Vérification automatique de protocoles de sécurité avec mémoire globale et boucles

Itsaka Rakotonirina sous la direction de Steve Kremer
INRIA Nancy Grand Est - EPC Cassis

02 juin 2014 - 11 juillet 2014

Résumé

Dans le cadre d'échange sécurisé de données, les protocoles cryptographiques mis en œuvre ont souvent besoin de manipuler une forme de registre permettant de conserver et mettre à jour des informations communes à plusieurs processus exécutés en parallèle ; en bref, une mémoire globale. Bien qu'il existe des outils de vérification automatique de protocoles, comme ProVerif pour ne citer qu'un des plus connus, peu permettent la modélisation d'une telle forme de mémoire. Une des exceptions, Tamarin, utilise un langage d'input en lequel a été traduit un dérivé du pi-calcul appliqué, Sapic, permettant ainsi une spécification haut niveau des protocoles. Le but de ce stage était d'étendre le calcul Sapic ainsi que sa traduction en lui ajoutant un constructeur de boucles.

1 Introduction

La conception algorithmique de cryptosystèmes est centrale dans la sécurité informatique, mais leur utilisation dans des protocoles sûrs est tout aussi importante. La vérification automatique de ces protocoles s'est largement développée dans la dernière décennie, donnant jour à de nombreux outils de cette spécificité. Néanmoins, la plupart proposent un formalisme inefficace pour représenter plusieurs acteurs en parallèle modifiant un état global commun, comme ProVerif [2] qui reste néanmoins un des vérificateurs automatiques les plus utilisés. Tamarin de S. Meier *et al.* [4] est une des rares exceptions, mais son langage d'input utilise des règles de réécriture de multi-ensembles (ou msr, de l'anglais *multiset rewriting rule*), formalisme très bas niveau loin de l'intuition qu'on se donne d'un protocole cryptographique. S. Kremer *et al.* [3] proposent pour cela un langage dérivé du pi-calcul, Sapic, et une traduction (démontrée correcte) en un jeu de msr afin d'exprimer les protocoles dans une syntaxe plus intuitive. Cela n'apporte pas d'expressivité supplémentaire mais a son utilité en pratique : la manipulation des msr étant un exercice fastidieux, les protocoles sont souvent simplifiés pour être représentés en Tamarin et on a ainsi inconsciemment tendance à exclure des potentielles attaques.

Le but de ce papier est de compléter [3] en apportant à Sapic un constructeur de boucles (sémantique à définir) afin de représenter plus facilement des protocoles comme TESLA [1]. En section 2, on présente quelques prérequis techniques et terminologiques, tous tirés de [3] et [4]. Une syntaxe et une sémantique des boucles est ensuite proposée en section 3, suivie de l'extension de la traduction de Sapic et les principaux éléments de la démonstration de sa correction vis-à-vis de la sémantique en section 4 (rédaction complète en annexe B). Enfin, les sources de Sapic ayant été complétées au cours de ce stage, quelques exemples de performances de l'implémentation du calcul étendu sont proposés en section 5.

2 Concepts préliminaires

Cette section donne quelques éléments pour la compréhension de Tamarin : les définitions de base sont en section 2.1 et la section 2.2 décrit la structure de Tamarin, à savoir la syntaxe de son langage d'input, sa sémantique et son formalisme pour représenter les protocoles de sécurité. Cela reprend les premières sections de [3], résumant lui-même le début de la seconde partie de [4].

2.1 Quelques définitions

2.1.1 Termes et faits

• On représente les messages échangés lors d'un protocole par des termes typés. Le type des messages sera noté msg, et on se donne deux sous-types, $fresh$ pour les secrets du protocole comme les clés secrètes de chiffrement ou les messages, et pub pour les constantes du protocole comme les noms des acteurs ou des registres. On leur associe chacun un ensemble infini et dénombrable et noms, respectivement FN et PN. On note $u : s$ pour dire qu'une variable u est de type s et on notera $\mathcal{V}_s$ l'ensemble (infini et dénombrable) des variables de type s, et $\mathcal{V}$ la réunion de tous les $\mathcal{V}_s$. Pour construire les termes, on fixe Σ un ensemble fini de fonctions avec arité qu'on appelle signature, f/n représentant $f \in \Sigma$ d'arité n, et Σ^n le sous-ensemble de Σ des fonctions d'arité n. $\mathcal{T}$ dénote alors l'ensemble des termes construits à partir des fonctions de Σ et des atomes de PN, FN et $\mathcal{V}$, et pour un terme t, $vars(t)$ (resp. $names(t)$) l'ensemble des variables (resp. noms) apparaissant dans t.

• La signature modélise les mécanismes fonctionnels du protocole : par exemple $\Sigma = \{enc/2,\ dec/2\}$ peut représenter le chiffrement symétrique, $enc(m, k)$ symbolisant le message m chiffré avec la clé k et $dec(c, k)$ le chiffré c déchiffré avec la clé k. Dans cet exemple, on veut aussi s'assurer que $dec(enc(m,\ k),\ k) = m$ pour tous termes m et k : c'est le rôle de la théorie équationnelle E, un ensemble d'égalité entre termes à travers le spectre duquel sont considérées toutes les égalités et opérations ensemblistes (on notera par exemple $=_E$, $\in_E$...). Il y a quelques restrictions, comme le fait que l'ensemble des variables du membre de gauche d'une égalité de E doit être strictement inclus dans celui du membre de droite : justifications et détails ne sont pas nécessaires à la compréhension de ce papier mais [4] peut être consulté pour plus de précisions.

• On dispose d'une autre signature, Σ_{fact}, disjointe de Σ, qu'on paritionne en deux ensembles, appelés faits linéaires et faits permanents, et on note $\mathcal{F} = \{F(t_1, \cdots, t_k) |\ \forall\ i \in [\![1; k]\!],\ t_i \in \mathcal{T},\ F \in \Sigma^k_{fact}\}$ l'ensemble des faits. Ils représentent l'état de la mémoire pendant l'exécution d'un protocole Tamarin : comme détaillé en section 2.2, un protocole maintient un multi-ensemble de faits initialement vide, et en appliquant des msr, consomme certains de ces faits et en produit de nouveaux. Les faits linéaires sont retirés du multi-ensemble mémoire après avoir été utilisés pour une msr, au contraire des faits permanents. On se donne de plus d'un fait linéaire `Fr` (pour les déclarations de variables fraîches) ainsi qu'un fait permanent `!K` (symbolisant les connaissances de l'attaquant). Enfin, on dit qu'un fait ou un terme est clos s'il ne contient pas de variables : on note $\mathcal{G}$ (resp. $\mathcal{M}$) l'ensemble des faits (resp. termes) clos, et pour un fait f, $ginsts(f)$ l'ensemble des instances closes de f (cette dernière notation est étendue intuitivement aux ensembles et séquences, définies au paragraphe suivant).

2.1.2 Quelques autres notations

• Une substitution $\sigma : \mathcal{V} \to \mathcal{T}$ est une fonction partielle telle que $\sigma(x)$ soit un terme de type s (ou un sous-type de s) pour tout $x : s$. On note $\sigma = \left\{{}^{t_1}/_{x_1}, \cdots, {}^{t_n}/_{x_n}\right\}$ la substitution de domaine $\mathbf{D}(\sigma) = \{x_1, \cdots, x_n\}$ et qui associe t_i à x_i pour tout $i \in [\![1; n]\!]$. On les étend homomorphiquement aux termes, faits, mais aussi aux ensembles, et on utilise une notation postfixe ($t\sigma$ signifie $\sigma(t)$). De plus, on convient que $g(x) = \bot$ quand $g(x)$ n'est pas défini, autrement dit quand $x \notin \mathbf{D}(g)$; on écrira

de plus $g[a \mapsto b]$ pour représenter la fonction de domaine $\mathbf{D}(g) \cup \{a\}$ qui à $x \neq a$ associe $g(x)$ et qui à a associe b.

- Pour un ensemble S, S^* (resp. $S^\#$) dénote l'ensemble des suites finies (resp. multi-ensembles finis) à éléments dans S.La suite d'éléments $e_1, \cdots, e_n$ sera quant à elle notée $[e_1, \cdots, e_n]$ et on utilise l'opérateur binaire $\cdot$ pour la concaténation d'un élément à une suite, que ce soit à la fin ou au début de cette dernière. On utilisera l'exposant $\#$ pour les opérations usuelles sur les multi-ensembles, par exemple $\subset^\#$, $\cup^\#$ ou $\setminus^\#$. On abusera aussi parfois des notations en considérant une suite finie comme un ensemble, ou un ensemble comme un multi-ensemble.

2.2 Tamarin

On définit les règles de réécriture de multi-ensemble (msr), langage d'input de Tamarin, permettant de manipuler un multi-ensemble de faits qui représentera l'état d'un protocole.

Définition 2.1 (Règle de réécriture de multi-ensemble ou msr)**.** *Une règle de réécriture de multi-ensemble ri est un triplet $(l, a, r) \in (\mathcal{F}^*)^3$, qu'on note l–[a]→r. On appelle $l = prems(ri)$ les prémices de la règle, $a = acts(ri)$ ses actions, et $r = concls(ri)$ ses conclusions.*

Définition 2.2 (Système de règles de réécriture)**.** *Un système de règles de réécriture R est un ensemble de règles de réécriture tel que tout l–[a]→$r \in R$ vérifie :*

- *l, a, et r ne contiennent pas de noms frais.*
- *r ne contient pas de fait* `Fr`*, sauf si $r =$*FRESH*, où* FRESH=[]–[]→[`Fr`(x :*fresh*)].*

De plus, on dit que R est bien formé si on a aussi :

- *pour tout l'–[a']→$r' \in_E ginsts(l$–[a]→$r)$, on a $\bigcap_{r''=_E r'} names(r'') \cap FN \subseteq \bigcap_{l''=_E l'} names(l'') \cap FN$.*

On peut à présent définir la relation de transition associée à un protocole (sous forme de système de réécriture) qui fait office de sémantique des msr.

Définition 2.3 (relation de transition)**.** *Soit R un système de réécriture de multi-ensemble. On définit $\rightarrow_R$ la relation de transition associée à R, partie de $\mathcal{G}^\# \times \mathcal{P} \times \mathcal{G}^\#$, par $S \xrightarrow{a}_R ((S \setminus^\# lfacts(l)) \cup^\# r)$ si l–[a]→$r \in ginsts(R \cup$FRESH$)$, $lfacts(l) \subseteq^\# S$ et $pfacts(l) \subseteq^\# S$.*

Définition 2.4 (Exécution msr)**.** *Soit R un système de règles de réécriture bien formé. On définit l'ensemble de ses exécutions $exec^{msr}(R)$ par :*

$$exec^{msr}(R) = \Big\{ \varnothing \xrightarrow{A_1}_R \cdots \xrightarrow{A_n}_R S_n \mid \forall\, a \in FN,\ \forall\, (i,j) \in [\![1; n-1]\!]^2, \big((i \neq j \ \wedge\ S_{i+1} \setminus^\# S_i = \{\mathtt{Fr}(a)\}) \ \Rightarrow\ (S_{j+1} \setminus^\# S_j \neq \{\mathtt{Fr}(a)\})\big) \Big\}$$

Une exécution msr est donc une suite d'applications de msr qui respecte l'unicité des noms frais. Pour exprimer des propriétés sur les protocoles, on se sert des actions étiquetant les msr utilisées, ce qui se formalise avec la définition suivante :

Définition 2.5 (Trace msr)**.** *Avec les mêmes notations que la définition précédente, on définit $traces^{msr}(R)$ l'ensemble de ses traces par :*

$$traces^{msr}(R) = \Big\{ [A_1, \cdots, A_n] \mid \forall\, i \in [\![0; n]\!],\ A_i \neq \varnothing \text{ et } \varnothing \xRightarrow{A_1}_R \cdots \xRightarrow{A_n}_R S_n \in exec^{msr}(R) \Big\}$$

où $\overset{A}{\Longrightarrow}_R$ *est défini par* $\xrightarrow{\varnothing}{}^*_R \xrightarrow{A}_R \xrightarrow{\varnothing}{}^*_R$ *avec des notations intuitives pour la composition.*

Les traces permettent d'exprimer des propriétés du premier ordre dans un formalisme qu'on va définir. De plus, on utilise un nouveau type, *temp* représentant les marqueurs temporels (matérialisant quand une action a lieu dans une trace). $\mathcal{V}_{temp}$ est donc l'ensemble des variables temporelles.

Définition 2.6 (Formule sur les traces)**.** *On appelle un atome de trace une égalité entre termes* $t_1 \approx t_2$*, une comparaison de marqueurs temporels* $i \lessdot j$*, une égalité entre marqueurs temporels* $i \doteq j$*, une action* $F@i$ *pour un fait* $F \in \mathcal{F}$*, un marqueur temporel* i *ou le faux* $\bot$*. Une formule de trace est une formule du premier ordre construite à partir d'atomes de trace.*

Afin de faire le lien entre les formules et ce qui précède, on attribue à chaque type s un domaine $\mathbf{D}(s)$: $\mathbf{D}(temp) = \mathbb{Q}$, $\mathbf{D}(msg) = \mathcal{M}$, $\mathbf{D}(fresh) = FN$ et $\mathbf{D}(pub) = PN$. On dit qu'une fonction $\theta : \mathcal{V} \to \mathcal{M} \cup \mathbb{Q}$ est une valuation si elle respecte ces types, autrement dit si pour tout type s on a $\theta(\mathcal{V}_s) \subseteq \mathbf{D}(s)$. On notera $t\theta$, l'application de l'extension homomorphique de θ au terme t.

Définition 2.7 (Relation de satisfaction)**.** *Soit* $tr = [A_1, \cdots, A_n] \in \mathcal{G}^*$ *une trace (au sens de la définition 2.5 mais aussi de la 3.3 plus loin en section 3.2), une valuation* θ *et une formule de trace* φ*. La satisfaction de* φ *par* (tr, θ)*, notée* $(tr, \theta) \vDash \varphi$*, est définie par induction sur la structure de* φ *:*

$$
\begin{array}{ll}
(tr,\theta) \vDash \bot & \textit{jamais} \\
(tr,\theta) \vDash F@i & \textbf{\textit{ssi}}\ \ \theta(i) \in [\![1;n]\!] \textit{ et } F\theta \in_E A_{\theta(i)} \\
(tr,\theta) \vDash i \lessdot j & \textbf{\textit{ssi}}\ \ \theta(i) < \theta(j) \\
(tr,\theta) \vDash i \doteq j & \textbf{\textit{ssi}}\ \ \theta(i) = \theta(j) \\
(tr,\theta) \vDash t_1 \approx t_2 & \textbf{\textit{ssi}}\ \ t_1\theta =_E t_2\theta \\
(tr,\theta) \vDash \neg\psi & \textbf{\textit{ssi}}\ \ (tr,\theta) \nvDash \psi \\
(tr,\theta) \vDash \varphi_1 \wedge \varphi_2 & \textbf{\textit{ssi}}\ (tr,\theta) \vDash \varphi_1 \textit{ et } (tr,\theta) \vDash \varphi_2 \\
(tr,\theta) \vDash \exists x : s.\psi & \textbf{\textit{ssi}}\ \ \textit{il existe } u \in \mathbf{D}(s) \textit{ tel que } (tr, \theta[x \mapsto u]) \vDash \psi
\end{array}
$$

Les autres opérateurs usuels sont définis par dualité ou avec les formules de De Morgan : $\varphi_1 \vee \varphi_2$ est défini comme $\neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \Rightarrow \varphi_2$ comme $\neg\varphi_1 \vee \varphi_2$, et $\forall x : s.\psi$ comme $\neg(\exists x : s.\neg\psi)$. Intuitvement, tr représente l'exécution dont on veut étudier la propriété φ, et θ fixe les variables et ordonne les actions dans le temps. De plus, on se permet d'omettre θ si φ est une formule sans variable libre puisque la satisfaction ne dépend alors pas de la valuation choisie. On définit ensuite la validité et la satisfaisabilité des formules :

Définition 2.8 (Validité et satisfaisabilité)**.** *Soit* $Tr \subseteq (\mathcal{G}^*)^*$ *un ensemble de traces (au sens de la définition 2.5 mais aussi 3.3 plus loin en section 3.2) et* φ *une formule de trace. On dit que* φ *est valide pour* Tr*, ce qu'on note* $Tr \vDash^\forall \varphi$ *si pour toute trace* $tr \in Tr$ *et toute valuation* θ *on a* $(tr, \theta) \vDash \varphi$*. On dit que* φ *est satisfiable pour* Tr*, ce qu'on note* $Tr \vDash^\exists \varphi$*, s'il exsiste une trace* $tr \in Tr$ *et une valuation* θ *tels que* $(tr, \theta) \vDash \varphi$*.*

Remarque 2.1. *On se permettra de noter* $R \vDash^\star \varphi$*, avec* $\star$ *le symbole* $\forall$ *ou* $\exists$*, et* R *un système de règles de réécriture, pour signifier que* $traces^{msr}(R) \vDash^\star \varphi$*. On remarquera l'analogie avec les traces de protocoles plus loin à la remarque 3.5.*

3 Syntaxe et sémantique de Sapic (avec les boucles)

On étudie à présent Sapic, un dérivé du pi-calcul appliqué dont on a dit qu'il existait une traduction en msr, proposée par S. Kremer *et al.* [3] et démontrée correcte. En section 3.1 on propose une vue d'ensemble de la syntaxe du langage (étendue avec les boucles) ainsi qu'une sémantique informelle, puis on précise celle des boucles en section 3.2. La sémantique complète est en annexe A.1.

3.1 Syntaxe et sémantique informelle

On définit la nouvelle syntaxe du langage ; elle étend celle de [3] en y ajoutant un constructeur $*$ pour l'itération. La grammaire est décrite en figure 1.

$$
\begin{array}{l}
\langle M, N \rangle \ ::= x, y, z \in \mathcal{V} \\
\quad \mid \ (M) \\
\quad \mid \ p \in PN \\
\quad \mid \ n \in FN \\
\quad \mid \ f(M_1, \cdots, M_n) \quad (\text{ où } f \in \Sigma^n) \\
\\
\langle P, Q \rangle \ ::= 0 \\
\quad \mid \ (P) \\
\quad \mid \ P|Q \\
\quad \mid \ !P \\
\quad \mid \ *P \quad \text{où } P \text{ ne contient ni réplication (!) ni itération (*)} \\
\quad \mid \ \nu n;\ P \\
\quad \mid \ \text{out}(M, N);\ P \\
\quad \mid \ \text{in}(M, N);\ P \\
\quad \mid \ \text{if } M = N \text{ then } P \text{ else } Q \\
\quad \mid \ \text{event } F;\ P \\
\quad \mid \ \text{insert } M, N;\ P \\
\quad \mid \ \text{delete } M;\ P \\
\quad \mid \ \text{lookup } M \text{ as } v \text{ in } P \text{ else } Q \\
\quad \mid \ \text{lock } M;\ P \\
\quad \mid \ \text{unlock } M;\ P \\
\quad \mid \ [L] \text{ –}[A]\rightarrow [R];\ P \qquad (L, R, A \in \mathcal{F}^*)
\end{array}
$$

FIGURE 1 – Syntaxe Sapic avec $*$

- | permet d'exécuter deux processus parallèlement : dans $P|Q$, les deux protocoles P et Q sont ainsi indépendants et exécutés chacun de leur côté. En particulier ils ne partagent pas les variables qu'ils introduisent (avec ν, `in` ou `lookup`).
- ! suit la même idée, à la différence près que le même protocole P est dupliqué un nombre arbitraire de fois et que ce sont ses copies qui sont exécutées indépendamment et en parallèle.
- 0 est le protocole trivial, terminant toute branche de l'arbre syntaxique d'un protocole.
- νa permet de déclarer un nom frais a pour le processus en cours.
- `event F` représente une action qui apparaîtra dans la trace des exécutions parvenant jusqu'à cette instruction.
- `insert` M, N et `delete` M permettent de gérer la mémoire globale associée à l'identifiant M. Des `insert` successifs permettent de modifier la valeur associée à un identifiant.
- `lock` M et `unlock` M garantissent l'exclusivité d'accès à une ressource M : si un seul processus à la fois parmi d'autres en parallèle doit pouvoir accéder à M, chacun essaie d'en obtenir l'accès par

l'instruction `lock` M, ce qui échoue si la ressource est déjà sous l'effet d'un `lock`, auquel cas on doit attendre qu'un `unlock` M soit réalisé pour pouvoir retenter le `lock` à son tour.

- `in`(M, N) et `out`(M, N) modélisent les échanges de messages N sur un canal M : `in` est la réception d'un message (avec filtrage de motif) et `out` son dépôt. On notera que la sémantique est synchrone : l'exécution est bloquée à un `in` tant qu'aucun message n'est reçu, tout comme elle ne se poursuit pas après un `out` tant que le message sous-jacent n'a pas été receptionné par quelqu'un (cela inclut un potentiel attaquant à qui ne serait pas destiné l'envoi).
- $[L]$ –$[A]$→ $[R]$ est une msr de Tamarin dont l'application est forcée au moment où elle est placée dans le protocole.
- `if` $M = N$ `then` P `else` Q correspond à l'exécution du protocole P si les termes M et N sont égaux modulo E, et à celle de Q sinon. Q est traité comme 0 en cas d'omission de la branche `else`.
- `lookup` M `as` v `in` P `else` Q regarde dans la mémoire globale (modifiée par les `insert` et `delete`) à l'identifiant M : si un terme y a été inséré, la variable v y sera liée dans la suite de l'exécution qui sera P, et si l'emplacement mémoire est vide, l'exécution se poursuit avec Q. On notera que `lookup` ne réalise pas de filtrage de motif et ne permet pas de redéfinir des variables (il faut donc α-renommer soit-même).
- L'itérateur $*$ répète un nombre arbitraire de fois un protocole P donné. Toutes les exécutions sont indépendantes : en particulier, les variables introduites dans P ne sont pas maintenues d'une itération sur l'autre.

Remarque 3.1. *Un protocole itéré par une étoile doit terminer clairement pour pouvoir le relancer, d'où l'interdiction des* `!` *et* `*` *sous les autres* `*`*. Une traduction permettant les boucles imbriquées a en fait précédé la version actuelle mais produisait un système de msr trop complexe pour être analysé par Tamarin sans lemme additionnel ; de plus, l'utilité des boucles imbriquées n'était pas flagrante.*

Exemple 3.1. *On donne un exemple de protocole modélisé avec la syntaxe de Sapic, avec la signature* $\Sigma = \{enc/2, dec/2, dec\_success/2, true/0\}$ *et la théorie équationnelle* $E = \{dec(enc(m,k),k) = m, dec\_success(enc(m,k),k) = true()\}$*. enc et dec décrivent le chiffrement symétrique comme en section 2.1.1 et dec_success est utilisé pour tester l'effectivité du chiffrement.*

```
ν k ;
*(
      ( ν m ; event Send(m) ; out(enc(m, k)) ; 0 )
      |
      ( in(c) ; if dec_success(c,k)=true() then (event Got(dec(c, k)) ; 0) )
 )
```

où on se permet d'omettre les canaux des **in** *et* **out** *pour signifier que les messages sont échangés sur un canal public, auquel l'attaquant a donc accès et sur lequel il peut publier des messages. On modélise alors une propriété de ce protocole selon les définitions 2.6 et 2.7 :*

$$\forall\, x : msg,\ i : temp.\ \mathrm{Got}(x)@i \ \Rightarrow\ (\exists\, j : temp.\ j \lessdot i \ \wedge\ \mathrm{Send}(x)@j)$$

serait une formule de trace représentant une propriété d'authenticité (tout message accepté par le second acteur a été envoyé par le premier et pas par l'attaquant). Sans s'attarder sur la démonstration qu'on peut laisser à Tamarin, cette formule est valide pour ce protocole.

3.2 Sémantique des protocoles

On doit définir le pouvoir déductif de l'attaquant car les actions de ce dernier font partie intégrante de l'exécution d'un protocole, puisqu'il peut aller jusqu'à publier des messages sur des canaux connus. Sa capacité de déduction $\nu\widetilde{n}.\sigma$ est définie par les règles d'inférence de la figure 2, où $\widetilde{n}$ est un ensemble de noms frais qui sont les secrets du protocole (que l'attaquant ne peut pas connaître a priori) et σ est une substitution matérialisant les informations connues de l'attaquant.

$$\frac{a \in FN \cup PN \quad a \notin \widetilde{n}}{\nu\widetilde{n}.\sigma \vdash a}\text{DName} \qquad \frac{\nu\widetilde{n}.\sigma \vdash t \quad t =_E t'}{\nu\widetilde{n}.\sigma \vdash t'}\text{DEq}$$

$$\frac{x \in \mathbf{D}(\sigma)}{\nu\widetilde{n}.\sigma \vdash x\sigma}\text{DFrame} \qquad \frac{\nu\widetilde{n}.\sigma \vdash t_1 \ \cdots \ \nu\widetilde{n}.\sigma \vdash t_n \quad f \in \Sigma^k}{\nu\widetilde{n}.\sigma \vdash f(t_1, \cdots, t_n)}\text{DAppl}$$

Figure 2 – Règles de déduction de l'attaquant

On donne à présent une sémantique opérationnelle aux boucles. On appelle une configuration (*configuration process* dans [3]) un 8-uplet $(\mathcal{E}, \mathcal{S}, \mathcal{S}^{MS}, \mathcal{P}, \mathcal{R}, \mathcal{I}, \sigma, \mathcal{L})$, où :

- $\mathcal{E} \subseteq FN$ représente les noms frais générés par le protocole (tous threads confondus).
- $\mathcal{S} : \mathcal{M} \to \mathcal{M}$ représente la mémoire globale (mise à jour par les `insert` et `delete`).
- $\mathcal{S}^{MS} \in \mathcal{G}^{\#}$ représente les faits qui ont été produits par les différentes applications de l'instruction $[L]\ –[A]\rightarrow\ [R]$ (et pas par les applications de msr générées par la traduction).
- $\mathcal{P}$, un multi-ensemble de processus clos, représente les protocoles exécutés en parallèle, travaillant de pair avec $\mathcal{R}$ et $\mathcal{I}$.
- $\mathcal{R}$ complète $\mathcal{P}$ et est le multi-ensemble de processus clos pouvant être répliqués : ainsi, dans la sémantique, un processus dans $\mathcal{R}$ peut produire arbitrairement une copie de lui-même dans $\mathcal{P}$. On notera que cet ensemble n'est pas présent dans le formalisme de [3] (voir remarque 3.3).
- $\mathcal{I}$ est une fonction partielle représentant les processus susceptibles d'être itérés (donc l'analogue de $\mathcal{R}$ pour l'itération). Elle associe à un nom id le processus P itéré par la boucle identifiée par id, ainsi que le nombre n de processus en parallèle encore en cours d'exécution dans P.
- σ, une substitution close, représente les messages échangés que l'attaquant a intercepté ; cela permet de modéliser la connaissance de ce dernier à un stade de l'exécution.
- $\mathcal{L} \subseteq \mathcal{M}$ représente les `lock` en cours de validité.

Remarque 3.2. *On annote les processus par des termes clos une absence d'annotations étant comprise comme un indice 0 telle que $\mathcal{I}(0)$ jamais défini. Ainsi on trouvera par la suite la notation P_{id} représentant le processus P annoté par id.*

Définition 3.1 (Sémantique des protocoles)**.** *La sémantique opérationnelle des protocoles est définie par l'ensemble de règles de la figure 7 en annexe A.1, contenant les règles données dans [3] (adaptées au nouveau formalisme) auxquelles on a ajouté :*

$$(\mathcal{E}, \mathcal{S}, \mathcal{S}^{MS}, \mathcal{P} \cup^{\#} \{*P\}, \mathcal{R}, \mathcal{I}, \sigma, \mathcal{L}) \ \to \ (\mathcal{E}, \mathcal{S}, \mathcal{S}^{MS}, \mathcal{P} \cup^{\#} \{P_{id}\}, \mathcal{R}, \mathcal{I}[id \mapsto (P, 1)], \sigma, \mathcal{L})$$
où id est un nom frais

$$(\mathcal{E}, \mathcal{S}, \mathcal{S}^{MS}, \mathcal{P}, \mathcal{R}, \mathcal{I}, \sigma, \mathcal{L}) \ \to \ (\mathcal{E}, \mathcal{S}, \mathcal{S}^{MS}, \mathcal{P} \cup^{\#} \{P_{id}\}, \mathcal{R}, \mathcal{I}[id \mapsto (P, 1)], \sigma, \mathcal{L})$$
si $id \in \mathbf{D}(\mathcal{I})$ et $\mathcal{I}(id) = (P, 0)$

$$(\mathcal{E}, \mathcal{S}, \mathcal{S}^{MS}, \mathcal{P} \cup^{\#} \{0_{id}\}, \mathcal{R}, \mathcal{I}, \sigma, \mathcal{L}) \ \to \ (\mathcal{E}, \mathcal{S}, \mathcal{S}^{MS}, \mathcal{P}, \mathcal{R}, \mathcal{I}[id \mapsto (P, n-1)], \sigma, \mathcal{L})$$
si $id \in \mathbf{D}(\mathcal{I})$ et $\mathcal{I}(id) = (P, n)$ avec $n > 0$

La première règle permet d'initialiser la boucle en lui associant un identifiant. La deuxième permet de lancer une itération : le $(P, 1)$ signifie qu'on itère un processus P et qu'on doit attendre la fin d'un processus en parallèle avant de pouvoir relancer une fois de plus la boucle (cet entier est incrémenté par la règle traitant la mise en parallèle, visible en annexe A.1). Enfin, la troisième règle comptabilise les fins de processus parallèles en question. On note $\rightarrow^*$ la clôture réflexive et transitive de $\rightarrow$. Pour les transitions étiquetées (voir annexe A.1), on note $C \overset{F}{\Longrightarrow} C'$ pour un fait F et deux configurations C et C' s'il existe deux configurations C_1 et C_2 telles que $C \xrightarrow{\varnothing}{}^* C_1 \xrightarrow{F} C_2 \xrightarrow{\varnothing}{}^* C'$.

Remarque 3.3. *Même pour des processus ne contenant pas de boucles, le formalisme des configurations ne reprend pas exactement celui de [3] bien que la sémantique intuitive sous-jacente soit exactement la même. Le centre du problème est la correction de la traduction de la section 4.2 par rapport à la sémantique : informellement, sa démonstration par induction dans [3] utilise un invariant qui imposerait pour être étendu aux boucles que $*P$ soit maintenu dans $\mathcal{P}$ au cours des différentes transitions ce qui forcerait à se passer du confort de la fonction $\mathcal{I}$.*

Remarque 3.4. *Vu la remarque précédente, les tests de performance et la démonstration de correction consignés dans [3] ne tiennent pas pour ce nouveau formalisme. Miraculeusement, il se trouve qu'il y avait une erreur dans l'implémentation des réplications (visible sur la figure 8 de [3]) et qu'elle correspond justement à la modification qu'on veut introduire : seule la démonstration de correction sera donc à adapter. D'une certaine manière, on étend bien la traduction de Sapic, en rectifiant au passage les définitions qui ne suivaient pas l'implémentation.*

De manière analogue à la représentation des protocoles en Tamarin, on peut alors définir ce qu'est une exécution et une trace d'un protocole Sapic.

Définition 3.2 (Exécution d'un protocole Sapic)**.** *Une exécution d'un protocole P est une suite d'applications de la relation $\rightarrow$ de la sémantique opérationnelle à partir de la configuration initiale $(\varnothing, \varnothing, \varnothing, \{P\}, \varnothing, \varnothing, \varnothing, \varnothing)$ en considérant qu'une fonction à domaine vide est représentée par $\varnothing$. Plus formellement, c'est une suite de configurations $(\mathcal{E}_i, \mathcal{S}_i, \mathcal{S}_i^{MS}, \mathcal{P}_i, \mathcal{R}_i, \mathcal{I}_i, \sigma_i, \mathcal{L}_i)$, $i \in [\![0; n]\!]$ telle que :*

$$\begin{cases} (\mathcal{E}_0, \mathcal{S}_0, \mathcal{S}_0^{MS}, \mathcal{P}_0, \mathcal{R}_0, \mathcal{I}_0, \sigma_0, \mathcal{L}_0) = (\varnothing, \varnothing, \varnothing, \{P\}, \varnothing, \varnothing, \varnothing, \varnothing) \\ \forall\ i \in [\![0; n-1]\!],\ (\mathcal{E}_i, \mathcal{S}_i, \mathcal{S}_i^{MS}, \mathcal{P}_i, \mathcal{R}_i, \mathcal{I}_i, \sigma_i, \mathcal{L}_i) \rightarrow (\mathcal{E}_{i+1}, \mathcal{S}_{i+1}, \mathcal{S}_{i+1}^{MS}, \mathcal{P}_{i+1}, \mathcal{R}_{i+1}, \mathcal{I}_{i+1}, \sigma_{i+1}, \mathcal{L}_{i+1}) \end{cases}$$

On appellera $exec^{pi}(P)$ l'ensemble des exécutions possibles du protocole P.

Définition 3.3 (Trace d'un protocole Sapic)**.** *Analoguement à la définition 2.5, on définit l'ensemble des traces d'un protocole par :*

$$traces^{pi}(P) = \left\{ [F_1, \cdots, F_n],\ C_0 \overset{F_1}{\Longrightarrow} \cdots \overset{F_n}{\Longrightarrow} C_n,\ (C_i)_{i \in [\![0;n]\!]} \in exec^{pi}(P) \right\}$$

Remarque 3.5. *On se permettra de noter $P \vDash^\star \varphi$, avec $\star$ le symbole $\forall$ ou $\exists$, et P un protocole, pour signifier que $traces^{pi}(P) \vDash^\star \varphi$. On remarquera l'analogie avec les traces msr à la remarque 2.1 et que la correction de la traduction repose sur le lien entre les deux concepts de trace.*

Exemple 3.2. *On reprend le protocole de l'exemple 3.1 pour illustrer les définitions ci-dessus, avec* $P = \nu\ m$ *;* **event** *Send(m)* *;* **out**(c) *;* $\mathbf{0}$ *et* $Q =$ **in**(c) *;* **if** *dec_ sucess(c,k)=true()* **then** (**event** *Got(dec(c, k))* *;* $\mathbf{0}$*). Une exécution possible de ce protocole serait alors :*

$$
\begin{aligned}
& (\varnothing, \varnothing, \varnothing, \{\nu\ k; *(P|Q)\}, \varnothing, \varnothing, \varnothing, \varnothing) \\
\rightarrow\ & (\{k\}, \varnothing, \varnothing, \{*(P|Q)\}, \varnothing, \varnothing, \varnothing, \varnothing) \\
\rightarrow\ & (\{k\}, \varnothing, \varnothing, \{(P|Q)_{id_1}\}, \varnothing, \mathcal{I}_1, \varnothing, \varnothing) \\
\rightarrow\ & (\{k\}, \varnothing, \varnothing, \{P_{id_1}, Q_{id_1}\}, \varnothing, \mathcal{I}_2, \varnothing, \varnothing) \\
\rightarrow\ & (\{k, m\}, \varnothing, \varnothing, \{(\textbf{event}\ \textit{Send(m)}\ ;\ \textbf{out}(\textit{enc(m, k)})\ ;\ \mathbf{0})_{id_1},\ Q_{id_1}\}, \varnothing, \mathcal{I}_2, \varnothing, \varnothing) \\
\xrightarrow{\text{Send}(m)}\ & (\{k, m\}, \varnothing, \varnothing, \{(\textbf{out}(\textit{enc(m, k)})\ ;\ \mathbf{0})_{id_1},\ Q_{id_1}\}, \varnothing, \mathcal{I}_2, \varnothing, \varnothing)
\end{aligned}
$$

$$
\text{où } \mathcal{I}_n(id) = \begin{cases} (P|Q, n) & \text{si } id = id_1 \\ \bot & \text{sinon} \end{cases}
$$

4 Extension de la traduction

On étend à présent la traduction de Sapic aux boucles. En section 4.1 est décrite la précédente traduction, puis le cas des boucles est traité en section 4.2. Enfin, la démonstration de correction de cette traduction par rapport à la sémantique des processus est étudiée en section 4.4.

4.1 Prérequis sur la traduction de Sapic

- On traduit les processus en trois parties (définitions précises en section 4.2 et annexe A.2) :

  - Un ensemble de msr MD modélisant la capacité de déduction de l'attaquant de la figure 2 en section 3.2, mais aussi la capacité de l'attaquant à produire des messages.
  - Des règles modélisant le corps de l'exécution : l'ensemble $\mathcal{P}$ (notation de la section 3.2) est modélisé par un ensemble de faits $\texttt{state}_p(\widetilde{x})$, où p est la position dans P (le processus initial) de l'instruction en cours d'exécution, et $\widetilde{x}$ l'ensemble des noms introduits depuis le début de l'exécution.
  - Une règle Init pour produire le premier `state`, à savoir $\texttt{state}_{[]}()$.

- On traduit également les formules de trace : pour $[\![P]\!]$ la traduction d'un protocole P et $[\![\varphi]\!]$ celle d'une formule φ, on veut faire en sorte que $P \vDash \varphi$ **ssi** $[\![P]\!] \vDash [\![\varphi]\!]$. On remarque l'abus de notation puisqu'on a seulement défini $\vDash^{\forall}$ et $\vDash^{\exists}$; de même, la définition $[\![\varphi]\!]$ dépendra de si on étudie la validité ou la satisfaisabilité de φ. La traduction précise des formules est donnée en section 4.3.

- Enfin, on impose quelques restrictions sur les protocoles traduisibles, qui couvrent néanmoins toujours tous les cas pratiques :

Définition 4.1 (Variables et faits réservés)**.** *On définit l'ensemble des variables reservées comme l'ensemble des* n_a*,* $a \in FN$*, et des* $lock_l$*,* $l \in \mathbb{N}$*. L'ensemble des fait réservés,* $\mathcal{F}_{res}$*, est l'ensemble des* $f(t_1, \cdots, t_n)$*, avec les* t_i *termes et* $f \in \Sigma^n_{fact}$ *parmi Init, Insert, Delete, IsIn, IsNotSet, state, exit, memo, Lock, Unlock, Out, Fr, In, Msg, ProtoNonce, Eq, NotEq, Event, InEvent, Reboot ou Finish.*

Définition 4.2 (Protocole bien formé)**.** *Un protocole clos est dit bien formé si :*

- *P ne contient ni variable réservée ni fait reservé.*
- *tout nom dans P est lié au plus une fois.*
- *le système de* `lock` *est cohérent, ce qui se définit formellement par le fait que $\overline{P}$ ne contienne pas $\bot$, où $\overline{P}$ est introduit en annexe à la définition A.1.*
- *toute instruction de P de la forme l–$[a]$→ r doit être bien formée au sens de la définition 2.2.*

On dit également qu'une formule de trace φ est bien formée si φ ne contient aucune variable reservée ou fait reservé.

4.2 Traduction de l'itération en msr

Traduire $*P$ en msr revient à traduire P, à cela près qu'on doit pouvoir relancer P après une itération, la difficulté étant justement de matérialiser la fin de l'exécution de P. On doit aussi faire abstraction de la terminaison d'un protocole quand on traite celle d'une de ses éventuelles copies. On produit pour cela un fait linéaire $\texttt{exit}_p(id)$ qui signifie que le protocole identifié par id et commençant à la position p a terminé son exécution. Par exemple pour $*(P|P)$, les deux P sont bien différenciés, puisqu'ils produisent respectivement $\texttt{exit}_{[11]}(id)$ et $\texttt{exit}_{[12]}(id)$ à la fin de leur d'exécution (id fixé préalablement). De même, toutes les copies de P dans $!*P$ se distinguent si on génère un identifiant différent à chaque nouvelle réplication de $*P$, chacune produisant alors un $\texttt{exit}_{[11]}(id)$ avec son id propre. On formalise cela avec la définition de E_{id} en figure 3 : par exemple pour produire $\texttt{exit}_{[]}(id)$ à partir de $P|Q$, il suffit d'avoir produit $\texttt{exit}_{[1]}(id)$ et $\texttt{exit}_{[2]}(id)$, puisque l'exécution à partir de la position [] (soit $P|Q$) est terminée quand celles aux positions [1] (soit P) et [2] (soit Q) sont terminées, ce qui se traduit par la règle $[\texttt{exit}_{[1]}(id), \texttt{exit}_{[2]}(id)] \to [\texttt{exit}_{[]}(id)]$.

$$
\begin{array}{rcl}
E_{id}(0,p,\widetilde{x}) & = & \{[\texttt{state}_p(\widetilde{x})] \to [\texttt{exit}_p(id)]\} \\
E_{id}(P|Q,p,\widetilde{x}) & = & \{[\texttt{exit}_{p\cdot 1}(id), \texttt{exit}_{p\cdot 2}(id)] \to [\texttt{exit}_p(id)]\} \\
 & & \cup\ E_{id}(P,p\cdot 1,\widetilde{x})\ \cup\ E_{id}(Q,p\cdot 2,\widetilde{x}) \\
E_{id}(!P,p,\widetilde{x}) & = & E_{id}(*P,p,\widetilde{x})\ =\ \bot \\
E_{id}(\text{if } M=N \text{ then } P & = & E_{id}(P,p\cdot 1,\widetilde{x})\left\{{}^{\texttt{exit}_p(id)}/_{\texttt{exit}_{p\cdot 1}(id)}\right\} \\
\text{else } Q,p,\widetilde{x}) & & \cup\ E_{id}(Q,p\cdot 2,\widetilde{x})\left\{{}^{\texttt{exit}_p(id)}/_{\texttt{exit}_{p\cdot 2}(id)}\right\} \\
E_{id}(\text{lookup } M \text{ as } v & = & E_{id}(P,p\cdot 1,\widetilde{x}\cup\{v\})\left\{{}^{\texttt{exit}_p(id)}/_{\texttt{exit}_{p\cdot 1}(id)}\right\} \\
\text{in } P \text{ else } Q,p,\widetilde{x}) & & \cup\ E_{id}(Q,p\cdot 2,\widetilde{x})\left\{{}^{\texttt{exit}_p(id)}/_{\texttt{exit}_{p\cdot 2}(id)}\right\} \\
E_{id}(instr;\ P,p,\widetilde{x}) & = & E_{id}(P,p\cdot 1,\widetilde{y})\left\{{}^{\texttt{exit}_p(id)}/_{\texttt{exit}_{p\cdot 1}(id)}\right\} \\
 & \text{où} & \widetilde{y} = \widetilde{x}\cup\{n_a : fresh\} \text{ si } instr = \nu a \\
 & & \widetilde{y} = \widetilde{x}\cup vars(N) \text{ si } instr = in(M,N) \\
 & & \widetilde{y} = \widetilde{x}\cup lock_l \text{ si } instr = lock^l s \\
 & & \widetilde{y} = \widetilde{x}\cup vars(l) \text{ si } instr = [l\ \text{–}[a]\!\to\ r] \\
 & & \widetilde{y} = \widetilde{x} \text{ sinon}
\end{array}
$$

FIGURE 3 – Définition de E_{id}

Remarque 4.1. *Les substitutions dans la définition sont des optimisations pour remplacer des chaînes $[\texttt{exit}_{p\cdot 1\cdot 1}(id)] \to [\texttt{exit}_{p\cdot 1}(id)] \to [\texttt{exit}_p(id)]$ par le raccourci $[\texttt{exit}_{p\cdot 1\cdot 1}(id)] \to [\texttt{exit}_p(id)]$. Cela ne change pas la sémantique, mais réduit le temps que passe ensuite Tamarin sur les démonstrations.*

Exemple 4.1. *On peut reprendre l'exemple 3.1 pour illustrer l'utilisation de E_{id}. Comme en pratique E_{id} ne sera calculé que sous une étoile, on donne $E_{id}(P,[111],\{n_k,id\})$ où P est le protocole en question (cela correspond à l'état de l'exécution juste après la lecture de l'étoile) :*

$$\begin{array}{rcl}
[\mathtt{state}_{[111111]}(n_k, id, n_m)] & \to & [\mathtt{exit}_{[111]}(id)] \\
[\mathtt{state}_{[11212]}(n_k, id, c)] & \to & [\mathtt{exit}_{[112]}(id)] \\
[\mathtt{state}_{[112111]}(n_k, id, c)] & \to & [\mathtt{exit}_{[112]}(id)] \\
[\mathtt{exit}_{[111]}(id), \mathtt{exit}_{[112]}(id)] & \to & [\mathtt{exit}_{[11]}(id)]
\end{array}$$

On définit alors la traduction de $*P$: il suffit d'une règle pour commencer une itération (et lui associer un identifiant) et de traduire P en remplaçant les fins par la structure donnée par E_{id} :

Définition 4.3. *On complète la définition de $[\![\ ]\!]$ de [3] pour des protocoles définis par la syntaxe de la figure 1 (définition complète en annexe A.2 figure 8) avec :*

$$[\![*P, p, \widetilde{x}]\!] = \{boot,\ finish\} \ \cup\ Looped_{id}(P, p \cdot 1, \widetilde{x})$$

où :
— $boot = [\mathtt{state}_p(\widetilde{x}), \mathtt{Fr}(id)] \to [!\mathtt{state}_{p\cdot 1}(\widetilde{x}, id)]$
— $finish = [\mathtt{exit}_{p\cdot 1}(id)] -\!\!/ Finish(id) /\!\!\to []$
— $Looped_{id}(P, p \cdot 1, \widetilde{x})$ *est l'ensemble :*

$$\left\{ \ell - [a] \to r \ \in\ [\![P, p \cdot 1, \widetilde{x} \cup \{id : fresh\}]\!] \mid r \neq [] \right\} \ \cup\ E_{id}(P, p \cdot 1, \widetilde{x} \cup \{id\})$$

où chaque règle de la forme :

$$[\mathtt{state}_{p\cdot 1}(\widetilde{t}, id) \cdot \ell] - [a] \to r$$

est remplacée par :

$$[!\mathtt{state}_{p\cdot 1}(\widetilde{t}, id) \cdot \ell] - [a,\ Reboot(id)] \to r$$

Remarque 4.2. *La traduction ci-dessus peut sembler de même effet que celle des réplications, mais la traduction des formules de trace a aussi été étendue : le nouvel axiome α_{loop} (voir section 4.3) permet de se restreindre aux exécutions où deux* Reboot(id) *sont toujours séparés d'un* Finish(id).

Définition 4.4 (Traduction des processus)**.** *Soit P un processus clos bien formé. On définit le système de règles de réécriture $[\![P]\!]$ comme* MD$\cup$Init$\cup$ $[\![\overline{P}, [], []]\!]$*, où :*

- Init$= [] -\!\!/$ Init $() /\!\!\to [\mathtt{state}_{[]}()]$
- *MD est défini en figure 4*
- $[\![P, p, \widetilde{x}]\!]$ *est défini inductivement pour un protocole P, p une position de P et $\widetilde{x}$ un ensemble de variables, dans la définition 4.3, ou de manière plus complète en annexe A.2 figure 8.*

$\mathtt{Out}(x)$	$-[\ ]\to$	$!\mathtt{K}(x)$	MDOut
$!\mathtt{K}(x)$	$-[K(x)]\to$	$\mathtt{In}(x)$	MDIn
	$-[\ ]\to$	$!\mathtt{K}(x : pub)$	MDPub
$\mathtt{Fr}(x : fresh)$	$-[\ ]\to$	$!\mathtt{K}(x : fresh)$	MDFresh
$!\mathtt{K}(x_1), \cdots, !\mathtt{K}(x_n)$	$-[\ ]\to$	$!\mathtt{K}(f(x_1, \cdots, x_n))$ pour $f \in \Sigma^k$	MDOut

Figure 4 – Les règles MD

Exemple 4.2. *On peut reprendre l'exemple 3.1 pour faire une mise en application, en calculant* $[\![P, [], []]\!]$. *Le nombre de règles rend néanmoins le résultat fastidieux à lire et comprendre : on se contentera des premières règles, jusqu'à la mise en parallèle (les suivantes ont soit déjà été énoncées dans l'exemple 4.1, soit peuvent être générée par celles de la précédente traduction [3]).*

$$\begin{array}{rcl}
[\mathtt{state}_{[]}(),\ \mathtt{Fr}(n_k)] & \text{–/ProtoNonce}(n_k)/\!\rightarrow & [\mathtt{state}_{[1]}(n_k)] \\
[\mathtt{state}_{[1]}(n_k),\ \mathtt{Fr}(id)] & \rightarrow & [!\mathtt{state}_{[11]}(n_k, id)] \\
[\mathtt{exit}_{[11]}(id)](n_k, id) & \text{–/Finish}(id)/\!\rightarrow & [] \\
[!\mathtt{state}_{[11]}(n_k, id)] & \text{–/Reboot}(id)/\!\rightarrow & [\mathtt{state}_{[111]}(n_k, id),\ \mathtt{state}_{[112]}(n_k, id)]
\end{array}$$

4.3 Traduction des formules de trace

Pour terminer, on traduit également les formules de trace : on restreint ainsi les applications des msr de $[\![P]\!]$ à certaines conditions pour que cela génère le même ensemble de traces (aux faits réservés près) qu'avec la sémantique opérationnelle.

Définition 4.5. *Soit* φ *une formule de trace bien formée. On définit :*

$$[\![\varphi]\!]_\forall = \alpha \Rightarrow \varphi \qquad \textit{et} \qquad [\![\varphi]\!]_\exists = \alpha \wedge \varphi$$

où la définition de α*, identique à celle de [3] au membre* α_{loop} *près, est en figure 5.*

$\alpha = \alpha_{init} \wedge \alpha_{eq} \wedge \alpha_{noteq} \wedge \alpha_{in} \wedge \alpha_{notin} \wedge \alpha_{lock} \wedge \alpha_{inev} \wedge \alpha_{loop}$

$$\begin{array}{rll}
\alpha_{init} & = \forall i, j. & (\text{Init}()@i \wedge \text{Init}()@j) \Rightarrow i \doteq j \\
\alpha_{eq} & = \forall x, y, i. & \text{Eq}(x, y)@i \Rightarrow x \approx y \\
\alpha_{noteq} & = \forall x, y, i. & \text{NotEq}(x, y)@i \Rightarrow \neg(x \approx y) \\
\alpha_{in} & = \forall x, y, t_3. & \text{IsIn}(x, y)@t_3 \Rightarrow \exists t_2.\text{Insert}(x, y)@t_2 \wedge t_2 \lessdot t_3 \\
 & & \qquad \wedge \forall t_1, y. \quad \text{Insert}(x, y)@t_1 \Rightarrow (t_1 \lessdot t_2 \vee t_1 \doteq t_2 \vee t_3 \lessdot t_1) \\
 & & \qquad \wedge \forall t_1. \quad \text{Delete}(x)@t_1 \Rightarrow (t_1 \lessdot t_2 \vee t_3 \lessdot t_1) \\
\alpha_{notin} & = \forall x, y, t_3. & \text{IsNotSet}(x)@t_3 \Rightarrow (\forall t_1, y.\ \text{Insert}(x, y)@t_1 \Rightarrow t_3 \lessdot t_1) \vee \\
 & & \qquad (\exists t_1.\ \text{Delete}(x)@t_1 \wedge t_1 \lessdot t_3 \\
 & & \qquad\quad \wedge \forall t_2, y.\ (\text{Insert}(x, y)@t_2 \wedge t_2 \lessdot t_3) \Rightarrow t_2 \lessdot t_1) \\
\alpha_{lock} & = \forall x, l, l', i, j. & \text{Lock}(l, x)@i \wedge \text{Lock}(l', x)@j \wedge i \lessdot j \\
 & & \qquad \Rightarrow \exists k.\ \text{Unlock}(l, x)@k \wedge i \lessdot k \wedge k \lessdot j \\
 & & \qquad \wedge (\forall l'', m.\ \text{Lock}(l'', x)@m \Rightarrow \neg(i \lessdot m \wedge m \lessdot k)) \\
 & & \qquad \wedge (\forall l'', m.\ \text{Unlock}(l'', x)@m \Rightarrow \neg(i \lessdot m \wedge m \lessdot k)) \\
\alpha_{inev} & = \forall t, i. & \text{InEvent}(t)@i \Rightarrow \exists j.\ \text{K}(t)@j \wedge (\forall k.\ \text{Event}()@k \Rightarrow (k \lessdot j \vee i \lessdot k)) \\
 & & \qquad \wedge (\forall k, t'.\ \text{K}(t')@k \Rightarrow (k \lessdot j \vee i \lessdot k)) \\
 & & \qquad \wedge (\forall k, t'.\text{K}(t')@k \Rightarrow (k \lessdot j \vee i \lessdot k \vee k \approx j)) \\
\alpha_{loop} & = \forall id, i. & \text{Reboot}(id)@i \Rightarrow \forall j.\ \text{Reboot}(id)@j \Rightarrow \\
 & & \qquad (i \lessdot j \Rightarrow \exists k.\ \text{Finish}(id)@k \wedge i \lessdot k \wedge k \lessdot j)
\end{array}$$

FIGURE 5 – Définition de α

On ne s'étendra pas sur l'explication de la définition des membres de la conjonction α qui reste assez intuitive, au moins au vu de la sémantique opérationnelle. Plus de détails peuvent être trouvés directement dans [3].

Remarque 4.3. *Les formules de trace doivent être sous une forme particulière pour pouvoir être étudiées par Tamarin : S. Meier les appelle des guarded trace formula dans sa thèse [4]. Cela n'induit pas de restriction majeure et n'intéressera que le lecteur voulant s'essayer à coder en Sapic ou Tamarin.*

4.4 Correction de la traduction

[3] fera norme pour la numérotation des résultats de cette section. Puisque ce papier le complète, on peut conserver sa démonstration de correction en ajoutant le cas des boucles dans les preuves inductives (aux changements de formalisme près). On suppose que le lecteur intéressé par les détails fera une lecture parallèle des deux documents et on ne consignera ici que le résultat principal et ceux dont la démonstration doit être modifiée, lesdites démonstrations étant laissées en annexe.

Théorème 1 (Correction de la traduction)**.** *Soit P un processus clos bien formé et φ une formule de trace bien formée. On a alors : $P \vDash^\star \varphi$ **ssi** $[\![P]\!] \vDash^\star [\![\varphi]\!]_\star$ où $\star$ est le symbole $\forall$ ou $\exists$.*

La démonstration de ce théorème repose principalement sur un lemme dont la démonstration est à adapter par rapport à [3], tant à cause de l'ajout des boucles que par le changement de formalisme (on pensera à la modification de la traduction des réplications) :

Lemme 1. *Soit P un processus clos bien formé. On a $traces^{pi}(P) = hide(filter(traces^{msr}([\![P]\!])))$ où $hide$ et $filter$ sont définis par :*

- *Pour Tr ensemble de traces : $hide(Tr) = \{hide(tr),\ tr \in Tr\}$*
- *Pour tr trace : $hide(tr) = \begin{cases} [] & si\ tr = [] \\ hide(tr') & si\ tr = F \cdot tr'\ avec\ F \subseteq \mathcal{F}_{res} \\ (F \backslash \mathcal{F}_{res}) \cdot hide(tr') & sinon,\ avec\ tr = F \cdot tr' \end{cases}$*
- *Pour Tr ensemble de traces : $filter(Tr) = \{tr \in Tr,\ \forall \theta.\ (tr, \theta) \vDash \alpha\}$ où α est défini à la figure 5.*

Ce lemme signifie que pourvu qu'on fasse abstraction des mots réservés générés par les exécutions msr de la traduction d'un protocole et qu'on ne considère que les exécutions qui vérifient les prérequis de α, la traduction et le protocole initial ont les mêmes propriétés. Une fois ce lemme démontré, il suffit de quelques propositions (détaillées dans [3]) pour aboutir théorème 1. On peut conserver sans aucun changement les démonstrations de tous ces résultats, exception faite du lemme 1 : l'annexe B ne se concentre donc que sur ce dernier.

5 Performances

À propos du protocole TESLA

TESLA est un protocole d'authentification de flux (stream authentification) introduit par J.D. Tygar *et al.* [1]. Deux acteurs, un envoyeur et un receveur, s'envoient une suite arbitrairement longue de messages et doivent faire abstraction du bruit : tout message que le receveur déclare avoir authentifié doit effectivement avoir été publié par l'envoyeur (et non pas par un attaquant par exemple). On notera qu'il existe plusieurs versions de TESLA et qu'on manipule la plus simple, interdisant la perte de paquets pendant le procédé. La version plus élaborée n'est pas encore maîtrisée : que ce soit par le biais de Sapic ou directement en Tamarin par S. Meier [4], on ne parvient pas encore à démontrer la propriété d'authentification. Enfin, on notera que TESLA nécessite une synchronisation des deux acteurs, ce qui est réalisé dans [1] avec une synchronisation des montres : Tamarin n'ayant pas de notion du temps, on pourrait considérer que la modélisation en msr s'écarte un peu du protocole initial, mais l'approximation reste assez minime pour qu'elle puisse être négligée.

Performances de l'implémentation

Après avoir complété les codes source du langage, on peut remplir le but initial de cet article, à savoir travailler plus facilement avec des protocoles contenant des boucles. On ne consigne ici que deux exemples : le protocole présenté dans l'exemple 3.1 ainsi que le protocole TESLA. Les résultats sont consignés en figure 6 et complètent ceux de l'article.

Protocole	Lemmes auxiliaires	Démonstration autonome*
Exemple 3.1	0	oui ($1s$)
TESLA	0	oui ($33s$)

* Sur Intel Core 2 Duo 2.26GHz, 2Go de RAM

FIGURE 6 – Performances de Tamarin sur des protocoles Sapic traduits

Lacunes de la traduction des boucles

Cette traduction a un défaut lié à l'algorithme qu'utilise Tamarin pour démontrer les lemmes. Elle ne permet pas, sans lemme auxiliaire ou assistance manuelle, de démontrer des lemmes reposant sur la propriété « dans l'itération $*P$ de P, il est possible que P soit itéré deux fois », alors que, pour TESLA par exemple, l'implémentation Tamarin de [4] en donne une démonstration quasi immédiate. Cela n'a pas eu d'impact sur les objectifs du stage puisqu'on ne s'intéressait pas à des propriétés de cette trempe, mais cela laisse présager une faiblesse de cette traduction. Le problème est en amont : le terme de « boucle » est plutôt flou, et on peut difficilement lui donner une sémantique à la fois générale et traduite efficacement, ce qui nécessite parfois s'adapter au cas par cas.

Pour résumer : la traduction a un large domaine d'action (elle permet de modéliser de beaucoup de protocoles), répond aux attentes qu'on avait d'elle (simplifier l'utilisation du prouveur de Tamarin), mais semble avoir des lacunes par rapport à des implémentations en Tamarin « à la main » quand il s'agit de démontrer certains lemmes, même si cela n'a pas été problématique pour le moment.

6 Conclusion

On a ainsi rendu le calcul Sapic riche d'un nouveau constructeur, et étendu sa traduction en msr ainsi que la démonstration de la correction de cette dernière. Cet ajout n'a aucun effet sur l'expressivité mais a le mérite de faire un pas de plus dans la simplification de l'utilisation de Tamarin, produisant des codes qui peuvent être compris même sans avoir étudié le langage, ce qui est un défi plutôt relevé sans l'intermédiaire de Sapic.

Des améliorations sont néanmoins encore possibles, comme la définition de boucles plus générales ; on pense par exemple à une syntaxe et traduction autorisant les boucles imbriquées si une utilité venait à leur être trouvée, ou encore à faire suivre la boucle d'autres instructions (pourvu qu'on lui définisse une terminaison propre). Comme cela a été évoqué à la remarque 3.1 p. 6, les premières tentatives partaient dans ce sens, mais ont été mises de côté car Tamarin ne parvenait pas à terminer sans lemme auxiliaire avec la traduction obtenue. Générer automatiquement ces lemmes pourrait être envisagé, ou même travailler directement à améliorer l'algorithme de démonstration de Tamarin (des faiblesses sont mentionnées dans la modélisation de certaines variantes de TESLA dans [4]). Par ailleurs, cela pourrait permettre de règler dans la foulée le manque de compatibilité entre le prouveur et la traduction évoqué à la fin de la section 5.

Références

[1] J.D. Tygar Adrian Perrig, Ran Canetti and Dawn Song. The TESLA broadcast authentification protocol. *RSA CryptoBytes*, Été 2005.

[2] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pages 82–96, 2001.

[3] Steve Kremer and Künnemann Robert. Automated analysis of security protocols with global state. *Proceedings of the 35th IEEE Symposium on Security and Privacy*, Mars 2014. Sources de Sapic disponibles à : http://sapic.gforge.inria.fr/.

[4] Simon Meier. *Advancing automated security protocol verification*. PhD thesis, ETH Zurich, 2013. Sources de Tamarin disponibles à : http://www.infsec.ethz.ch/research/software/tamarin.

A Définitions annexes complètes

Définition A.1 (Annotation)**.** *Soit P un processus clos. On définit le processus annoté $\overline{P}$ par :*

$$
\begin{array}{rcl}
\overline{0} & = & 0 \\
\overline{P|Q} & = & \overline{P}|\overline{Q} \\
\overline{!P} & = & !\overline{P} \\
\overline{*P} & = & *\overline{P} \\
\overline{\text{if } t_1 = t_2 \text{ then } P \text{ else } Q} & = & \text{if } t_1 = t_2 \text{ then } \overline{P} \text{ else } \overline{Q} \\
\overline{\text{lookup } M \text{ as } x \text{ in } P \text{ else } Q} & = & \text{lookup } M \text{ as } x \text{ in } \overline{P} \text{ else } \overline{Q} \\
\overline{instr; P} & = & instr; \overline{P} \quad \textit{où } instr \textit{ ne contient pas le symbole lock ou unlock} \\
\overline{\text{lock } t; P} & = & \text{lock}^l\ t; \overline{au(P,t,l)} \quad \textit{où } l \in \mathbb{N} \textit{ est un indice frais} \\
\overline{\text{unlock}^l\ t; P} & = & \text{unlock}^l\ t; \overline{P} \\
\overline{\text{unlock } t; P} & = & \bot
\end{array}
$$

où $au(P,t,l)$ annote du label l le premier unlock avec le paramètre t rencontré, ce qui s'écrit :

$$
\begin{array}{rcl}
au(0,t,l) & = & 0 \\
au(P|Q,t,l) & = & \bot \\
au(!P,t,l) & = & \bot \\
au(*P,t,l) & = & \bot \\
au(\text{if } t_1 = t_2 \text{ then } P \text{ else } Q,t,l) & = & \text{if } t_1 = t_2 \text{ then } au(P,t,l) \text{ else } au(Q,t,l) \\
au(\text{lookup } M \text{ as } x \text{ in } P \text{ else } Q,t,l) & = & \text{lookup } M \text{ as } x \text{ in } au(P,t,l) \text{ else } au(Q,t,l) \\
au(instr; P,t,l) & = & instr; au(P,t,l) \quad \textit{où } instr \neq \text{unlock } t \\
au(\text{unlock } t; P,t,l) & = & \text{unlock}^l\ t; P
\end{array}
$$

A.1 Sémantique opérationnelle

La figure 7 donne la sémantique de tous les constructeurs Sapic selon le formalisme de la section 3.2. Elle découle de celle de [3] en s'adaptant à la nouvelle structure des configurations ; on rappelle de plus que tous les processus peuvent être considérés comme étiquetés quitte à ce que ce soit une constante par défaut (qu'on omettra parfois).

Opérations standard :

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{0_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \rightarrow (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \quad \text{si } id\notin\mathbf{D}(\mathcal{I})$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{(P|Q)_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \rightarrow (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{P_{id},Q_{id}\},\mathcal{R},\mathcal{I}',\sigma,\mathcal{L})$$
$$\text{où } \mathcal{I}'=\begin{cases}\mathcal{I}[id\mapsto(P',n+1)] & \text{si } \mathcal{I}(id)=(P',n)\\ \mathcal{I} & \text{si } id\notin\mathbf{D}(\mathcal{I})\end{cases}$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{!P\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \rightarrow (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P},\mathcal{R}\cup^{\#}\{P\},\mathcal{I},\sigma,\mathcal{L})$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{*P\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \rightarrow (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{P_{id}\},\mathcal{R},\mathcal{I}[id\mapsto(P,1)],\sigma,\mathcal{L}) \quad \text{où } id \text{ est un nom frais}$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \rightarrow (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{P_{id}\},\mathcal{R},\mathcal{I}[id\mapsto(P,1)],\sigma,\mathcal{L}) \quad \text{si } id\in\mathbf{D}(\mathcal{I}) \text{ et } \mathcal{I}(id)=(P,0)$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{0_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \rightarrow (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P},\mathcal{R},\mathcal{I}[id\mapsto(P,n-1)],\sigma,\mathcal{L}) \quad \text{si } id\in\mathbf{D}(\mathcal{I}) \text{ et } \mathcal{I}(id)=(P,n)$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P},\mathcal{R}\cup^{\#}\{P\},\mathcal{I},\sigma,\mathcal{L}) \rightarrow (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{P\},\mathcal{R}\cup^{\#}\{P\},\mathcal{I},\sigma,\mathcal{L})$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{(\nu a;\ P)_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \rightarrow (\mathcal{E}\cup\{a'\},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{P_{id}\left\{{}^{a'}/_{a}\right\}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \quad \text{où } a' \text{ est frais}$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \xrightarrow{K(M)} (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \quad \text{si } \nu\mathcal{E}.\sigma\vdash M$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{(\text{out}(M,N);\ P)_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \xrightarrow{K(M)} (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{P_{id}\},\mathcal{R},\mathcal{I},\sigma\cup\left\{{}^{N}/_{x}\right\},\mathcal{L}) \quad \text{si } x \text{ est frais et } \nu\mathcal{E}.\sigma\vdash M$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{(\text{in}(M,N);\ P)_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \xrightarrow{K(\langle M,N\tau\rangle)} (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{P_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \quad \text{si } \tau \text{ clôt } N,\ \nu\mathcal{E}.\sigma\vdash M,\ \nu\mathcal{E}.\sigma\vdash N\tau$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{(\text{out}(M,N);P)_{id_1},\ (\text{in}(M',N');Q)_{id_2}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \rightarrow (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{P_{id_1},Q_{id_2}\tau\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \quad \text{si } M=_E M' \text{ et } \tau \text{ clôt } N' \text{ avec } N=_E N'\tau$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{(\text{if } M=N \text{ then } P \text{ else } Q)_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \rightarrow (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{P_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \quad \text{si } M=_E N$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{(\text{if } M=N \text{ then } P \text{ else } Q)_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \rightarrow (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{Q_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \quad \text{si } M\neq_E N$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{(\text{event}(F);P)_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \xrightarrow{F} (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{P_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L})$$

Opérations sur la mémoire globale :

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{(\text{insert } M,N;\ P)_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \rightarrow (\mathcal{E},\mathcal{S}[M\mapsto N],\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{P_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L})$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{(\text{delete } M;P)_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \rightarrow (\mathcal{E},\mathcal{S}[M\mapsto\bot],\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{P_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L})$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{(\text{lookup } M \text{ as } x \text{ in } P \text{ else } Q)_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \rightarrow (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{P_{id}\left\{{}^{V}/_{x}\right\}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \quad \text{si } \mathcal{S}(N)=_E V \text{ est défini et } N=_E M$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{(\text{lookup } M \text{ as } x \text{ in } P \text{ else } Q)_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \rightarrow (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{Q_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \quad \text{si } \mathcal{S}(N) \text{ n'est défini pour aucun } N=_E M$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{(\text{lock } M;P)_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \rightarrow (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{P_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}\cup\{M\}) \quad \text{si } M\notin_E\mathcal{L}$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{(\text{unlock } M;P)_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \rightarrow (\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{P\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}\backslash\{M'\mid M'=_E M\})$$

$$(\mathcal{E},\mathcal{S},\mathcal{S}^{MS},\mathcal{P}\cup^{\#}\{(l\text{–}[a]\!\rightarrow\! r;\ P)_{id}\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L}) \xrightarrow{a'} (\mathcal{E},\mathcal{S},\mathcal{S}^{MS}\backslash^{\#}lfacts(l')\cup^{\#}r',\mathcal{P}\cup^{\#}\{P_{id}\tau\},\mathcal{R},\mathcal{I},\sigma,\mathcal{L})$$
$$\text{si } \tau \text{ clôt } l\text{–}[a]\!\rightarrow\! r,\ l'\text{–}[a']\!\rightarrow\! r'=_E l\text{–}[a]\!\rightarrow\! r,\ lfacts(l')\subseteq^{\#}\mathcal{S}^{MS},\ pfacts(l')\subseteq S^{MS}$$

FIGURE 7 – Sémantique opérationnelles des protocoles en Sapic

A.2 Traduction des processus en msr

La figure 8 donne la définition complète de $[\![P, p, \widetilde{x}]\!]$, avec les notations du corps du texte. Elle est identique à celle de [3] aux détails près de la réplication (corrigée pour être en accord avec l'implémentation pratique) et de l'itération (abstente dans la syntaxe de [3]).

$$
\begin{array}{rcl}
[\![0, p, \widetilde{x}]\!] & = & \{[\mathtt{state}_p(\widetilde{x})] \to []\} \\
[\![P|Q, p, \widetilde{x}]\!] & = & \{[\mathtt{state}_p(\widetilde{x})] \to [\mathtt{state}_{p\cdot1}(\widetilde{x}), \mathtt{state}_{p\cdot2}(\widetilde{x})]\} \ \cup\ [\![P, p\cdot1, \widetilde{x}]\!] \ \cup\ [\![Q, p\cdot2, \widetilde{x}]\!] \\
[\![!P, p, \widetilde{x}]\!] & = & \{[\mathtt{state}_p(\widetilde{x})] \to [!\mathtt{state}_{p\cdot1}(\widetilde{x})]\} \ \cup\ [\![P, p\cdot1, \widetilde{x}]\!] \left\{ {}^{!\mathtt{state}_{p\cdot1}(\widetilde{x})}/_{\mathtt{state}_{p\cdot1}(\widetilde{x})} \right\} \\
[\![*P, p, \widetilde{x}]\!] & = & \{[\mathtt{state}_p(\widetilde{x}), \mathtt{Fr}(id)] \to [!\mathtt{state}_{p\cdot1}(\widetilde{x}, id : fresh)], \\
 & & [\mathtt{exit}_{p\cdot1}(id)] – [Finish(id)] \to []\} \\
 & & \cup\ Reboot_{id,p\cdot1}\left([\![P, p\cdot1, \widetilde{x}]\!] \backslash Fins([\![P, p\cdot1, \widetilde{x}]\!])\right) \\
 & & \cup\ E_{id}(P, p\cdot1, \widetilde{x} \cup \{id : fresh\}) \\
[\![\nu a;\ P, p, \widetilde{x}]\!] & = & \{[\mathtt{state}_p(\widetilde{x})] – [\mathrm{ProtoNonce}(n_a : fresh)] \to [\mathtt{state}_{p\cdot1}(x, n_a : fresh)]\} \\
 & & \cup\ [\![P, p\cdot1, (\widetilde{x}, n_a)]\!] \\
[\![\mathrm{out}(M, N); P, p, \widetilde{x}]\!] & = & \{[\mathtt{state}_p(\widetilde{x}), \mathrm{In}(M)] – [\mathrm{InEvent}(M)] \to [\mathrm{Out}(N), \mathtt{state}_{p\cdot1}(\widetilde{x})], \\
 & & [\mathtt{state}_p(\widetilde{x})] \to [\mathrm{Msg}(M, N), \mathtt{state}_p^{semi}(\widetilde{x})], \\
 & & [\mathtt{state}_p^{semi}(\widetilde{x}), \mathrm{Ack}(M, N)] \to [\mathtt{state}_{p\cdot1}(\widetilde{x})]\} \ \cup\ [\![P, p\cdot1, \widetilde{x}]\!] \\
[\![\mathrm{in}(M, N); P, p, \widetilde{x}]\!] & = & \{[\mathtt{state}_p(\widetilde{x}), \mathrm{In}(\langle M, N\rangle)] – [\mathrm{InEvent}(\langle M, N\rangle)] \to [\mathtt{state}_{p\cdot1}(\widetilde{x} \cup vars(N)), \\
 & & [\mathtt{state}_p(\widetilde{x}), \mathrm{Msg}(M, N)] \to [\mathtt{state}_{p\cdot1}(\widetilde{x} \cup vars(N)), \mathrm{Ack}(M, N)]\} \\
 & & \cup\ [\![P, p\cdot1, \widetilde{x} \cup vars(N)]\!] \\
[\![\text{if } M = N \text{ then } P & = & \{[\mathtt{state}_p(\widetilde{x})] – [\mathrm{Eq}(M, N)] \to [\mathtt{state}_{p\cdot1}(\widetilde{x})], \\
\text{else } Q, p, \widetilde{x}]\!] & & [\mathtt{state}_p(\widetilde{x})] – [\mathrm{NotEq}(M, N)] \to [\mathtt{state}_{p\cdot2}(\widetilde{x})]\} \\
 & & \cup\ [\![P, p\cdot1, \widetilde{x}]\!] \ \cup\ [\![Q, p\cdot2, \widetilde{x}]\!] \\
[\![\text{event } F; P, p, \widetilde{x}]\!] & = & [\mathtt{state}_p(\widetilde{x}) – [\mathrm{Event}(), F] \to [\mathtt{state}_{p\cdot1}(\widetilde{x})] \ \cup\ [\![P, p\cdot1, \widetilde{x}]\!] \\
[\![\text{insert } s, t; P, p, \widetilde{x}]\!] & = & [\mathtt{state}_p(\widetilde{x})] – [\mathrm{Insert}(s, t)] \to [\mathtt{state}_{p\cdot1}(\widetilde{x})] \ \cup\ [\![P, p\cdot1, \widetilde{x}]\!] \\
[\![\text{delete } s; P, p, \widetilde{x}]\!] & = & [\mathtt{state}_p(\widetilde{x})] – [\mathrm{Delete}(s)] \to [\mathtt{state}_{p\cdot1}(\widetilde{x})] \ \cup\ [\![P, p\cdot1, \widetilde{x}]\!] \\
[\![\text{lookup } M \text{ as } v & = & \{[\mathtt{state}_p(\widetilde{x})] – [\mathrm{IsIn}(M, v)] \to [\mathtt{state}_{p\cdot1}(\widetilde{x})], \\
\text{in } P \text{ else } Q, p, \widetilde{x}]\!] & & [\mathtt{state}_p(\widetilde{x})] – [\mathrm{IsNotSet}(M)] \to [\mathtt{state}_{p\cdot2}(\widetilde{x})]\} \\
 & & \cup\ [\![P, p\cdot1, \widetilde{x}]\!] \ \cup\ [\![Q, p\cdot2, \widetilde{x}]\!] \\
[\![\mathrm{lock}^l\ s; P, p, \widetilde{x}]\!] & = & [\mathtt{state}_p(\widetilde{x}), \mathtt{Fr}(lock_l)] – [\mathrm{Lock}(lock_l, s)] \to [\mathtt{state}_{p\cdot1}(\widetilde{x}, lock_l)] \ \cup\ [\![P, p\cdot1, (\widetilde{x}, lock_l)]\!] \\
[\![\mathrm{unlock}^l\ s; P, p, \widetilde{x}]\!] & = & [\mathtt{state}_p(\widetilde{x})] – [\mathrm{Unlock}(lock_l, s)] \to [\mathtt{state}_{p\cdot1}(\widetilde{x})] \ \cup\ [\![P, p\cdot1, \widetilde{x}]\!] \\
[\![(l – [a] \to r); P, p, \widetilde{x}]\!] & = & [\mathtt{state}_p(\widetilde{x}), l] – [\mathrm{Event}(), a] \to [\mathtt{state}_{p\cdot1}(\widetilde{x} \cup vars(l)), r] \ \cup\ [\![P, p\cdot1, \widetilde{x} \cup vars(l)]\!]
\end{array}
$$

FIGURE 8 – Définition de $[\![P, p, \widetilde{x}]\!]$, avec E_{id} et $Reboot_{p,id}$ définis en section 4.2

De plus, on aura parfois besoin dans la démonstration de manipuler uniquement les règles produites à une position donnée du protocole. On formalise cela par la définition suivante :

Définition A.2. *Soit P un processus clos et p une position de P. On définit :*

$$
[\![P]\!]_{=p} = [\![P, [], []]\!] \ \cap\ \left(\bigcup_{\widetilde{x} \in \mathcal{V}^*} E_{id}(P, p, \widetilde{x}) \ \cup \bigcup_{(l,r,a) \in (\mathcal{G}^\#)^2 \times \mathcal{G}^*} \{[\mathtt{state}_p(\widetilde{x}), l] – [a] \to r\} \right)
$$

B Mise à jour de la démonstration de correction

La suite reprend les annexes B et C de l'article de S. Kremer *et al.* [3] afin d'en adapter les démonstrations pour les étendre au formalisme des boucles. On suit dans une large mesure l'organisation de l'article, à quelques nouvelles définitions et lemmes près : ceux numérotés en fonction de la section ont été ajoutés pour ce papier, et ceux en numérotation simple sont copiés ou adaptés de l'article. Dans ce dernier cas, ils conservent la numérotation d'origine pour faciliter la lecture parallèle des deux documents.

On notera enfin que tout n'a pas été recopié pour ne pas aloudir la structure : les lemmes dont la démonstration reste absolument ichangée et les définitions associées sont omis et peuvent être consultés sur le papier d'origine.

B.1 Première inclusion

Définition 20. *Soit P un processus clos, $\mathcal{P}$ et $\mathcal{R}$ deux multi-ensembles de processus, $\mathcal{I}$ une fonction associant un identifiant à un couple (processus,entier), et S un multi-ensemble de faits. On écrit $(\mathcal{P},\mathcal{R},\mathcal{I}) \leftrightarrow_P S$ s'il existe une fonction ϕ : $\left(\mathcal{P} \cup^{\#} \mathcal{R} \cup^{\#} fst(\mathrm{Im}\,(\mathcal{I}))\right) \to \{\mathtt{state}_p(\widetilde{t}) \in^{\#} S\}^{\#}$ telle que :*

1) *ϕ est surjective.*

2) *Chaque élément de $\mathrm{Im}\,(\phi)$ a soit exactement un unique antécédent par ϕ, soit exactement deux antécédents dont un dans $fst(\mathrm{Im}\,(\mathcal{I}))$ et un dans $\mathcal{P}$, soit deux ou plus avec exactement un dans $\mathcal{R}$ et les autres dans $\mathcal{P}$.*

De plus, si $\phi(Q_{id}) = \mathtt{state}_p(\widetilde{t})$ on a :

3) *$P_{|p}\tau = Q_{id}\varrho$ avec τ substitution et ϱ un renommage bijectif des noms frais de Q_{id} non liés à une variable.*

4) $\exists\ ri \in_E ginsts\left(\left[\!\left[P\right]\!\right]_{=p}\right),\ \left(\mathtt{state}_p(\widetilde{t}) \in^{\#} prems(ri)\right)\ \wedge\ \left(id \in \mathbf{D}(\mathcal{I})\ \Leftrightarrow\ id \in \widetilde{t}\right)$

où $\left[\!\left[P\right]\!\right]_{=p}$ est défini à la définition A.2. De plus, si ϕ a été définie et qu'il n'y a pas de confusion possible dans le contexte, on se permettra de noter $Q_{id} \leftrightarrow_P \mathtt{state}_p(\widetilde{t})$ pour signifier que $\phi(Q_{id}) = \mathtt{state}_p(\widetilde{t})$. On notera qu'on note $\mathtt{state}_p(\widetilde{t})$ de façon générique, mais que le fait `state` *en question peut très bien être permanent (vu les traductions de* `*` *et* `!`*)*

Remarque 2. *On notera que $\leftrightarrow_P$ a les deux propriétés suivantes :*

- *Si $(\mathcal{P}_1,\mathcal{R}_1,\mathcal{I}_1) \leftrightarrow_P S_1$ et $(\mathcal{P}_2,\mathcal{R}_2,\mathcal{I}_2) \leftrightarrow_P S_2$, alors $\left(\mathcal{P}_1 \cup^{\#} \mathcal{P}_2, \mathcal{R}_1 \cup^{\#} \mathcal{R}_2, \mathcal{I}_1 \cup \mathcal{I}_2\right) \leftrightarrow_P \left(S_1 \cup^{\#} S_2\right)$ pourvu que $\mathcal{I}_1$ et $\mathcal{I}_2$ soient à supports disjoints.*

- *Si $(\mathcal{P}_1,\mathcal{R}_1,\mathcal{I}_1) \leftrightarrow_P S_1$ et $Q \leftrightarrow_P \mathtt{state}_p(\widetilde{t})$ pour $Q \in \mathcal{P}_1$ et $\mathtt{state}_p(\widetilde{t}) \in S_1$, alors :*
$(\mathcal{P}_1 \backslash^{\#} \{Q\}, \mathcal{R}_1, \mathcal{I}_1) \leftrightarrow_P S_1 \backslash^{\#} \{\mathtt{state}_p(\widetilde{t})\}$

Les deux définitions suivantes servent à étendre à la manipulation des boucles l'invariant utilisé pour le lemme 10 : $End_i(p, id)$ modélise le fait qu'à la i^e application de msr d'un ensemble de règles, l'exécution du protocole à la position p sous l'identifiant de boucle id est terminée, et $Exec_i(p, id)$ qu'elle est terminée ou en cours d'exécution.

Définition B.1. *Soit P un protocole, S_i un multi-ensemble de faits, p une position et id un identifiant. On définit :*

$$\begin{aligned} End_i(p, id) \quad = \quad & \mathtt{exit}_p(id) \in^{\#} \mathcal{S}_i \\ & \vee \ \exists\ \Pi \in pos(P)^*,\ \begin{cases} [\mathtt{exit}_{pos}(id),\ pos \in \Pi] \rightarrow [\mathtt{exit}_p(id)] \in [\![P]\!] \\ \forall\ pos \in \Pi,\ End_i(pos, id) \end{cases} \end{aligned}$$

où $[\mathtt{exit}_{pos}(id),\ pos \in \Pi]$ est l'analogue séquentiel de la notation ensembliste $\{\mathtt{exit}_{pos}(id),\ pos \in \Pi\}$. On n'utilisera cette définition que dans le contexte du lemme 10, où les notations ont exactement les mêmes nom qu'ici, d'où le fait qu'on se permette un abus de notation en omettant P et S_i en paramètre de la fonction.

Définition B.2. *Soit P un protocole, S_i un multi-ensemble de faits, $\mathcal{P}_i$ et $\mathcal{R}_i$ deux multi-ensembles de processus, $\mathcal{I}$ une fonction associant un identifiant à un couple (processus,entier), p une position et id un identifiant. On suppose que $(\mathcal{P}_i, \mathcal{R}_i, \mathcal{I}_i) \leftrightarrow_P S_i$. On définit alors :*

$$\begin{aligned} Exec_i(p, id) \quad = \quad & \exists\ (Q_{id}, \widetilde{x}) \in (\mathcal{P}_i \times {\mathcal{M}_\Sigma}^*)\ ,\ Q_{id} \leftrightarrow \mathtt{state}_p(\widetilde{x}) \\ & \vee\ End_i(p, id) \\ & \vee\ \exists\ ri \in ginsts\left([\![P]\!]_{=p}\right),\ \begin{cases} \mathtt{state}_p(\widetilde{x}) \in prems(ri) \\ \forall\ \mathtt{state}_{p'}(\widetilde{y}) \in concls(ri),\ Exec_i(p', id) \end{cases} \end{aligned}$$

où $[\![P]\!]_{=p}$ est défini à la définition A.2. Pour la même raison que précédemment, on se permet d'omettre les paramètres ensemblistes fixés au début de la définition.

Lemme B.1. *Avec les notations des définitions B.1 et B.2 :*

$$\begin{cases} Exec_i(p, id) \\ \{Q_{id} \in \mathcal{P}_i\} = \varnothing \end{cases} \Rightarrow\ End_i(p, id)$$

Démonstration. Si $\mathcal{P}_i$ ne contient aucun protocole indicé par id, la définition de $Exec_i(p, id)$ devient équivalente à :

$$End_i(p, id)\ \vee\ \exists\ ri \in ginsts\left([\![P]\!]_{=p}\right),\ \begin{cases} \mathtt{state}_p(\widetilde{x}) \in prems(ri) \\ \forall\ \mathtt{state}_{p'}(\widetilde{y}) \in concls(ri),\ Exec_i(p', id) \end{cases}$$

Il suffit alors de montrer qu'avec cette nouvelle définition :

$$\left(\exists\ ri \in ginsts\left([\![P]\!]_{=p}\right),\ \begin{cases} \mathtt{state}_p(\widetilde{x}) \in prems(ri) \\ \forall\ \mathtt{state}_{p'}(\widetilde{y}) \in concls(ri),\ Exec_i(p', id) \end{cases}\right) \implies End_i(p, id)$$

Ce qui découle d'une induction sur la structure d'une position, quasi immédiate, pourvu qu'on remarque que pour p maximal dans $pos(P)$ pour l'ordre préfixe, le premier membre de l'implication ne peut pas être réalisé.

□

Lemme B.2. *Avec les notations des définitions B.1 et B.2 :*

$$\exists\ Q_{id} \in \mathcal{P}_i \ \Leftrightarrow\ id \in \mathbf{D}(\mathcal{I}) \ \wedge\ \mathrm{card}\{Q_{id} \in \mathcal{P}_i,\ Q \text{ processus}\} = snd(\mathcal{I}(id))$$

Démonstration. Induction immédiate en listant les cas de la sémantique opérationnelle. □

Lemme 10. *Soit P un processus clos bien formé. Si :*

$$(\mathcal{E}_0, \mathcal{S}_0, \mathcal{S}_0^{MS}, \mathcal{P}_0, \mathcal{R}_0, \mathcal{I}_0, \sigma_0, \mathcal{L}_0) \xrightarrow{E_1} (\mathcal{E}_1, \mathcal{S}_1, \mathcal{S}_1^{MS}, \mathcal{P}_1, \mathcal{R}_1, \mathcal{I}_1, \sigma_1, \mathcal{L}_1) \xrightarrow{E_2} \cdots \xrightarrow{E_n} (\mathcal{E}_n, \mathcal{S}_n, \mathcal{S}_n^{MS}, \mathcal{P}_n, \mathcal{R}_n, \mathcal{I}_n, \sigma_n, \mathcal{L}_n)$$

avec $(\mathcal{E}_0, \mathcal{S}_0, \mathcal{S}_0^{MS}, \mathcal{P}_0, \mathcal{R}_0, \mathcal{I}_0, \sigma_0, \mathcal{L}_0) = (\varnothing, \varnothing, \varnothing, \{P\}, \varnothing, \varnothing, \varnothing, \varnothing)$, alors il existe $(F_1, S_1), \cdots, (F_{n'}, S_{n'})$ tels que :

$$\varnothing \xrightarrow[[\![P]\!]]{F_1} S_1 \xrightarrow[[\![P]\!]]{F_2} \cdots \xrightarrow[[\![P]\!]]{F_{n'}} S_{n'} \in exec^{msr}([\![P]\!])$$

et qu'il existe une fonction $f : [\![0; n]\!] \to [\![0; n']\!]$ strictement croissante telle que $f(n) = n'$ et que pour tout $i \in [\![0; n]\!]$ on ait :

1) $\mathcal{E}_i = \left\{ a \mid ProtoNonce(a) \in \bigcup_{j=1}^{f(i)} F_j \right\}$

2) $\forall\ t \in \mathcal{M},\ \mathcal{S}_i(t) = \begin{cases} u & si\ \exists\ j \leqslant f(i),\ \mathrm{Insert}(t,u) \in F_j \\ & \quad \wedge\ \forall\ (j',u') \in ([\![0;n']\!] \times \mathcal{M}_\Sigma),\ j < j' \leqslant f(i) \Rightarrow \mathrm{Insert}(t,u') \notin^{\#} F_{j'} \\ & \quad \wedge\ \mathrm{Delete}(t) \notin^{\#} F_{j'} \\ \bot & sinon \end{cases}$

3) $\mathcal{S}_i^{MS} = S_{f(i)} \backslash^{\#} \mathcal{F}_{res}$

4) $(\mathcal{P}_i, \mathcal{R}_i, \mathcal{I}_i) \ \leftrightarrow_P\ S_{f(i)}$

5) $\{x\sigma_i \mid x \in \mathbf{D}(\sigma_i)\}^{\#} = \left\{ \mathtt{Out}(t) \in \bigcup_{k=0}^{f(i)} S_k \right\}^{\#}$

6) $\mathcal{L}_i =_E \left\{ t \mid \exists\ j \leqslant f(i),\ \exists\ u \in \mathcal{V},\ \mathrm{Lock}(u,t) \in^{\#} F_j \ \wedge\ \forall\ k \in]\!]j, f(i)]\!],\ \mathrm{Unlock}(u,t) \notin F_k \right\}$

7) $[F_1, \cdots, F_{n'}] \vDash \alpha$ *où α est défini à la figure 5 en section 4.3.*

8) $\exists\ k \in [\![f(i-1)+1; f(i)]\!],\ E_i = F_k \ \wedge \bigcup_{\substack{k=f(i-1)+1 \\ k \neq j}}^{f(i)} \subseteq \mathcal{F}_{res}$

9) $\forall\ id \in \mathbf{D}(\mathcal{I}_i),\ \begin{cases} \mathcal{I}_i(id) = (Q, n) \\ Q \leftrightarrow_P \mathtt{state}_p(\widetilde{x}) \end{cases} \Rightarrow \begin{cases} Exec_{f(i)}(p, id) \\ \mathtt{state}_p(\widetilde{x})\ permanent \end{cases}$

10) $\forall\ Q \in \mathcal{R}_i,\ Q \leftrightarrow_P \mathtt{state}_p(\widetilde{x}) \ \Rightarrow\ \mathtt{state}_p(\widetilde{x})\ permanent$

11) $\forall\ Q_{id} \ \leftrightarrow_P\ !\mathtt{state}_p(\widetilde{x}),\ \begin{cases} fst(\mathcal{I}_i(id)) \leftrightarrow_P !\mathtt{state}_p(\widetilde{x}) & si\ id \in \mathbf{D}(\mathcal{I}_i) \\ \exists\ Q \in^{\#} \mathcal{R}_i,\ Q \leftrightarrow_P !\mathtt{state}_p(\widetilde{x}) & sinon \end{cases}$

12) $\forall\ id \in \mathbf{D}(\mathcal{I}_i),\ \left(\mathrm{Reboot}(id) \in^{\#} E_{f(i)} \ \wedge\ \forall\ k \in]\!]f(i), n']\!],\ \mathrm{Finish}(id) \notin^{\#} E_k\right)$

$$\Rightarrow\ \forall\ j \in [\![i, n]\!],\ \forall\ Q_{id} \in \mathcal{P}_j,\ Q_{id} < fst(\mathcal{I}_j(id))$$

*où $P < Q$ **ssi** l'arbre syntaxique de P est strictement inclus dans celui de Q*

Intuitivement, ce résultat énonce que toute exécution d'un protocole au sens de la sémantique opérationnelle peut se représenter comme une suite d'applications de msr de la traduction de ce protocole. Autrement dit, on justifie que l'ensemble des traces produites par la traduction contient celles produites par la sémantique ; l'inclusion réciproque étant traitée dans le lemme 12.

Toutes les conditions à maintenir servent simplement à assurer l'hérédité, le résultat étant démontré par récurrence sur n. Les huit premières sont directement tirées de l'article et les dernières ont été ajoutées pour traiter les boucles. On remarquera que cette démonstration reprend celle de l'article pour la plupart des cas, mais que tous devraient être (dans l'idéal) revus, puisque la nouvelle définition de la réplication (et celle de l'itération) modifie les règles produites par le processus sous-jacent (un fait `state` est transformé en fait permanent), sans compter la modification de la définition de $\leftrightarrow$. Néanmoins, les quelques arguments à ajouter sont tous identiques, sauf pour quelques exceptions (`0`, `|`, `!`, `*` et `K`) qui seront traitées au début.

Démonstration. On démontre le résultat par récurrence sur n le nombre de transitions de l'exécution du protocole.

▪ Pour $n = 0$, on choisit $f(1) = 0$ et on choisit une unique application de la règle Init : on a ainsi $S_1 = \{\mathtt{state}_{[]}()\}^{\#}$. Sauf pour la condition 4, la vérification des conditions 1 à 12 est immédiates (souvent car la plupart des éléments de la configuration étudiée sont vides). Pour `4)` on doit montrer que $\left(\{P\}^{\#}, \varnothing, \varnothing\right) \leftrightarrow_P \mathtt{state}_{[]}()$: en reprenant les notations de la définition de $\leftrightarrow$, il suffit de choisir $\tau = \varnothing$, $\varrho = \varnothing$. Pour ri, on remarque en étudiant les cas uns à uns sur la figure 8 que $[\![P]\!]_{=[]} = [\![P, [], []]\!]_{[]}$ contient toujours une règle ri de la forme voulue (le second membre de la conjonction étant assuré puisque $\mathcal{I}$ est vide).

▪ On suppose à présent que l'invariant est vérifié pour une exécution à $n-1$ transitions, $n > 0$, et on veut montrer qu'il est toujours valable pour n transitions. Par hypothèse de récurrence, on peut fixer une fonction $f_p : [\![0; n-1]\!] \to [\![0; n']\!]$ strictement croissante et une exécution :

$$(\mathcal{E}_0, \mathcal{S}_0, \mathcal{S}_0^{MS}, \mathcal{P}_0, \mathcal{R}_0, \mathcal{I}_0, \sigma_0, \mathcal{L}_0) \xrightarrow{E_1} (\mathcal{E}_1, \mathcal{S}_1, \mathcal{S}_1^{MS}, \mathcal{P}_1, \mathcal{R}_1, \mathcal{I}_1, \sigma_1, \mathcal{L}_1) \xrightarrow{E_2} \cdots \xrightarrow{E_n} (\mathcal{E}_n, \mathcal{S}_n, \mathcal{S}_n^{MS}, \mathcal{P}_n, \mathcal{R}_n, \mathcal{I}_n, \sigma_n, \mathcal{L}_n)$$

telle que les 12 conditions de l'invariant soient vérifiées. On peut donc fixer une fonction telle que $(\mathcal{P}_{n-1}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}) \leftrightarrow_P S_{n'}$. Dans ce contexte, on peut se permettre sans ambiguïté d'écrire $Q \leftrightarrow_P \mathtt{state}_p(\widetilde{t})$ si l'image de Q par cette fonction est $\mathtt{state}_p(\widetilde{t})$. On raisonne à présent par disjonction de cas sur la n^e transition ; pour éviter d'avoir à déclarer à chaque cas les notations, on étendra la précédente exécution de s étapes et démontrera que les 12 conditions de l'invariant restent vérifiées au rang n. On étend de plus la fonction f_p en $f : \left\{ \begin{array}{ccc} [\![0; n]\!] & \to & [\![0; n' + s]\!] \\ i & \mapsto & \left\{ \begin{array}{ll} f_p(i) & \text{si } i \in [\![0; n-1]\!] \\ n' + s & \text{sinon} \end{array} \right. \end{array} \right.$.

• **Cas** $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}_{n-1}^{MS}, \mathcal{P}' \cup^{\#} \{0_{id}\}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1}) \to (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}_{n-1}^{MS}, \mathcal{P}', \mathcal{R}_{n-1}, \mathcal{I}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1})$ où $id \notin \mathbf{D}(\mathcal{I}_{n-1})$

Par hypothèse de récurrence, $(\mathcal{P}_{n-1}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}) \leftrightarrow_P S_{n'}$. On peut donc fixer p et $\widetilde{t}$ tels que $0_{id} \leftrightarrow_P \mathtt{state}_p(\widetilde{t})$ et on a donc $ri \in [\![P]\!]_{=p}$ tel que $\mathtt{state}_p(\widetilde{t})$ soit une prémice de ri d'après la définition 20. Comme $\mathcal{I}(id)$ n'est pas défini, vu la définition de $[\![P]\!]_{=p}$, ce 0 n'a pas été traduit sous une boucle donc $ri = [\mathtt{state}_p(\widetilde{t})] \to []$. On choisit alors $s = 1$ et on étend la précédente exécution msr par une application de la règle ri. La vérification des 12 conditions de l'invariant est immédiate sauf

pour la 4 et la 11. On distingue deux cas :

1^{er} cas : $\texttt{state}_p(\widetilde{t})$ est linéaire

Alors `11)` est vérifiée immédiatement. De plus, vu la remarque 2, comme alors $S_{n'+1} = S_{n'} \backslash \{\texttt{state}_p(\widetilde{t})\}$ et $\mathcal{P}_n = \mathcal{P}_{n-1} \backslash \{0_{id}\}$, `4)` est également vérifiée.

2^e cas : $\texttt{state}_p(\widetilde{t})$ est permanent

Autrement dit, $ri = [!\texttt{state}_p(\widetilde{t})] \rightarrow []$ (ce qui arrive en pratique si le 0 étudié était répliqué). D'après `11)`, comme $id \notin \mathbf{D}(\mathcal{I}_{n-1})$ on en déduit qu'on a un 0_{id} dans $\mathcal{R}_{n-1}$ qui soit aussi en relation avec $!\texttt{state}_p(\widetilde{t})$ via $\leftrightarrow_P$. Vu la définition de $\leftrightarrow_P$, on est donc dans le cas où $!\texttt{state}_p(\widetilde{t})$ a un nombre arbitraire d'antécédents dont exactement un dans $\mathcal{R}_{n-1}$: en retirer un (le $0_{id} \in \mathcal{P}_{n-1}$) n'affecte pas cette propriété, donc `4)` reste vérifiée (et la vérification de `11)` au rang n est immédiate).

• **Cas** $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}^{MS}_{n-1}, \mathcal{P}' \cup^{\#} \{0_{id}\}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1}) \rightarrow (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}^{MS}_{n-1}, \mathcal{P}', \mathcal{R}_{n-1}, \mathcal{I}_{n-1}[id \mapsto (Q, n-1)], \sigma_{n-1}, \mathcal{L}_{n-1})$ où $id \in \mathbf{D}(\mathcal{I}_{n-1})$ et $\mathcal{I}_{n-1}(id) = (Q, n)$

Par hypothèse de récurrence, $(\mathcal{P}_{n-1}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}) \leftrightarrow_P S_{n'}$. On fixe alors p et $\widetilde{t}$ tels que $0_{id} \leftrightarrow_P \texttt{state}_p(\widetilde{t})$, et on a donc $ri \in [\![P]\!]_{=p}$ tel que $\texttt{state}_p(\widetilde{t})$ soit une prémice de ri d'après la définition 20. Comme $id \in \mathbf{D}(\mathcal{I}_{n-1})$, vu la définition de $[\![P]\!]_{=p}$, ce 0 a été traduit sous une boucle et $ri = [\texttt{state}_p(\widetilde{t})]\text{–}[a]\rightarrow[\texttt{exit}_p(id)]$ avec $a = \varnothing$ ou $a = \{\text{Reboot}(id)\}$ (on remarquera que l'existence de cette règle est entre autre garantie par le fait que dans la définition de $\leftrightarrow_P$, on impose $id \in \widetilde{t}$). On choisit alors $s = 1$ et on étend la précédente exécution msr avec ri. On distingue alors les cas :

1^{er} cas : $\texttt{state}_p(\widetilde{t})$ linéaire et $a = \varnothing$

La vérification de toutes les conditions de l'invariant est immédiate, sauf pour les 4 et 9. Comme aucun fait `state` n'est produit, la condition 4 découle de la remarque 2. Pour la condition 9, seule le membre de la conjonction concernant $Exec$ reste à démontrer : par induction sur la structure d'une position, on reprend la définition de $Exec$ et tout découle de la condition `9)` donnée par l'hypothèse de récurrence, sauf pour le cas de la position p qui est néanmoins vérifiée puisque $\texttt{exit}_p(id) \in^{\#} S_{n'+1}$. L'invariant reste donc vérifié.

2^e cas : $\texttt{state}_p(\widetilde{t})$ linéaire et $a = \{\text{Reboot}(id)\}$

Impossible car les Reboot sont toujours produits avec une prémice `state` permanente.

3^e cas : $\texttt{state}_p(\widetilde{t})$ permanent et $a = \varnothing$

Impossible car vu la définition de $[\![P]\!]_{=p}$, cela imposerait d'avoir une réplication sous une itération, ce qui est interdit par la syntaxe.

4^e cas : $\texttt{state}_p(\widetilde{t})$ permanent et $a = \{\text{Reboot}(id)\}$

D'après la condition `11)` de l'invariant au rang $n-1$, $fst(\mathcal{I}_{n-1}(id))$ et $0_{id} \in^{\#} \mathcal{P}_{n-1}$ sont en relation avec $!\texttt{state}_p(\widetilde{t})$ via $\leftrightarrow_P$. On est donc dans le cas de la définition de $\leftrightarrow_P$ où $!\texttt{state}_p(\widetilde{t})$ a exactement deux antécédents dont un dans $fst(\text{Im}\,(\mathcal{I}_{n-1}))$.

Comme l'application de la règle ri ne produit aucun fait `state`, la condition 4 de l'invariant reste vérifiée (et $!\texttt{state}_p(\widetilde{t})$ aura $fst(\mathcal{I}_{n-1}(id))$ comme unique antécédent par la nouvelle application). Toutes les autres conditions de l'invariant à part `7)` sont vérifiées immédiatement.

Pour justifier cette dernière, on utilise `12)` au rang $n-1$: si l'action Reboot(id) qui a été ajoutée au rang n était précédée dans l'exécution msr d'un autre Reboot(id) sans Finish(id) intercalaire, `12)` donnerait que tous les processus indicés par id dans $\mathcal{P}_{n-1}$ (entre autre) devraient être des sous-processus stricts de $fst(\mathcal{I}_{n-1}(id))$. Or, comme on a dit que $0_{id} \leftrightarrow_P !\texttt{state}_p(\widetilde{t})$ et $fst(\mathcal{I}_{n-1}) \leftrightarrow_P !\texttt{state}_p(\widetilde{t})$, on a par définition $fst(\mathcal{I}_{n-1}(id)) = 0$ (on n'a pas besoin de se préoccuper des renommages avec τ et ϱ puisque 0 ne contient pas de variable). Cela est incohérent car $0_{id} \in^{\#} \mathcal{P}_{n-1}$ par hypothèse alors 0 n'est pas un sous processus strict de 0 ; α_{loop} est donc bien vérifié, donc α également puisque les autres membres de la conjonction découlent de la condition 7 au rang $n-1$. L'invariant reste donc bien vérifié.

- **Cas** $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}^{MS}_{n-1}, \mathcal{P}' \cup^{\#} \{(Q|R)_{id}\}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1}) \rightarrow (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}^{MS}_{n-1}, \mathcal{P}' \cup^{\#} \{Q_{id}, R_{id}\}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1})$ si $id \notin \mathbf{D}(\mathcal{I}_{n-1})$

Comme $(\mathcal{P}_{n-1}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}) \leftrightarrow_P S_{n-1}$, on a p et $\widetilde{t}$ tels que $(Q|R)_{id} \leftrightarrow_P \texttt{state}_p(\widetilde{t})$ et donc également $ri \in [\![P]\!]_{=p}$ tel que $\texttt{state}_p(\widetilde{t})$ soit une prémice de de ri d'après la définition 20. Par définition de $[\![P]\!]_{=p}$, comme $\mathcal{I}_{n-1}(id)$ n'est pas défini, ce $Q|R$ n'a pas été traduit sous une boucle donc $ri = [\texttt{state}_p(\widetilde{t})] \rightarrow [\texttt{state}_{p\cdot 1}(\widetilde{t}),\ \texttt{state}_{p\cdot 2}]$. On choisit alors $s = 1$ et on étend la précédente exécution msr par l'application de la règle ri. La vérification des 12 conditions de l'invariant est immédiate sauf pour 4 et 11 (la `12)` découle du fait qu'on ne produit des sous-processus des précédents). On distingue deux cas :

1^{er} cas : $\texttt{state}_p(\widetilde{t})$ est linéaire

Alors `11)` est vérifiée immédiatement. De plus, vu la remarque 2 (en combinant une fois son second point et deux fois son premier), on obtient que `4)` est également vérifié.

2^{e} cas : $\texttt{state}_p(\widetilde{t})$ est permanent

Autrement dit, $ri = [!\texttt{state}_p(\widetilde{t})] \rightarrow []$ (ce qui arrive en pratique si le $Q|R$ est répliqué). D'après `11)`, comme $id \notin \mathbf{D}(\mathcal{I}_{n-1})$, on en déduit qu'on a un $Q|R$ dans $\mathcal{R}_{n-1}$ en relation avec $!\texttt{state}_p(\widetilde{t})$ via $\leftrightarrow_P$. Vu la définition de $\leftrightarrow_P$, on est dans cas où $!\texttt{state}_p(\widetilde{t})$ a un nombre arbitraire d'antécédents dont exactement un dans $\mathcal{R}_{n-1}$: en retirer un (le $Q|R \in \mathcal{P}_{n-1}$) n'affecte pas cette propriété, donc `4)` reste vérifiée (et la vérification de `11)` au rang n est immédiate).

- **Cas** $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}^{MS}_{n-1}, \mathcal{P}' \cup^{\#} \{(Q|R)_{id}\}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1}) \rightarrow (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}^{MS}_{n-1}, \mathcal{P}' \cup^{\#} \{Q_{id}, R_{id}\}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}[id \mapsto (P', n+1)], \sigma_{n-1}, \mathcal{L}_{n-1})$ si $id \notin \mathbf{D}(\mathcal{I}_{\mathbf{n-1}})$ et $\mathcal{I}_{n-1}(id) = (P', n)$

Comme $(\mathcal{P}_{n-1}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}) \leftrightarrow_P S_{n-1}$, on a p et $\widetilde{t}$ tels que $(Q|R)_{id} \leftrightarrow_P \texttt{state}_p(\widetilde{t})$ et donc également $ri \in [\![P]\!]_{=p}$ tel que $\texttt{state}_p(\widetilde{t})$ soit une prémice de de ri d'après la définition 20. Comme $id \in \mathbf{D}(\mathcal{I}_{n-1})$, vu la définition de $[\![P]\!]_{=p}$, ce $Q|R$ a été traduit sous une boucle et $ri = [\texttt{state}_p(\widetilde{t})]$–$[a]\rightarrow[\texttt{state}_{p\cdot 1}(\widetilde{t}), \texttt{state}_{p\cdot 2}(\widetilde{t})]$ avec $a = \varnothing$ ou $a = \{\text{Reboot}(id)\}$. On choisit alors $s = 1$ et on étend la précédente exécution msr avec ri. On distingue alors les cas :

1^{er} cas : $\texttt{state}_p(\widetilde{t})$ linéaire et $a = \varnothing$

La vérification de toutes les conditions de l'invariant est identique au 1^{er} cas du point précédent.

2^e cas : $\texttt{state}_p(\widetilde{t})$ linéaire et $a = \text{Reboot}(id)$

Impossible car les Reboot sont toujours produits avec une prémice `state` permanente.

3^e cas : $\texttt{state}_p(\widetilde{t})$ permanent et $a = \varnothing$

Impossible car vu la définition de $[\![P]\!]_{=p}$, cela imposerait d'avoir une réplication sous une itération, ce qui est interdit par la syntaxe.

4^e cas : $\texttt{state}_p(\widetilde{t})$ permanent et $a = \{\text{Reboot}(id)\}$

D'après la condition `11)` de l'invariant au rang $n-1$, $fst(\mathcal{I}_{n-1}(id))$ et $(Q|R)_{id} \in^{\#} \mathcal{P}_{n-1}$ sont en relation avec $!\texttt{state}_p(\widetilde{t})$ via $\leftrightarrow_P$. On est donc dans le cas de la définition de $\leftrightarrow_P$ où $!\texttt{state}_p(\widetilde{t})$ a exactement deux antécédents dont un dans $fst(\text{Im}\,(\mathcal{I}_{n-1}))$.

Il suffit alors de définir la nouvelle application pour $\leftrightarrow_P$ en associant Q à $\texttt{state}_{p\cdot 1}(\widetilde{t})$, R à $\texttt{state}_{p\cdot 2}(\widetilde{t})$ et en maintenant le reste des associations ($!\texttt{state}_p(\widetilde{t})$ aura alors $fst(\mathcal{I}_{n-1}(id))$ comme unique antécédent via $\leftrightarrow_P$ au rang n). Toutes les autres conditions de l'invariant à part `7)` sont vérifiées immédiatement.

Pour justifier cette dernière, on utilise `12)` au rang $n-1$: si l'action Reboot(id) qui a été ajoutée au rang n était précédée dans l'exécution msr d'un autre Reboot(id) sans Finish(id) intercalaire, `12)` donnerait que tous les processus indicés par id dans $\mathcal{P}_{n-1}$ (entre autre) devraient être des sous-processus stricts de $fst(\mathcal{I}_{n-1}(id))$. Or, comme on a dit que $(P|Q)_{id} \leftrightarrow_P !\texttt{state}_p(\widetilde{t})$ et $fst(\mathcal{I}_{n-1}) \leftrightarrow_P !\texttt{state}_p(\widetilde{t})$, on a par définition $fst(\mathcal{I}_{n-1}(id))$ et $Q|R$ ayant les mêmes arbres syntaxiques (aux noms des variables près). Cela contredirait alors `12)` : α_{loop} est donc bien vérifié, donc α également puisque les autres membres de la conjonction découlent de la condition 7 au rang $n-1$. L'invariant reste donc bien vérifié.

• **Cas** $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}^{MS}_{n-1}, \mathcal{P}' \cup^{\#} \{!Q\}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1}) \rightarrow (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}^{MS}_{n-1}, \mathcal{P}', \mathcal{R}_{n-1} \cup^{\#} \{Q\}, \mathcal{I}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1})$

Comme $(\mathcal{P}_{n-1}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}) \leftrightarrow_P S_{n-1}$, on a p et $\widetilde{t}$ tels que $!Q \leftrightarrow_P \texttt{state}_p(\widetilde{t})$ et donc également $ri \in [\![P]\!]_{=p}$ tel que $\texttt{state}_p(\widetilde{t})$ soit une prémice de de ri d'après la définition 20. Par définition de $[\![P]\!]_{=p}$, on a donc $ri = [\texttt{state}_p(\widetilde{t})] \rightarrow [!\texttt{state}_{p\cdot 1}(\widetilde{t})]$ (on rappelle que même si $\texttt{state}_p(\widetilde{t})$ ne porte pas ici la marque des états permanents pour simplifier les notations, il peut a priori en être un). Cette msr ne peut pas être étiquetée par l'action Reboot(id) car une réplication ne peut pas être placée sous une itération. On choisit alors $s = 1$ et on étend la précédente exécution msr avec l'application de la règle ri ; on distingue deux cas :

1^{er} cas : $\texttt{state}_p(\widetilde{t})$ est linéaire

La vérification de toutes les conditions de l'invariant sauf `4)`et `10)` est immédiate. On conserve la définition de l'application induite par $\leftrightarrow_P$ pour les processus encore définis et on l'étend avec $Q \in^{\#} \mathcal{R}_n \leftrightarrow_P !\texttt{state}_{p\cdot 1}(\widetilde{t})$. La vérification de `4)` est alors immédiate et celle de `10)` découle de la

construction.

2^e cas : $\texttt{state}_p(\widetilde{t})$ est permanent

La vérification de toutes les condition de l'invariant à part `4)`, `10)`et `11)`est immédiate. D'après `11)`au rang $n-1$, comme $!Q \leftrightarrow_P !\texttt{state}_p(\widetilde{t})$ mais que $!Q$ ne peut pas porter d'identifiant du domaine de $\mathcal{I}_{n-1}$, on en déduit que $!Q \in^{\#} \mathcal{R}_{n-1}$ et que ce dernier élément est en relation avec $!\texttt{state}_p(\widetilde{t})$ via $\leftrightarrow_P$. On en déduit qu'on est dans la configuration où $!\texttt{state}_p(\widetilde{t})$ a un nombre arbitraire d'antécédents dont exactement un dans $\mathcal{R}_{n-1}$ (qui est $!Q$) : retirer un d'eux (ici, $!Q \in \mathcal{P}_{n-1}$) n'affecte pas cette propriété, et il suffit donc de suivre la construction du premier cas pour conclure.

• **Cas** $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}^{MS}_{n-1}, \mathcal{P}_{n-1}, \mathcal{R}' \cup^{\#} \{Q\}, \mathcal{I}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1}) \rightarrow (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}^{MS}_{n-1}, \mathcal{P}_{n-1} \cup^{\#} \{Q\}, \mathcal{R}' \cup^{\#} \{Q\}, \mathcal{I}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1})$

Comme $(\mathcal{P}_{n-1}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}) \leftrightarrow_P S_{n-1}$, on a p et $\widetilde{t}$ tels que $Q \in^{\#} \mathcal{R}_{n-1} \leftrightarrow_P !\texttt{state}_p(\widetilde{t})$ (cet état étant permanent d'après `10)`). Vu la définition de $\leftrightarrow_P$, un état peut avoir un nombre arbitraire d'antécédents pourvu qu'un exactement soit dans $\mathcal{R}_n = \mathcal{R}_{n-1}$ et tous les autres dans $\mathcal{P}_n$: il suffit alors pour vérifier `4)`au rang n d'étendre la définition de $\leftrightarrow_P$ avec $Q \in^{\#} \mathcal{P}_n \leftrightarrow_P !\texttt{state}_p(\widetilde{t})$. Toutes les autres conditions de l'invariant se vérifient immédiatement.

• **Cas** $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}^{MS}_{n-1}, \mathcal{P}_{n-1}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1}) \xrightarrow{K(M)} (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}^{MS}_{n-1}, \mathcal{P}_{n-1}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1})$ si $\nu\mathcal{E}.\sigma \vdash M$

Le traitement de ce cas est en tout point identique à celui de l'article.

• **Cas** $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}^{MS}_{n-1}, \mathcal{P}' \cup^{\#} \{*Q\}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1}) \rightarrow (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}^{MS}_{n-1}, \mathcal{P}' \cup^{\#} \{Q_{id}\}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}[id \mapsto (Q,1)], \sigma_{n-1}, \mathcal{L}_{n-1})$ où id est un nom frais

Comme $(\mathcal{P}_{n-1}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}) \leftrightarrow_P S_{n-1}$, on a p et $\widetilde{t}$ tels que $*Q \leftrightarrow_P \texttt{state}_p(\widetilde{t})$ et donc également $ri \in [\![P]\!]_{=p}$ tel que $\texttt{state}_p(\widetilde{t})$ soit une prémice de de ri d'après la définition 20. Par définition de $[\![P]\!]_{=p}$, on a donc $ri = [\texttt{state}_p(\widetilde{t}), \texttt{Fr}(id)] \rightarrow [!\texttt{state}_{p\cdot 1}(\widetilde{t}, id)]$ (on rappelle que même si $\texttt{state}_p(\widetilde{t})$ ne porte pas ici la marque des états permanents pour simplifier les notations, il peut a priori en être un). Cette msr ne peut pas être étiquetée par l'action Reboot(id) car la syntaxe interdit les boucles imbriquées. On choisit alors $s = 1$ et on étend la précédente exécution msr avec l'application de la règle ri, ce qui ne pose pas de problème avec les règles FRESH puisque id est un nom frais. On distingue deux cas :

1^{er} cas : $\texttt{state}_p(\widetilde{t})$ est linéaire

La vérification de toutes les conditions de l'invariant sauf `4)`et `9)`est immédiate. On conserve la définition de l'application induite par $\leftrightarrow_P$ pour les processus encore définis et on l'étend avec $Q \in^{\#} \mathcal{P}_n \leftrightarrow_P !\texttt{state}_{p\cdot 1}(\widetilde{t}, id)$ et $Q \in^{\#} fst(\mathrm{Im}\,(\mathcal{I}_n)) \leftrightarrow_P !\texttt{state}_{p\cdot 1}(\widetilde{t}, id)$. La vérification de `4)` est alors immédiate (on est dans la configuration où $!\texttt{state}_{p\cdot 1}(\widetilde{t}, id)$ a exactement deux antécédents dont exactement un dans $fst(\mathrm{Im}\,(\mathcal{I}_n))$) et celle de `9)`découle de la construction.

2^e cas : $\texttt{state}_p(\widetilde{t})$ est permanent

La vérification de toutes les condition de l'invariant à part `4)`, `9)`et `11)`est immédiate. D'après

`11)`au rang $n-1$, comme $*Q \leftrightarrow_P !\texttt{state}_p(\widetilde{t})$ mais que $*Q$ ne peut pas porter d'identifiant du domaine de $\mathcal{I}_{n-1}$ (ou on aurait une réplication sous une itération), on en déduit que $*Q \in^{\#} \mathcal{R}_{n-1}$ et que ce dernier élément est en relation avec $!\texttt{state}_p(\widetilde{t})$ via $\leftrightarrow_P$. On est donc dans la configuration où $!\texttt{state}_p(\widetilde{t})$ a un nombre arbitraire d'antécédents dont exactement un dans $\mathcal{R}_{n-1}$ (qui est $*Q$) : retirer un d'eux (ici, $*Q \in \mathcal{P}_{n-1}$) n'affecte pas cette propriété, et il suffit donc de suivre la construction du premier cas pour conclure.

• **Cas** $(\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}^{MS}_{n-1}, \mathcal{P}_{n-1}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}, \sigma_{n-1}, \mathcal{L}_{n-1}) \rightarrow (\mathcal{E}_{n-1}, \mathcal{S}_{n-1}, \mathcal{S}^{MS}_{n-1}, \mathcal{P}_{n-1} \cup^{\#} \{Q\}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}[id \mapsto (Q,1)], \sigma_{n-1}, \mathcal{L}_{n-1})$ si $\mathcal{I}_{n-1}(id) = (Q,0)$

Comme $(\mathcal{P}_{n-1}, \mathcal{R}_{n-1}, \mathcal{I}_{n-1}) \leftrightarrow_P S_{n'}$, on peut fixer p et $\widetilde{t}$ tels que $fst(\mathcal{I}_{n-1}(id)) \leftrightarrow_P !\texttt{state}_p(\widetilde{t})$ (ce fait est permanent d'après `9)`au rang $n-1$). D'après `9)` au rang $n-1$ on a également $Exec_{n'}(p, id)$; or d'après le lemme B.2, on a également $\{R_{id} \in^{\#} \mathcal{P}_i,\ R\ processus\} = \varnothing$. Donc d'après le lemme B.1 on a $End_{n'}(p, id)$: autrement dit, avec une démonstration simple par induction (en comparant la structure de End définition B.1 et de E_{id} figure 3), on peut construire une suite d'applications de msr de $[\![P]\!]_{=p}$ contenant uniquement des faits `exit` et aboutissant sur une règle étiquetée par l'action Finish(id). Il suffit alors de prolonger l'exécution msr précédente avec ces règles comme on va le justifier ci-dessous.

On note déjà que la vérification de toutes les conditions de l'invariant exceptées `4)`, `9)`, `11)` est immédiate. Pour la `4)`, on prolonge la définition de $\leftrightarrow_P$ au rang n avec l'association $Q_{id} \in \mathcal{P}_n \leftrightarrow_P !\texttt{state}_p(\widetilde{t})$. En effet, d'après `11)`, même si dans la définition de $\leftrightarrow_P$ on autorise $!\texttt{state}_p(\widetilde{t})$ à avoir un antécédent dans $fst(\mathrm{Im}\,(\mathcal{I}_n))$ et un autre quelconque dans $\mathcal{P}_n$, ce dernier ne peut être en pratique qu'un processus indicé par id, sans quoi avec `11)` on aurait deux éléments de $fst(\mathrm{Im}\,(\mathcal{I}_n))$ en relation avec le même fait, ce qui contredit la définition de $\leftrightarrow_P$.

D'après le lemme B.2, $\mathcal{P}_n$ contient Q_{id} comme unique processus indicé par id, donc on est bien dans la configuration où $!\texttt{state}_p(\widetilde{t})$ a deux antécédents dont exactement un dans $fst(\mathrm{Im}\,(\mathcal{I}_n))$ et un autre dans $\mathcal{P}_n$, d'où `4)` au rang n. La vérification de `9)` est alors immédiate, et celle de `11)` également.

• On peut ensuite se passer de traiter les autres cas de la sémantique. En effet, l'étude de chaque transition de la sémantique se décompose à chaque en quatre temps :

`1)` Mise en place des hypothèses ; cette partie est directement copiée de l'article à cela près qu'on rajoute aux règles ri (avec les notations précédentes) une action a pouvant être Reboot(id) ou vide.

`2)` Étude du cas où les prémices $\texttt{state}_p(\widetilde{t})$ (notations précédentes) sont linéaires. Cela correspond à la démonstration de l'article ; on remarquera que tous les nouvelles conditions de l'invariant introduites depuis l'article sont vérifiées immédiatement et que la modification de la définition de $\leftrightarrow_P$ n'importe pas puisqu'on a adapté la remarque 2 au nouveau formalisme.

`3)` Étude du cas où les prémices $\texttt{state}_p(\widetilde{t})$ sont permanentes et où a est vide. Cela correspond au cas où le processus suit directement une réplication ; on calque la construction de l'article à la différence près de l'extension de l'application définissant $\leftrightarrow_P$. Il suffit pour cela de suivre le 2^e cas de l'étude de la transition associée aux processus parallèles non itérés, p.23, la vérification de `4)` y étant également détaillée.

`4)` Étude du cas où les prémices $\texttt{state}_p(\widetilde{t})$ sont permanentes et où $a = \text{Reboot}(id)$. Cela correspond au cas où le processus suit directement une itération ; tout est identique au point `3)` ci-dessus,

ci ce n'est qu'on suit cette fois le 4^e cas de l'étude de la transition associée aux processus parallèles itérés, p.23, la vérification de `4)` et `7)` y étant également détaillée.

On se permet donc de se passer d'un fastidieux travail de recopiage et de copier-coller qui n'a pas un intérêt très poussé. On notera néanmoins que dans le cas de l'instruction νa, il faut ajouter l'itération à la liste des constructeurs générant des noms frais (voir le détail de la démonstration de l'article [3]), bien que cela n'affecte pas le raisonnement. Cela achève donc la démonstration de ce lemme.

□

B.2 Inclusion réciproque

Définition 22. *Soit P un processus clos, $\mathcal{P}$ et $\mathcal{R}$ deux multi-ensembles de processus, $\mathcal{I}$ une fonction associant un identifiant à un couple (processus,entier), et S un multi-ensemble de faits. On écrit $(\mathcal{P},\mathcal{R},\mathcal{I}) \leftrightsquigarrow_P S$ s'il existe ϕ : $\left(\mathcal{P} \cup^{\#} \mathcal{R} \cup^{\#} fst(\mathrm{Im}\,(\mathcal{I}))\right) \to \{\mathtt{state}_p(\widetilde{t}) \in^{\#} S\}^{\#}$ telle que :*

`1)` *ϕ est surjective.*

`2)` *Chaque élément de $\mathrm{Im}\,(\phi)$ a soit exactement un unique antécédent par ϕ, soit exactement deux antécédents dont un dans $fst(\mathrm{Im}\,(\mathcal{I}))$ et un dans $\mathcal{P}$, soit deux ou plus avec exactement un dans $\mathcal{R}$ et les autres dans $\mathcal{P}$.*

De plus, si $\phi(Q_{id}) = \mathtt{state}_p(\widetilde{t})$ on a :

`3)` $\mathtt{state}_p(\widetilde{t}) \in_E prems(R)$ *pour* $R \in ginsts\left(\llbracket P \rrbracket_{=p}\right)$

`4)` *Soit θ une substitution closant $\mathtt{state}_p(\widetilde{x}) \in prems\left(\llbracket P \rrbracket_{=p}\right)$ telle que $\widetilde{t} = \widetilde{x}\theta$. Alors :*

$\left(P_{\lfloor p}\tau\right))\,\varrho =_E Q_{id}$

pour τ substitution et ϱ un renommage bijectif des noms frais de Q_{id} non liés à une variable, définis par : $\begin{cases} \tau(x) = \theta(x) & \text{si } x \text{ n'est pas une variable réservée} \\ \varrho(a) = a' & \text{si } \theta(n_a) = a' \end{cases}$

Remarque 3. *On notera que $\leftrightsquigarrow_P$ a les deux propriétés suivantes :*

- *Si $(\mathcal{P}_1,\mathcal{R}_1,\mathcal{I}_1) \leftrightsquigarrow_P S_1$ et $(\mathcal{P}_2,\mathcal{R}_2,\mathcal{I}_2) \leftrightsquigarrow_P S_2$, alors $\left(\mathcal{P}_1 \cup^{\#} \mathcal{P}_2, \mathcal{R}_1 \cup^{\#} \mathcal{R}_2, \mathcal{I}_1 \cup \mathcal{I}_2\right) \leftrightsquigarrow_P \left(S_1 \cup^{\#} S_2\right)$ pourvu que $\mathcal{I}_1$ et $\mathcal{I}_2$ soient à supports disjoints.*

- *Si $(\mathcal{P}_1,\mathcal{R}_1,\mathcal{I}_1) \leftrightsquigarrow_P S_1$ et $Q \leftrightsquigarrow_P \mathtt{state}_p(\widetilde{t})$ pour $Q \in \mathcal{P}_1$ et $\mathtt{state}_p(\widetilde{t}) \in S_1$, alors :*
$(\mathcal{P}_1 \backslash^{\#} \{Q\}, \mathcal{R}_1, \mathcal{I}_1) \leftrightsquigarrow_P S_1 \backslash^{\#} \{\mathtt{state}_p(\widetilde{t})\}$

Suit un lemme qu'on utilisera dans la démonstration du lemme 12 ; on note toujours $\sqsubseteq$ l'ordre préfixe sur les positions.

Lemme B.3. *Soit P un processus clos bien formé, et une exécution msr satisfaisant α :*

$$\varnothing \xrightarrow{E_1}_{\llbracket P \rrbracket} S_1 \xrightarrow{E_2}_{\llbracket P \rrbracket} \cdots \xrightarrow{E_n}_{\llbracket P \rrbracket} S_n \in exec^{msr}(\llbracket P \rrbracket)$$

On fixe $i \in [\![1;n]\!]$ et on suppose que $\text{Reboot}(id) \notin E_j$ *pour tout* $j \geqslant i$. *Alors :*

1) $\forall\ (p_1,p_2) \in pos(P)^2,\ \forall\ \widetilde{x} \in \mathcal{T}_\Sigma^*,\ \left\{ \begin{array}{l} \texttt{state}_{p_1}(\widetilde{x}, id) \in S_i \\ p_2 \sqsubseteq p_1 \end{array} \right. \Rightarrow\ \texttt{state}_{p_1}(\widetilde{x}, id)$ *permanent*

2) $\forall\ (p_1,p_2) \in pos(P)^2,\ \forall\ \widetilde{x} \in \mathcal{T}_\Sigma^*,\ \left\{ \begin{array}{l} \texttt{state}_{p_1}(\widetilde{x}, id) \in S_i \\ \texttt{exit}_{p_2}(id) \in S_i \\ p_2 \sqsubseteq p_1 \end{array} \right. \Rightarrow\ \texttt{state}_{p_1}(\widetilde{x}, id)$ *permanent*

3) $\forall\ (p_1,p_2) \in pos(P)^2,\ \left\{ \begin{array}{l} \texttt{exit}_{p_1}(id) \in S_i \\ \texttt{exit}_{p_2}(id) \in S_i \\ p_2 \sqsubseteq p_1 \end{array} \right. \Rightarrow\ p_1 = p_2$

Démonstration. On ne rédige pas cette démonstration dans le détail, qui repose sur trois inductions rapides sur la structure de p_1. On donne néanmoins le raisonnement général :

▪ Dans un premier temps, on démontre 1) : le seul détail à mentionner est qu'aucun fait `state` permanent ne peut être produit sous la boucle étudiée (dont l'existence est justifiée par la présence de l'identifiant id dans $\texttt{state}_{p_1}(\widetilde{x}, id)$). Cela découle de la définition de $[\![P]\!]$ puisque les boucles et les réplications ne peuvent pas être présentes sous une autre boucle par définition de la syntaxe. On obtient donc même un résultat un peu plus fort, à savoir que sous une boucle, tous les faits `state` sont progressivement consommés (jusqu'au prochain Reboot de la boucle), sauf l'unique fait `state` permanent initial.

▪ On démontre ensuite 2) par une induction de même nature. Le seul cas où la vérification de 2) n'est pas immédiate est celui où $p_1 = p_2$, mais l'hérédité découle alors de 1).

▪ Pour 3), on procède de même : la vérification est immédiate pourvu qu'on remarque qu'avec 1) et 2), S_i ne peut pas contenir deux exemplaires d'un même fait `exit` ; du moins, cette justification suffit si le `exit` n'est pas produit par une règle dont le membre gauche contient un `state` permanent. Dans ce cas l'argument est un peu différent : l'unicité des `exit` dans S_i découle de α_{loop}, puisque cette règle porte nécessairement l'action $\text{Reboot}(id)$ par définition de la syntaxe et de $[\![P]\!]$.

□

Lemme 12. *Soit P un processus clos bien formé. Si :*

$$\varnothing \xrightarrow{E_1}_{[\![P]\!]} S_1 \xrightarrow{E_2}_{[\![P]\!]} \cdots \xrightarrow{E_n}_{[\![P]\!]} S_n \in exec^{msr}([\![P]\!])$$

est normal (voir définition 21 de l'article) et que $[E_1, \cdots, E_n] \vDash \alpha$ *(voir figure 5 en section 4.3), alors il existe une suite de configurations* $(\mathcal{E}_i, \mathcal{S}_i, \mathcal{S}_i^{MS}, \mathcal{P}_i, \mathcal{R}_i, \mathcal{I}_i, \sigma_i, \mathcal{L}_i)_{i \in [\![0;n]\!]}$ *et* $F_1, \cdots, F_{n'}$ *tels que :*

$$(\mathcal{E}_0, \mathcal{S}_0, \mathcal{S}_0^{MS}, \mathcal{P}_0, \mathcal{R}_0, \mathcal{I}_0, \sigma_0, \mathcal{L}_0) \xrightarrow{F_1} (\mathcal{E}_1, \mathcal{S}_1, \mathcal{S}_1^{MS}, \mathcal{P}_1, \mathcal{R}_1, \mathcal{I}_1, \sigma_1, \mathcal{L}_1) \xrightarrow{F_2} \cdots \xrightarrow{F_n} (\mathcal{E}_n, \mathcal{S}_n, \mathcal{S}_n^{MS}, \mathcal{P}_n, \mathcal{R}_n, \mathcal{I}_n, \sigma_n, \mathcal{L}_n)$$

avec $(\mathcal{E}_0, \mathcal{S}_0, \mathcal{S}_0^{MS}, \mathcal{P}_0, \mathcal{R}_0, \mathcal{I}_0, \sigma_0, \mathcal{L}_0) = (\varnothing, \varnothing, \varnothing, \{P\}, \varnothing, \varnothing, \varnothing, \varnothing)$ *et qu'il existe une fonction* $f : [\![1;n]\!] \to [\![0;n']\!]$ *croissante et surjective telle que* $f(n) = n'$ *et que pour tout* $i \in [\![1;n]\!]$ *on ait :*

1) $\mathcal{E}_{f(i)} = \left\{ a \in FN \mid ProtoNonce(a) \in_E \bigcup_{j=1}^{i} E_j \right\}$

2) $\forall\ t \in \mathcal{M},\ \mathcal{S}_{f(i)}(t) = \left\{ \begin{array}{ll} u & si\ \exists\ j \leqslant i,\ \text{Insert}(t,u) \in E_j \\ & \quad \wedge\ \forall\ (j',u') \in ([\![0;n']\!] \times \mathcal{M}_\Sigma),\ j < j' \leqslant i \Rightarrow \text{Insert}(t,u') \notin^{\#} E_{j'} \\ & \quad \wedge\ \text{Delete}(t) \notin^{\#} E_{j'} \\ \bot & sinon \end{array} \right.$

3) $\mathcal{S}^{MS}_{f(i)} = S_i \backslash^{\#} \mathcal{F}_{res}$

4) $(\mathcal{P}_{f(i)}, \mathcal{R}_{f(i)}, \mathcal{I}_{f(i)}) \ \leftrightsquigarrow_P \ S_i$

5) $\{x\sigma_{f(i)} \mid x \in \mathbf{D}(\sigma_{\mathbf{f(i)}})\}^{\#} = \left\{ \mathtt{Out}(t) \in \bigcup_{k=0}^{i} S_k \right\}^{\#}$

6) $\mathcal{L}_{f(i)} =_E \left\{ t \mid \exists\ j \leqslant i,\ \exists\ u \in \mathcal{V},\ \mathrm{Lock}(u,t) \in^{\#} E_j \ \wedge\ \forall\ k \in]\!]j,i]\!],\ \mathrm{Unlock}(u,t) \notin_E E_k \right\}$

7) $hide([E_1, \cdots, E_n]) =_E [F_1, \cdots, F_{n'}]$

8) $\forall\ id \in \mathbf{D}(\mathcal{I}_{f(i)}),\ \begin{cases} (\mathrm{Finish(id)} \in E_i \ \wedge\ \forall\ j > i,\ \mathrm{Reboot}(id) \notin E_j) \ \vee\ \forall\ j \in [\![1;i]\!],\ \mathrm{Reboot}(id) \notin E_j \\ fst(\mathcal{I}_{f(i)}(id)) \leftrightsquigarrow_P !\mathtt{state}_p(\widetilde{x}) \end{cases}$

$$\Rightarrow\ \exists\ Q_{id} \in^{\#} \mathcal{P}_{f(i)},\ Q_{id} \leftrightsquigarrow_P !\mathtt{state}_p(\widetilde{x})$$

9) $\forall\ !\mathtt{state}_p(\widetilde{x}) \in S_i,\ \begin{cases} fst(\mathcal{I}_{f(i)}(id)) \leftrightsquigarrow_p !\mathtt{state}_p(\widetilde{x}) & si\ id \in \mathbf{D}(\mathcal{I}_{f(i)}) \cap \widetilde{x} \\ \exists\ Q \in \mathcal{R}_{f(i)},\ Q \leftrightsquigarrow_P !\mathtt{state}_p(\widetilde{x}) & sinon \end{cases}$

10) $\forall\ id \in \mathbf{D}(\mathcal{I}_{f(i)}),\ \big(\mathrm{Reboot}(id) \in^{\#} E_i \ \wedge\ \forall\ k \in]\!]i,n]\!],\ \mathrm{Finish}(id) \notin^{\#} E_k\big)$

$$\Rightarrow\ \forall\ j \in [\![i,n]\!],\ \forall\ Q_{id} \in \mathcal{P}_{f(j)},\ Q_{id} < fst(\mathcal{I}_{f(j)}(id))$$

*où $P < Q$ **ssi** l'arbre syntaxique de P est strictement inclus dans celui de Q*

11) $\forall\ id \in \mathbf{D}(\mathcal{I}_{f(i)}),\ \forall\ Q_{id} \in^{\#} \mathcal{P}_{(f(i))},\ \forall\ \mathtt{state}_p(\widetilde{x}) \in^{\#} S_i,\ Q_{id} \leftrightsquigarrow_P \mathtt{state}_p(\widetilde{x}) \ \Rightarrow\ id \in \widetilde{x}$

Comme on l'a expliqué précédemment, ce lemme est une sorte de réciproque du lemme 10, et permet de justifier que toute exécution msr de la traduction peut s'interpréter comme une exécution du même protocole au sens de la sémantique opérationnelle.

Démonstration. Pour cette démonstration, on se permettra d'ajouter des règles fictives de la forme $[!\mathtt{state}_p(\widetilde{x})] \to [!\mathtt{state}_p(\widetilde{x})]$, où $!\mathtt{state}_p(\widetilde{x}) \in prems([\![P]\!])$. L'utilisation de ces règles n'influencent en rien une exécution msr ; pourvu que ce soit à une étape où elles sont applicables, elles peuvent être ajoutée sans dommage à n'importe quel étape de l'exécution. Elles ne représentent rien en substance, mais on a besoin de ce petit tour pour s'adapter à la nouvelle sémantique de `!` : plus précisément, cela permet de maintenir la surjectivité de f alors que le lancement d'un nouveau processus répliqué se fait en une seule application de msr mais en deux étapes de la sémantique.

On raisonne par récurrence sur n le nombre d'étapes de l'exécution msr :

▪ Comme une exécution *msr* normale (voir définition 21 de l'article) contient toujours une application initiale de la règle INIT, on initialise la récurrence à $n = 1$. On choisit alors $n' = 0$ et donc $(\mathcal{E}_0, \mathcal{S}_0, \mathcal{S}^{MS}_0, \mathcal{P}_0, \mathcal{R}_0, \mathcal{I}_0, \sigma_0, \mathcal{L}_0) = (\varnothing, \varnothing, \varnothing, \{P\}, \varnothing, \varnothing, \varnothing, \varnothing)$. On définit f par $f(1) = 0$; toutes les conditions de l'invariant sont vérifiées immédiatement (pour `4)` il suffit de choisir l'application qui associe P à $\mathtt{state}_{[]}()$, avec $\theta = \varnothing$, $\varrho = \varnothing$, $\tau = \varnothing$).

▪ On suppose à présent que l'invariant est vérifié pour $n-1$, $n \geqslant 2$, et on veut démontrer que les 11 conditions de l'invariant restent valables au rang n. On se donne donc :

$$\varnothing \xrightarrow{E_1}_{[\![P]\!]} S_1 \xrightarrow{E_2}_{[\![P]\!]} \cdots \xrightarrow{E_n}_{[\![P]\!]} S_n \in exec^{msr}([\![P]\!])$$

une exécution msr normale telle que $[E_1, \cdots, E_n] \vDash \alpha$ et on veut construire une exécution associée remplissant les 11 conditions de l'invariant. Comme $\varnothing \xrightarrow[]{E_1}_{[\![P]\!]} S_1 \xrightarrow[]{E_2}_{[\![P]\!]} \cdots \xrightarrow[]{E_n}_{[\![P]\!]} S_n \in exec^{msr}([\![P]\!])$ est normale et que $n \geqslant 2$, il existe $m < n$ tel que $S_m \rightarrow_R^* S_n$ pour $R = \{$MDOut, MDPub, MDFresh, MDAppl, Fresh$\}$ avec également $\varnothing \xrightarrow[]{E_1}_{[\![P]\!]} S_1 \xrightarrow[]{E_2}_{[\![P]\!]} \cdots \xrightarrow[]{E_m}_{[\![P]\!]} S_m \in exec^{msr}([\![P]\!])$ normale. On applique alors l'hypothèse de récurrence sur cette dernière exécution msr : il existe donc $n' \in \mathbb{N}$ et une fonction croissante $f_p : [\![1; m]\!] \rightarrow [\![0; n']\!]$ tels que les 11 conditions de l'invariant sont vérifiées.

On raisonne alors par disjonction de cas sur la n^e msr appliquée. Sauf mention contraire explicite, on étendra la précédente exécution d'une étape, de $(\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{S}^{MS}_{n'}, \mathcal{P}_{n'}, \mathcal{R}_{n'}, \mathcal{I}_{n'}, \sigma_{n'}, \mathcal{L}_{n'})$ à $(\mathcal{E}_{n'+1}, \mathcal{S}_{n'+1}, \mathcal{S}^{MS}_{n'+1}, \mathcal{P}_{n'+1}, \mathcal{R}_{n'+1}, \mathcal{I}_{n'+1}, \sigma_{n'+1}, \mathcal{L}_{n'+1})$ avant de montrer que toutes les conditions de l'invariant tiennent toujours à rang n. En effet, on notera que $S_n \backslash^{\#} S_m$ ne contient que des faits `Fr` et `!K`, et que $S_m \backslash^{\#} S_n$ ne contient que des faits `Fr` et `Out` : les conditions 3, 4, et 5 de l'invariant sont donc valables même jusqu'au rang $n-1$. Il en va de même pour les conditions 1, 2, 6, 7, 8 et 9 puisque $E_{m+1} = \cdots = E_{n-1} = \varnothing$. De plus, la fonction sera alors définie comme suit :

$$f(i) = \begin{cases} f_p(i) & \text{si } i \in [\![1; m]\!] \\ n' & \text{si } i \in]\!]m, n[\![\\ n'+1 & \text{si } i = n \end{cases}$$

On fixe alors une application telle que $(\mathcal{P}_{n'}, \mathcal{R}_{n'}, \mathcal{I}_{n'}) \leftrightsquigarrow_P S_m$ et on se permettra par la suite de noter $Q \leftrightsquigarrow_P \mathtt{state}_p(\widetilde{t})$ si elle envoie Q sur $\mathtt{state}_p(\widetilde{t})$. Enfin, pour la disjonction de cas qui suivra, on note $l_{linear} =_E S_{n-1} \backslash^{\#} S_n$ et $r = S_n \backslash^{\#} S_{n-1}$; l_{linear} ne contient que des faits linéaires et r des faits linéaires ou permanents, et la donnée de l_{linear}, E_n et r permet d'identifier de manière unique la règle utilisée, qui sera de la forme :

$$[l_{linear}, l_{persistent}]\text{–}[E_n]\rightarrow r$$

avec $l_{persistent} \subset^{\#}_E S_{n-1}$, la seule exception étant $[\![]\text{–}[a]\rightarrow[]; P, p, \widetilde{x}]\!] = [\![\mathtt{event}a; P, p, \widetilde{x}]\!]$. Ce cas est en majorité traité dans l'article. Si la règle R en question est déterminée de manière unique, on se donne $ri \in ginsts(R)$.

- **Cas** $ri = [\mathtt{state}_p(\widetilde{t})] \rightarrow []$

Comme $(\mathcal{P}_{n'}, \mathcal{R}_{n'}, \mathcal{I}_{n'}) \leftrightsquigarrow_P S_{n-1}$, on peut fixer $Q \in^{\#} \mathcal{P}_{n'}$ tel que $Q \leftrightsquigarrow_P \mathtt{state}_p(\widetilde{t})$. Soit alors θ une substitution closant $\mathtt{state}_p(\widetilde{x}) \in prems\left([\![P]\!]_{=p}\right)$ telle que $\widetilde{t} = \widetilde{x}\theta$. θ induit la définition d'une substitution τ et d'un renommage ϱ selon la définition 22. D'après la forme de ri et puisque $Q = P_{|p}\tau\varrho$, on déduit que $Q = 0$, et ce 0 n'est pas indicé par un identifiant de $\mathbf{D}(\mathcal{I}_{f(i-1)})$ puisque sinon, par définition de $[\![P]\!]_{=p}$ la conclusion de ri devrait contenir un fait `exit`. On étend alors la précédente exécution d'une étape ainsi :

$$(\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{S}^{MS}_{n'}, \mathcal{P}_{n'}, \mathcal{R}_{n'}, \mathcal{I}_{n'}, \sigma_{n'}, \mathcal{L}_{n'}) \rightarrow (\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{S}^{MS}_{n'}, \mathcal{P}_{n'} \backslash^{\#} \{0_{id}\}, \mathcal{R}_{n'}, \mathcal{I}_{n'}, \sigma_{n'}, \mathcal{L}_{n'})$$

La vérification des conditions de l'invariant (sauf `4`) qui découle de la remarque 3) est immédiate.

- **Cas** $ri = [\mathtt{state}_p(\widetilde{t})] \rightarrow [\mathtt{exit}_p(id)]$

Comme $(\mathcal{P}_{n'}, \mathcal{R}_{n'}, \mathcal{I}_{n'}) \leftrightsquigarrow_P S_{n-1}$, on peut fixer $Q \in^{\#} \mathcal{P}_{n'}$ tel que $Q \leftrightsquigarrow_P \mathtt{state}_p(\widetilde{t})$. Soit alors θ une substitution closant $\mathtt{state}_p(\widetilde{x}) \in prems\left([\![P]\!]_{=p}\right)$ telle que $\widetilde{t} = \widetilde{x}\theta$. θ induit la définition d'une substitution τ et d'un renommage ϱ selon la définition 22. D'après la forme de ri et puisque $Q = P_{|p}\tau\varrho$,

on déduit que $Q = 0$, et même plus précisément 0_{id} avec $id \in \mathbf{D}(\mathcal{I}_{f(i-1)})$ puisque par définition de $[\![P]\!]_{=p}$ ce 0 a été traduit sous une boucle donc a reçu un identifiant. On étend alors la précédente exécution d'une étape ainsi :

$$(\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{S}^{MS}_{n'}, \mathcal{P}_{n'}, \mathcal{R}_{n'}, \mathcal{I}_{n'}, \sigma_{n'}, \mathcal{L}_{n'}) \to (\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{S}^{MS}_{n'}, \mathcal{P}_{n'} \backslash^{\#} \{0_{id}\}, \mathcal{R}_{n'}, \mathcal{I}_{n'}[id \mapsto (R, N-1)], \sigma_{n'}, \mathcal{L}_{n'})$$
$$\text{où } \mathcal{I}_{n'}(id) = (R, N)$$

La vérification des conditions de l'invariant (sauf `4)` qui découle de la remarque 3) est immédiate.

- **Cas** $ri = [\mathtt{state}_p(\widetilde{t}),\ \mathtt{Fr}(id)] \to [!\mathtt{state}_{p\cdot 1}(\widetilde{t}, id)]$

Comme $(\mathcal{P}_{n'}, \mathcal{R}_{n'}, \mathcal{I}_{n'}) \leftrightsquigarrow_P S_{n-1}$, on peut fixer $Q \in^{\#} \mathcal{P}_{n'}$ tel que $Q \leftrightsquigarrow_P \mathtt{state}_p(\widetilde{t})$. Soit alors θ une substitution closant $\mathtt{state}_p(\widetilde{x}) \in prems\left([\![P]\!]_{=p}\right)$ telle que $\widetilde{t} = \widetilde{x}\theta$. θ induit la définition d'une substitution τ et d'un renommage ϱ selon la définition 22. D'après la forme de ri et puisque $Q = P_{|p}\tau\varrho$, on déduit que $Q = *Q'$ pour un certain processus Q'. Comme le fait $\mathtt{Fr}(id)$ ne peut être généré qu'une fois par exécution, on en déduit que $id \notin \mathbf{D}(\mathcal{I}_{n'})$ et on peut étendre la précédente exécution d'une étape ainsi :

$$(\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{S}^{MS}_{n'}, \mathcal{P}_{n'}, \mathcal{R}_{n'}, \mathcal{I}_{n'}, \sigma_{n'}, \mathcal{L}_{n'}) \to (\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{S}^{MS}_{n'}, \mathcal{P}_{n'} \backslash^{\#} \{Q\} \cup^{\#} \{Q'\}, \mathcal{R}_{n'}, \mathcal{I}_{n'}[id \mapsto (Q', 1)], \sigma_{n'}, \mathcal{L}_{n'})$$

La vérification de toutes les conditions de l'invariant sauf `4)` et `8)` est immédiate, et pour cette dernière il suffit de prolonger l'implication induite par $(\mathcal{P}_{n'}, \mathcal{R}_{n'}, \mathcal{I}_{n'}) \leftrightsquigarrow_P S_{n-1}$ (restreinte à l'aide de la remarque 3) avec $Q' \leftrightsquigarrow_P !\mathtt{state}_{p\cdot 1}(\widetilde{t}, id)$ et $fst(\mathcal{I}_{n'+1}(id)) \leftrightsquigarrow_P !\mathtt{state}_{p\cdot 1}(\widetilde{t}, id)$. `8)` en découle alors immédiatement.

- **Cas** $ri = [\mathtt{exit}_{pos}(id),\ pos \in \Pi] \to [\mathtt{exit}_p(id)]$ pour $\Pi \in pos(P)^*$

Dans ce cas, on n'étend pas l'exécution. Autrement dit, $f(i) = \begin{cases} f_p(i) & \text{si } i \in [\![1; m]\!] \\ n' & \text{si } i \in]\!]m, n]\!] \end{cases}$

La vérification des conditions de l'invariant découle directement de l'hypothèse de récurrence.

- **Cas** $ri = [\mathtt{exit}_p(id)]$–[Finish$(id)$]→[]

Comme $\mathtt{exit}_p(id) \in^{\#} E_i$, d'après le lemme B.3, S_{n-1} ne contient aucun autre fait `exit` et uniquement des `state` permanents (sauf s'ils ne contiennent pas l'identifiant id). En particulier, l'absence de faits `exit` rend impossible la présence d'un Finish(id) préalable sans Reboot(id) intercalaire. Or, par définition de $[\![P]\!]_{=p}$, id a été généré par un fait de la forme $[\mathtt{state}_{p'}(\widetilde{x}), \mathtt{Fr}(id)] \to [!\mathtt{state}_{p'\cdot 1}(\widetilde{x})]$ avec $id \in \widetilde{x}$, donc on a $!\mathtt{state}_{p'\cdot 1}(\widetilde{x}) \in S_{n-1}$ puisque ce fait est permanent ; la définition de $[\![P]\!]$ en corrélation avec la syntaxe (qui interdit les réplications et les itérations à une position ayant $p' \cdot 1$ comme préfixe strict) montre que tout fait $\mathtt{state}_{p''}(\widetilde{y})$ avec $p' \sqsubset p''$ est linéaire. S_{n-1} contient donc un unique fait `state` avec l'identifiant id, et il s'agit de $!\mathtt{state}_{p'\cdot 1}(\widetilde{x})$. **(⋆)**

D'après `10)` au rang $n-1$, comme on a dit qu'aucune action Finish(id) ne pouvait être présente depuis le dernier Reboot(id) (qui était, disons, dans E_{i_0}) et que $fst(\mathcal{I}_{f(i_0)}(id)) = fst(\mathcal{I}_{f(n-1)}(id))$ par définition de la sémantique, tout processus indicé par id est un sous-processus strict de $fst(\mathcal{I}_{f(n-1)}(id))$. Comme d'après `9)` on a $fst(\mathcal{I}_{f(n-1)}(id)) \leftrightsquigarrow_P !\mathtt{state}_{p'\cdot 1}(\widetilde{x})$, $\mathcal{P}_{n'}$ ne contient aucun processus indicé par id : en effet, par définition de $\leftrightsquigarrow_P$, si $Q_1 < Q_2$, $Q_1 \leftrightsquigarrow_P \mathtt{state}_{p_1}(\widetilde{x}_1)$ et $Q_2 \leftrightsquigarrow_P \mathtt{state}_{p_2}(\widetilde{x}_2)$, alors $p_2 \sqsubset p_1$. Un processus indicé par id devrait donc être en relation avec un $\mathtt{state}_{p''}(\widetilde{y})$ où $p' \sqsubset p''$

(et $id \in \widetilde{y}$ d'après `11)`), ce qui est impossible (d'après **(⋆)**). En particulier, d'après le lemme B.2, cela signifie que $snd(\mathcal{I}_{f(n-1)}(id)) = 0$. On peut donc étendre la précédente exécution par :

$$(\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{S}^{MS}_{n'}, \mathcal{P}_{n'}, \mathcal{R}_{n'}, \mathcal{I}_{n'}, \sigma_{n'}, \mathcal{L}_{n'}) \to (\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{S}^{MS}_{n'}, \mathcal{P}_{n'} \cup^{\#} \{Q_{id}\}, \mathcal{R}_{n'} \cup^{\#} \{Q'\}, \mathcal{I}_{n'}[id \mapsto (Q,1)], \sigma_{n'}, \mathcal{L}_{n'})$$
$$\text{où } \mathcal{I}_{n-1}(id) = (Q,0)$$

La vérification de toutes les conditions de l'invariant sauf `4)` et `8)` au rang n est immédiate. Pour `4)`, il suffit d'étendre l'application définie par $(\mathcal{P}_{n'}, \mathcal{R}_{n'}, \mathcal{I}_{n'}) \leftrightsquigarrow_P S_{n-1}$ avec $Q_{id} \leftrightsquigarrow_P !\texttt{state}_{p'\cdot 1}(\widetilde{x})$ (cela est possible car vu ce qui précède, $!\texttt{state}_{p'\cdot 1}(\widetilde{x})$ avait $fst(\mathcal{I}_{n-1}(id))$ comme unique antécédent pour $\leftrightsquigarrow_P$ au rang $n-1$). La vérification de `8)` en découle alors.

- **Cas** $ri = [\texttt{state}_p(id)] \to [!\texttt{state}_{p\cdot 1}(\widetilde{t})]$

Comme $(\mathcal{P}_{n'}, \mathcal{R}_{n'}, \mathcal{I}_{n'}) \leftrightsquigarrow_P S_{n-1}$, on peut fixer $Q \in^{\#} \mathcal{P}_{n'}$ tel que $Q \leftrightsquigarrow_P \texttt{state}_p(\widetilde{t})$. Soit alors θ une substitution closant $\texttt{state}_p(\widetilde{x}) \in prems\left([\![P]\!]_{=p}\right)$ telle que $\widetilde{t} = \widetilde{x}\theta$. θ induit la définition d'une substitution τ et d'un renommage ϱ selon la définition 22. D'après la forme de ri et puisque $Q = P_{|p}\tau\varrho$, on déduit que $Q = !Q'$ pour un certain processus Q'. On étend l'exécution d'une étape ainsi :

$$(\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{S}^{MS}_{n'}, \mathcal{P}_{n'}, \mathcal{R}_{n'}, \mathcal{I}_{n'}, \sigma_{n'}, \mathcal{L}_{n'}) \to (\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{S}^{MS}_{n'}, \mathcal{P}_{n'} \backslash^{\#} \{Q\}, \mathcal{R}_{n'} \cup^{\#} \{Q'\}, \mathcal{I}_{n'}, \sigma_{n'}, \mathcal{L}_{n'})$$

La vérification de toutes les conditions de l'invariant est immédiate, sauf `4)` qui n'en reste pas moins plutôt simple puisqu'il suffit de modifier l'application définie par $(\mathcal{P}_{n'}, \mathcal{R}_{n'}, \mathcal{I}_{n'}) \leftrightsquigarrow_P S_{n-1}$ en remplaçant $Q \leftrightsquigarrow_P \texttt{state}_p(\widetilde{t})$ par $Q' \leftrightsquigarrow_P !\texttt{state}_{p\cdot 1}(\widetilde{t})$.

- **Cas** $ri = [!\texttt{state}_p(\widetilde{t}), \ell]\text{–}[\text{Reboot}(id), a]\to r$

Par définition de $[\![P]\!]_{=p}$ et puisque le processus est bien formé, on a $id \in \widetilde{t} \cap \mathbf{D}(\mathcal{I}_{n-1})$. Puisque par hypothèse, l'exécution msr étudiée satisfait α_{loop}, on peut fixer i_0 tel que $\text{Reboot}(id) \notin E_j$ si $i_0 \leqslant j < n$ et qu'on ait un $i_1 \in [\![i_0; n-1]\!]$ tel que $\text{Finish}(id) \in E_{i_1}$. D'après `9)`, on peut donc appliquer `8)` : on a ainsi $Q_{id} \in \mathcal{P}_{n-1}$ tel que $Q_{id} \leftrightsquigarrow_P !\texttt{state}_{p'\cdot 1}(\widetilde{x})$.

Il suffit alors d'utiliser la même transition que le cas $[\texttt{state}_p(\widetilde{t}), \ell]\text{–}[a]\to r$. La justification de maintien de l'invariant au rang n est identique, sauf pour `4)` où l'argument autour de $\texttt{state}_p(\widetilde{t})$ est différent (puisqu'ici il est permanent) : vu le paragraphe précédent, on a $fst(\mathcal{I}_n(id)) = fst(\mathcal{I}_{n-1}(id)) \leftrightsquigarrow_P !\texttt{state}_p(\widetilde{t})$. Donc on était dans la configuration de la définition de $\leftrightsquigarrow_P$ où $!\texttt{state}_p(\widetilde{t})$ a exactement deux antécédents dont un dans $fst(\text{Im}\,(\mathcal{I}_n))$. $!\texttt{state}_p(\widetilde{t})$ a donc un unique antécédent dans la nouvelle définition de l'application associée à $\leftrightsquigarrow_P$ et on vérifie donc bien la définition.

- **Cas** $ri = [!\texttt{state}_p(\widetilde{t}), \ell]\text{–}[a]\to r$, où a ne contient pas d'action Reboot

On se permet, comme expliqué au début de la démonstration, de supposer qu'on étend la précédente exécution msr de deux applications de règles, dont la première est $[!\texttt{state}_p(\widetilde{t})] \to [!\texttt{state}_p(\widetilde{t})]$. Comme a ne contient pas d'action Reboot et que, vu la définition de $[\![P]\!]$ et de la syntaxe, aucun fait permanent ne peut être produit pour un sous processus strict d'une boucle, on en déduit que $\widetilde{t} \cap \mathbf{D}(\mathcal{I}_{n-1}) = \varnothing$. D'après `9)`, on a donc $Q \in \mathcal{R}_{n-1}$ tel que $Q \leftrightsquigarrow_P !\texttt{state}_p(\widetilde{x})$. On étend alors la précédente exécution par l'étape :

$$(\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{S}^{MS}_{n'}, \mathcal{P}_{n'}, \mathcal{R}_{n'}, \mathcal{I}_{n'}, \sigma_{n'}, \mathcal{L}_{n'}) \to (\mathcal{E}_{n'}, \mathcal{S}_{n'}, \mathcal{S}^{MS}_{n'}, \mathcal{P}_{n'} \cup^{\#} \{Q\}, \mathcal{R}_{n'} \cup^{\#} \{Q'\}, \mathcal{I}_{n'}, \sigma_{n'}, \mathcal{L}_{n'})$$

et on poursuit avec la construction définie dans le cas $ri = [\mathtt{state}_p(\widetilde{t}), \ell]\text{–}[a]\text{→}r$. On définit alors $f : [\![1; n+1]\!] \to [\![0; n'+2]\!]$ par :

$$f : \begin{cases} f_p(i) & \text{si } i \in [\![1; m]\!] \\ n' & \text{si } i \in]\!]m, n[\![\\ n'+1 & \text{si } i = n \\ n'+2 & \text{sinon} \end{cases}$$

La justification de maintien de l'invariant au rang n est immédiate, et celle au rang $n+1$ est identique à celle du cas $ri = [\mathtt{state}_p(\widetilde{t}), \ell]\text{–}[a]\text{→}r$, sauf pour `4)` où l'argument autour de $\mathtt{state}_p(\widetilde{t})$ est différent (puisqu'ici il est permanent) : $Q \in \mathcal{R}_{n-1} \subset^{\#} \mathcal{R}_{n+1}$, et $Q \leftrightsquigarrow_P !\mathtt{state}_p(\widetilde{t})$. Donc on était dans la configuration de la définition de $\leftrightsquigarrow_P$ où $!\mathtt{state}_p(\widetilde{t})$ a un nombre arbitraire d'antécédents dont exactement un dans $\mathcal{R}_{n+1}$. La nouvelle application associée à $\leftrightsquigarrow_P$ et vérifie donc bien la définition.

- Les autres cas sont ceux de l'article (si on fait abstraction de $ri = [\mathtt{state}_p(\widetilde{t})] \to []$ qui a été modifié et $ri = [!\mathtt{state}_p(\widetilde{t})] \to [\mathtt{state}_{p\cdot 1}(\widetilde{t})]$ qui ne suivait pas la définition).La démonstration de conservation des sept premières conditions de l'invariant reste de rigueur malgré le nouveau formalisme puisqu'on a adapté la remarque 3, et les conditions 8, 9, 10 et 11 s'obtiennent immédiatement avec l'hypothèse de récurrence (éventuellement, on peut ajouter pour `10)` que les seuls processus indicés ajoutés au fil des transitions sont des sous-processus de ceux déjà présents dans $\mathcal{P}_{f(i)}$). Cela achève donc la démonstration de ce lemme.

□