A Processus analysés avec π-*sa*

On présente dans cette appendice quelques exemples de processus analysés avec succes par π-*sa*. Pour chaque exemple, les résultats viennent directement de l'analyseur et aucun calcul n'a été effectué à la main.

A.1 Les processus alternés

A.1.1 Fichier source

```
A :=*a?[x](x![a] + c?[u]d![u]);
B :=*b?[x](x![b] + c![e]);
C :=a![b];

(A|B|C);;
```

A.1.2 Accessibilité

$$
\begin{array}{l}
(\nu \mathrm{a})(\nu \mathrm{b})(\nu \mathrm{c})(\nu \mathrm{d})(\nu \mathrm{e})\\
\quad (*\mathrm{a}?^1[x](x!^2[\mathrm{a}] + \mathrm{c}?^3[u]\cancel{\mathrm{d}!^4[u]})\\
\quad |\, *\mathrm{b}?^5[x](x!^6[\mathrm{a}] + \mathrm{c}!^7[\mathrm{e}])\\
\quad |\, \mathrm{a}!^8[\mathrm{b}])
\end{array}
$$

A.1.3 Compteur de processus

On note $\sharp(i)$ le nombre d'instances du processus P tel que i soit l'étiquette du premier symbole de P, au cours des différentes étapes d'exécution.

π-*sa* calcule notamment les contraintes suivantes :

$$
\begin{cases}
\sharp(1) = \sharp(5) = 1\\
\sharp(4) = 0\\
\sharp(i) \in [|0;1|],\ \forall i \in \{2;3;6;7;8\}\\
\displaystyle\sum_{i\in\{2;3;6;7;8\}} \sharp(i) = 1
\end{cases}
$$

π-*sa* a détecté une exclusion mutuelle entre le sous-processus $(\mathrm{c}?^3[u]\mathrm{d}!^4[u])$ et le sous-processus $(\mathrm{c}!^7[\mathrm{e}])$.

Cette contrainte suffit à montrer que ces deux sous-processus ne peuvent pas communiquer entre eux. Elle est de plus trouvée très rapidement à l'issue de la deuxième itération.

A.2 Le serveur trois ports

A.2.1 Fichier source

((*make?[](♯socket)(♯out)(♯infoclient)
((socket?[in](out![in]|make![]))
|socket![infoclient]))
| make![]
| make![]
| make![]
| make?[]make?[]make?[]make?[]test![]);;

A.2.2 Accessibilité

$$
\begin{array}{l}
(\nu \mathrm{make})(\nu \mathrm{test}) \\
\quad ((*\mathrm{make}?^{1}[](\sharp \mathit{socket})(\sharp \mathit{out})(\sharp \mathit{infoclient}) \\
\qquad\qquad (\mathit{socket}?^{2}[\mathit{in}](\mathit{out}!^{3}[\mathit{in}]|\mathrm{make}!^{4}[])) \\
\qquad\qquad |\mathit{socket}!^{5}[\mathit{infoclient}])) \\
\quad |\ \mathrm{make}!^{6}[] \\
\quad |\ \mathrm{make}!^{7}[] \\
\quad |\ \mathrm{make}!^{8}[] \\
\quad |\ \mathrm{make}?^{9}[]\mathrm{make}?^{10}[]\mathrm{make}?^{11}[]\mathrm{make}?^{12}[]\cancel{\mathrm{test}!^{13}[]})
\end{array}
$$

π-*sa* a permis de détecter que le sous-processus (test!^{13}) est inaccessible. Ainsi, il ne peut jamais y avoir simultanément plus de trois *message* (make![]) au cours d'une exécution.

A.2.3 Compteur de processus

On note $\sharp(i)$ le nombre d'instances du processus P étiqueté i, au cours des différentes étapes d'exécution.

π-*sa* calcule entre autres les contraintes suivantes :

$$
\begin{cases}
\sharp(1) = 1 \\
\sharp(13) = 0 \\
\sharp(i) \in [|0;3|],\ \forall i \in \{2;3;4;5\} \\
\sharp i \in [|0;1|],\ \forall i \in [|6;12|] \\
\sharp(4) + \sharp(6) + \sharp(7) + \sharp(8) = 3 - \sharp(2) - \sharp(10) - 2 \times \sharp(11) - 3 \times \sharp(12)
\end{cases}
$$

La contrainte d'exclusion mutuelle détectée par π-*sa*, permet de déduire par réduction que le sous-processus étiqueté 13 est inaccessible. En effet, celui-ci n'est accessible qu'à partir d'une configuration qui contiendrait un *message* (make![]) et un sous processus étiqueté 12.

Or $\sharp(12) \geqslant 1 \implies \sharp(4) + \sharp(6) + \sharp(7) + \sharp(8) \leqslant 0$.
Puis $\sharp(12) \geqslant 1 \implies \sharp(4) = \sharp(6) = \sharp(7) = \sharp(8) = 0$.

A.2.4 Sécurité

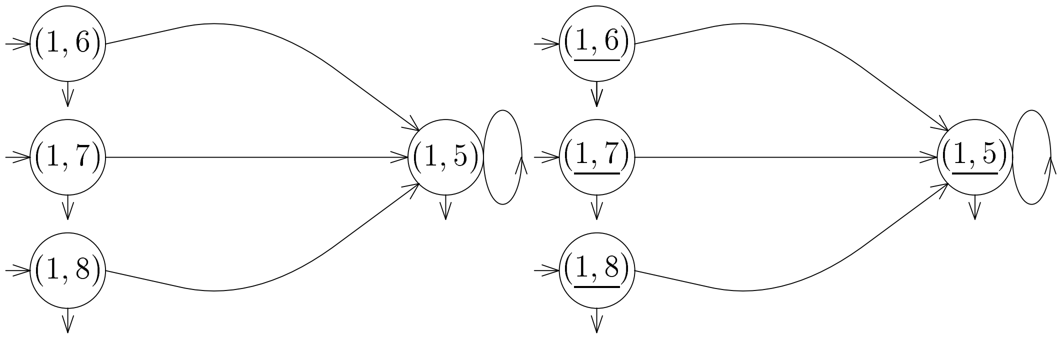
π-*sa* prouve aussi la sécurité du protocole, en montrant que l'information émise par le client ne peut être renvoyée que sur son propre canal de sortie.

Pour cela, on examine les environnements que l'on peut associer au processus 3 :

$$\begin{cases} (3, \mathit{out}, \mathit{out}) & \mapsto \left((g, \underline{g}),\ \left\{ \lambda = \underline{\lambda},\ \forall \lambda \in \Sigma \right. \right) \\ (3, \mathit{in}, \mathit{infoclient}) & \mapsto \left((g, \underline{g}),\ \left\{ \lambda = \underline{\lambda},\ \forall \lambda \in \Sigma \right. \right) \end{cases}$$

où g et $\underline{g}$ sont données figure A.1

Figure A.1 graphes des identifiants



Ces contraintes suffisent à montrer que dans une tâche de la forme $(\mathit{out}![\mathit{in}], \mathit{id})$, l'identifiant du canal *out* auquel est affecté la variable *out*, et l'identifiant du canal *infoclient* auquel est affecté la variable *in*, sont tous deux égaux à *id*.

Ainsi, lorsqu'un *message infoclient* est émis sur un canal *out*, ces deux canaux ont nécessairement été créés par la même réplication du serveur. Ils appartiennent donc au même client.

A.3 Le serveur erroné

A.3.1 Fichier source

$$
\begin{array}{l}
((\text{*make?}[](\sharp\text{socket})(\sharp\text{out})(\sharp\text{infoclient}) \\
\qquad\qquad\qquad (\text{socket?}[\text{in}]\text{out!}[\text{in}] \\
\qquad\qquad\qquad\ |\text{make!}[] \\
\qquad\qquad\qquad\ |\text{socket!}[\text{infoclient}])) \\
|\ \text{make!}[] \\
|\ \text{make!}[] \\
|\ \text{make!}[])
\end{array}
$$

A.3.2 Accessibilité

$$
\begin{array}{l}
(\nu\text{make})(\nu\text{test}) \\
\quad ((\text{*make?}^1[](\sharp \mathit{socket})(\sharp \mathit{out})(\sharp \mathit{infoclient}) \\
\qquad\qquad\qquad (\mathit{socket}?^2[\mathit{in}]\mathit{out}!^3[\mathit{in}] \\
\qquad\qquad\qquad\ |\mathit{socket}!^4[\mathit{infoclient}])) \\
\qquad\qquad\qquad\ |\text{make!}^5[] \\
\quad |\ \text{make!}^6[] \\
\quad |\ \text{make!}^7[] \\
\quad |\ \text{make!}^8[])
\end{array}
$$

A.3.3 Compteur de processus

On considère une étape d'exécution, on note $\sharp(i)$ le nombre d'instances du processus P étiqueté i et $\underline{\sharp}(\lambda)$ le nombre de transitions étiquetée λ, effectuées à partir d'une configuration initiale pour atteindre cette étape.

π-*sa* calcule entre autres les contraintes suivantes :

$$
\begin{cases}
\sharp(1) = 1 \\
\sharp(13) = 0 \\
\sharp(i) \in [|0;\infty|[,\ \forall i \in \{2;3;4;5\} \\
\sharp i \in [|0;1|],\ \forall i \in [|6;12|] \\
\sharp(2) = \underline{\sharp}(1,5) + \underline{\sharp}(1,6) + \underline{\sharp}(1,7) + \underline{\sharp}(1,8) - \sharp(3)
\end{cases}
$$

π-*sa* signale qu'il ne peut majorer le nombre d'instances simultanées du sous-processus étiqueté par 2. Ce n'est bien sûr qu'un avertissement, puisque l'analyse effectue une approximation supérieure de la sémantique collectrice concrète.

π-*sa* a donc su faire la différence entre un protocole réaliste qui n'utilise qu'un nombre limité de ressources physiques, et un protocole qui en utilise un nombre arbitraire.

Par ailleurs, π-*sa* donne une piste pour trouver la source de l'erreur potentielle, sous la forme d'une contrainte linéaire. Celle-ci exprime les liens de causes à effets entre le nombre de sous-processus présents et celui des transitions effectuées.

A.3.4 Sécurité

π-*sa* prouve la sécurité du protocole. En montrant que l'information émise par le client ne peut être réémise que sur son propre canal de sortie.

Les contraintes calculées sur les identifiants sont les mêmes que pour le protocole du serveur 3 ports (Cf A.2.4).

A.4 L'anneau de communication

A.4.1 Fichier source

```
init        :=make![left0];
link(x,y)   :=edge![x,y];
close(x)    :=edge![x,first];
call(x)     :=make![x];

(*make?[last](♯next)
            ( link(last,next)
            | call(next)
            | [last=next]test1![])
| *make?[last]close(last)
| init
| *edge?[x,y][x=y]test2![x]);;
```

A.4.2 Accessibilité

$$
\begin{array}{l}
(\nu\text{make})(\nu\text{edge})(\nu\text{first})(\nu\text{test1})(\nu\text{test2})\\
\quad (\text{*make?}^{1}[\mathit{last}](\nu\,\mathit{next})\\
\qquad\qquad\qquad (\text{edge!}^{2}[\mathit{last},\mathit{next}]\\
\qquad\qquad\qquad |\text{make!}^{3}[\mathit{left}]\\
\qquad\qquad\qquad |[\mathit{last}=^{4}\mathit{next}]\cancel{\text{test1!}^{5}[]})\\
\quad |\text{*make?}^{6}[\mathit{last}](\text{edge!}^{7}[\mathit{last},\text{first}])\\
\quad |\text{make!}^{8}[\text{first}]\\
\quad |\text{*edge?}^{9}[x,y][x=^{10}y]\ \text{test2!}^{11}[x]
\end{array}
$$

A.4.3 Compteurs de processus

On note $\sharp(i)$ le nombre d'instances du processus P étiqueté par i soit l'étiquette du premier symbole de P, au cours des différentes étapes d'exécution.

π-*sa* calcule entre autres les contraintes suivantes :

$$
\begin{cases}
\sharp(1) = \sharp(6) = \sharp(9) = 1\\
\sharp(5) = 0\\
\sharp(i) \in [|0;1|],\ \forall i \in \{3;4;7;8\}\\
\sharp(i) \in [|0;\infty|[,\ \forall i \in [|2;10;11|]\\
\underline{\sharp}(9,2) + \sharp(2) = \underline{\sharp}(1,8) + \underline{\sharp}(1,3)\\
\underline{\sharp}(9,7)\sharp(3) + \sharp(7) + \sharp(8) = 1
\end{cases}
$$

Remarque A.4.1 *L'analyse n'est pas aussi précise, lorsque l'on ne change pas le paramètre n du domaine* $\mathfrak{V}$*, pour la troisième itération. En effet, si l'on garde* $n = 0$*, la réduction n'est pas assez fine pour contrebalancer l'imprécision provoquée par l'opérateur d'élargissement, et l'analyseur trouve alors la contrainte* $\sharp(7) \in [|0;2|]$*, au lieu de la contrainte* $\sharp(7) \in [|0;1|]$*.*

A.4.4 Sécurité

π-*sa* prouve la sécurité du protocole, en montrant que chaque sommet est soit connecté au sommet initial, soit connecté à son successeur.

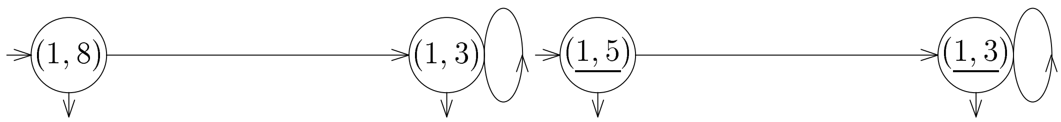
Pour cela, on examine les environnements que l'on peut associer aux processus 2 et 7.

On traite par exemple le cas où la connection est assurée par le sous-processus (edge!^{2}[*last*,*next*]) et que le canal first n'est pas affecté à la variable *last*.

$$\begin{cases}(2, last, next) & \mapsto \left((g,\underline{g}), \left\{(1,3) = (\underline{1,3}) + 1\right.\right) \\ (2, next, next) & \mapsto \left((g,\underline{g}), \left\{(1,3) = (\underline{1,3})\right.\right)\end{cases}$$

où g et $\underline{g}$ *sont données figure A.2*

Figure A.2 graphes des identifiants



Ces contraintes suffisent à montrer que dans une tâche de la forme (edge![*last*,*next*],*id*), lorsque le canal first n'est pas affecté à la variable *last*, l'identifiant du canal *next* auquel est affecté la variable *last* est obtenu en concaténant la lettre $(1,3)$ à *id*, alors que l'identifiant du canal *next* auquel est affecté la variable *next*, est égal à *id*. Ainsi, la variable à laquelle est affecté canal *next* a été créée par la réplication de ressource qui a succédé à celle qui avait créé la variable à laquelle est affecté le canal *last*

A.4.5 Topologie du réseau

On sait désormais, que chaque sommet est soit connecté à son successeur, soit connecté au sommet initial.

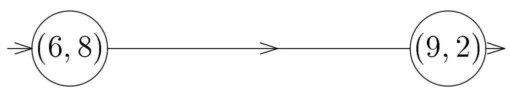
Par ailleurs, π-*sa* nous informe que seul le sommet initial peut éventuellement peut être connecté à lui même.

Pour cela, on examine tous les environnements que l'on peut associer aux processus 11.

$$\begin{cases} (11, test2, test2) & \mapsto \left((g', \varepsilon_1), \begin{cases} (6,8) = 1 \\ (9,2) = 1 \end{cases} \right) \\ (11, x, first) & \mapsto \left((g', \varepsilon_1), \begin{cases} (6,8) = 1 \\ (9,2) = 1 \end{cases} \right) \end{cases}$$

où g' est donné figure A.3

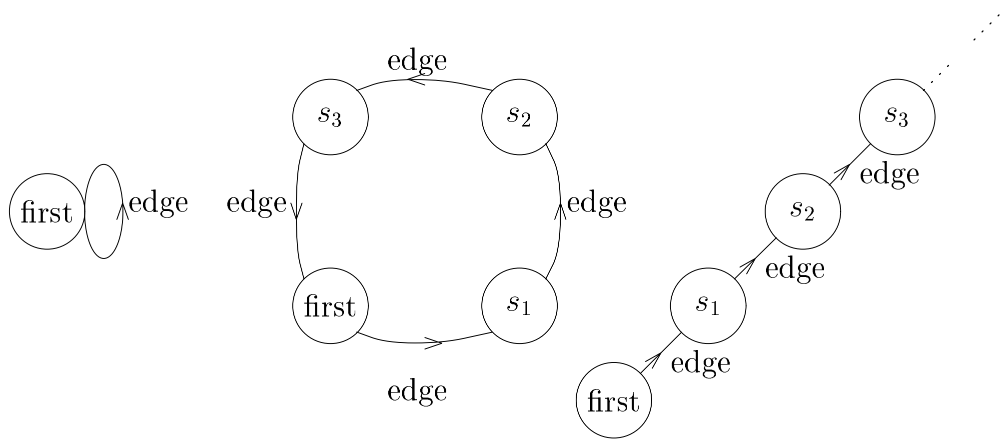
Figure A.3 graphe de l'identifiants



D'autre part, π-*sa* nous avertit qu'il se peut que des dérivations infinies existent. En effet, il ne peut trouver de borne au nombre de transitions étiquetés $(1, 3)$ que peut comporter une exécution. Cette information est obtenue par le compteur de tâches : $(\sharp(2) \in [|0; \infty|[)$.

On décrit alors figure A.4 les trois formes que peut prendre un réseau de communication formé au cours d'une exécution du processus **ring**.

Figure A.4 topologies des réseaux engendrés par le processus **ring**



B Prévisions

B.1 Un terme ouvert

B.1.1 Fichier source

```
    P:=( a?[]b?[]test![]
       | a?[]b![]
       | a![]) ;

(*make[]((♯ a)(♯ b)P | make![])
| make![])
```

B.1.2 Accessibilité

$$
\begin{array}{l}
(\nu \mathrm{make})(\nu \mathrm{test}) \\
\quad (*\mathrm{make}?^1[](\nu a)(\nu b) \\
\qquad\qquad (a?^2[]b?^3[]\cancel{\mathrm{test}!^4[]} \\
\qquad\qquad |a?^5[]b!^6[] \\
\qquad\qquad |a!^7[] \\
\qquad\qquad |\mathrm{make}![]) \\
\quad |\mathrm{make}![])
\end{array}
$$

L'analyse ouverte du sous-processus **P** indépendamment de son contexte permet de déduire que le sous-processus (test!^{4}[]) est inaccessible.

En effet, l'analyseur détecte que seuls les canaux (make) et (test) sont publiques. Ainsi, le contexte du sous-processus **P** ne peut interférer avec celui-ci ni sur le canal a, ni sur le canal b. Puis, l'analyse des exclusions mutuelles montre que sous ces conditions, le sous-processus (test!^{4}[]) est inaccessible.