est PSPACE-complet pour un MSC-graphe M borné et un nHMSC N.*

Conclusion

On peut dresser les tableaux des complexités de chaque problème. On notera en noir les résultats déjà connus, en couleur les résultats nouveaux du papier (sauf le seul résultat rouge, déjà connu). En bleu, on note les temps restant dans la même classe de complexité que le cas plat, en vert ceux passant dans une classe de complexité plus grande, et enfin en rouge celles dans une classe de complexité exponentiellement plus grande (cas le pire).

M	fifo	futur	S-réalisation	concurrence
Plat	$\|M\|^2$	$\|M\|$	$p\|M\|^2$	$\|M\|^2$
Nested	$\|M\|^2$	$(p+c)^2\|M\|$	$p\|M\|^5$	$(p+c)^2\|M\|^2$

M	N	motifs (N sans port)	plongement
Plat	Plat	$\|M\|\|N\|$	$\|M\|+\|N\|$
Plat	Nested	$\|M\|\|N\|^2$	$(p+c)^2\|M\|\|N\|$
Nested	Plat	$\|M\|\|N\|$	$\|M\|+\|N\|$
Nested	Nested	$\|M\|^2\|N\|^2$	PSPACE (NP-dur ?)

M	G	appartenance faible	appartenance	intersection
Plat	Plat	$\|G\|+\|M\|$	NP-cplt	PSPACE-cplt
Plat	Nested	$\|M\|^3\|G\|^3$	NP-cplt	PSPACE-cplt
Nested	Plat	$\|M\|\|G\|^3$	PSPACE-cplt	PSPACE-cplt
Nested	Nested	PSPACE-cplt	PSPACE-cplt	EXPSPACE-cplt

On peut ainsi se rendre compte qu'à part pour le plongement, la hiérarchie ne coûte rien pour les nMSC.

Un surcoût peut arriver quand on considère les nHMSC, même si la complexité devient rarement exponentiellement plus grande, comme on aurait pu le craindre au début.

Cependant, le problème n'était pas de militer pour ou contre la hiérarchie : écrire de manière hiérarchique ne peut qu'accélérer les algorithmes, et souvent de manière exponentielle par rapport au cas sans hiérarchie.

De toute façon, en pratique, la hiérarchie est utilisée dans les spécifications , donc on se devait de trouver les meilleurs algorithmes possibles, ce qui était loin d'être le cas avant.

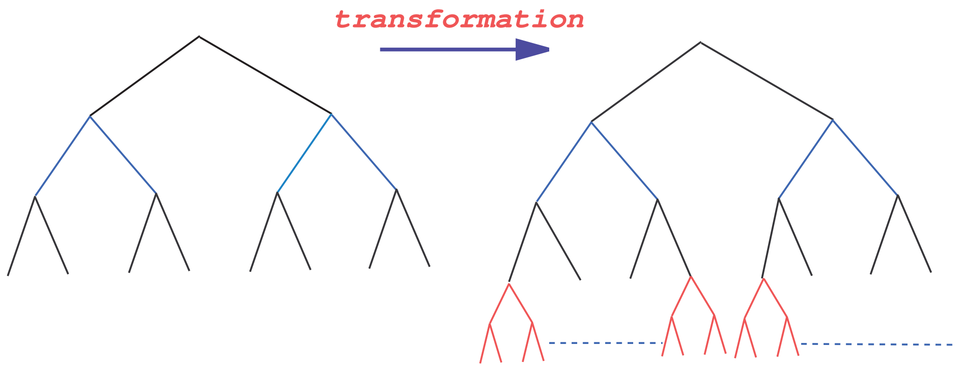
Annexe

Lemma 5. *TQ(1-in-3)BF se réduit polynomialement à TQ(3-CNV)BF.*

démonstration

Soit ϕ en forme 3 CNV sur n variables. On va montrer qu'il existe ψ sur $48n$ variables tq $\forall v$ valuation, $\phi(v)$ est vrai $\Leftrightarrow \exists v'$ prolongeant v tq $\psi(v')$ est vrai.

Ainsi, on peut transformer tout arbre de preuve d'une instance de TQBF en TQ(1-in-3)BF, en rajoutant des noeuds existentiels au plus bas niveau pour quantifier les variables additionnelles.



Il ne reste plus qu'à montrer la propriété.

On va passer par NAESAT (Not all equal).

Soit ϕ une instance de 3CNV SAT.

A toute clause $C_j = (x \vee y \vee z)$ on associe $\exists(a,b)$ tq $C'_j = R(x,y,a) \wedge R(z,\neg a,b) \wedge R(a,b,1)$, où $R(x,y,z)$ est faux si et seulement si $x = y = z$. Il est très facile de montrer que cela convient.

Exemple : pour C_j faux, $x = y = z = 0$. Supposons qu'il existe a, b tq C'_j est vrai. Alors $a = 1$, donc $b = 1$. Mais la troisième clause à tous ses littéraux à vrai, et donc $R(a,b,1) = R(1,1,1)$ est faux, donc C'_j aussi.

De même, prenons ϕ une instance de NAESAT.

A toute clause $C_j = R(x,y,z)$, on associe $\exists(a,b,c,d,e)$ tq $C'_j = S(x,a,b) \wedge S(\neg y,b,c) \wedge S(z,c,d) \wedge S(a,d,e)$, avec $S(x,y,z)$ vraie si et seulement si $x = 1, y = z = 0$ ou $x = z = 0, y = 1$ ou $x = y = 0, z = 1$ De même, il est très facile de montrer que cela convient.

Exemple : Supposons que $x = y = z = 0$. C_j est donc fausse. Supposons qu'il existe a, b, c, d, e tq C'_j soit vraie. Nécessairement, $b = c = 0$ pour que la clause concernant y soit vraie. Donc $a = d = 1$. La quatrième clause a deux littéraux à vrai, elle est donc fausse.

De plus, toutes ces réductions sont polynomiales.

CQFD

□

Corollaire 3 *TQ(1-in-3)BF est PSPACE-complet car TQ(3CNV)BF l'est (voir [Pap94]).*

Références

[AEY00] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Inference of message sequence charts. In *Proceedings of the 22nd International Conference on Software Engineering, Limerick (Ireland)*, pages 304–313. ACM, 2000.

[AEY01] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Realizability and verification of msc graphs. In *Proceedings of ICALP*, 2001.

[AHP96] Rajeev Alur, Gerard H. Holzmann, and Doron A. Peled. An analyzer for message sequence charts. *Software Concepts and Tools*, 17(2) :70–77, 1996.

[AY99] Rajeev Alur and Mihalis Yannakakis. Model checking of message sequence charts. In *Proceedings of the 10th International Conference on Concurrency Theory CONCUR'99, Eindhoven (The Netherlands)*, number 1664 in Lecture Notes in Computer Science. Springer, 1999.

[FH65] N.J. Fine and H.S.Wilf. Uniqueness theorems for periodic functions. *PAMS : Proceedings of the American Mathematical Society*, 16, 1965.

[GMP01] Elsa Gunter, Anca Muscholl, and Doron Peled. Compositional message sequence charts. In T. Margaria, editor, *Proceedings of the Tools and Algorithms for the Construction and Analysis of Systems, 7th International Workshop (TACAS'01)*, Lecture Notes in Computer Science. Springer, 2001. To appear.

[itu96] Itu-ts recommendation z120, 1996.

[MP99] Anca Muscholl and Doron Peled. Message sequence graphs and decision problems on Mazurkiewicz traces. In M. Kutylowski and L. Pacholski, editors, *Proceedings of the 24thSymposium on Mathematical Foundations of Computer Science (MFCS'99), Szklarska Poreba (Poland) 1999*, number 1672 in Lecture Notes in Computer Science, pages 81–91. Springer, 1999.

[MPS98] Anca Muscholl, Doron Peled, and Zhendong Su. Deciding properties of message sequence charts. In M. Nivat, editor, *Proceedings of the 1st International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'98), Lisbon, Portugal, 1998*, number 1378 in Lecture Notes in Computer Science, pages 226–242. Springer, 1998.

[MST97] M. Miyazaki, A. Shinohara, and M. Takeda. An improved pattern matching algorithm for strings in terms of straight-line programs. In A. Apostolico and J. Hein, editors, *Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching*, number 1264, pages 1–11, Aarhus, Denmark, 1997. Springer-Verlag, Berlin.

[Pap94] C.H Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[Pla94] W. Plandowski. Testing equivalence of morphisms on context free languages. In *Proceedings of ESA*, 1994.

[Ryt99] Wojciech Rytter. Algorithms on compressed strings and arrays. In *Conference on Current Trends in Theory and Practice of Informatics*, pages 48–65, 1999.

[Sch78] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proc. 10th ACM Symposium on Theory of Computing (STOC)*, 1978.