Logique de séparation concurrente : une preuve de correction en théorie des jeux

Léo Stefanesco, stage encadré par Paul-André Melliès

17 septembre 2016

1 Introduction

Les raisonnements sur les programmes concurrents à mémoire partagés sont souvent subtils, car les différents fils d'exécutions peuvent interférer. Les fils du programme doivent alors se synchroniser, afin de contrôler ces interférences. Lesquelles peuvent aussi entraîner des types d'erreurs qui sont propres aux programmes concurrents, comme les *deadlock* — lorsque deux fils sont en train d'attendre que l'autre termine quelque chose — et les *accès concurrents* — lorsque qu'un fil écrit dans un emplacement mémoire qui est en train d'être utilisé par un autre fil. Plusieurs logiques ont été proposées depuis les années 1970, parmi lesquelles la méthode de Owicki-Gries[10], puis les méthodes de *rely-guarantee*[8]. La *logique de séparation concurrente* est l'une des plus prometteuses.

La logique de séparation concurrente est une extension de la logique de séparation. Cette dernière, inventée au tournant du siècle par John Reynolds, Peter O'Hearn, Samin Ishtiaq et Hongseok Yang[13] ajoute à la logique de Hoare un nouveau connecteur logique, le produit de séparation, noté $*$, qui permet d'exprimer que deux propriétés sont vérifiées par deux parties disjointes de la mémoire. La règle emblématique de la logique de séparation, et qui repose sur son produit, est la «règle de frame» :

$$\frac{\vdash \{P\}C\{Q\}}{\vdash \{P * R\}C\{Q * R\}}$$

ce qui permet de raisonner compositionnellement, en oubliant la partie de la mémoire qui ne concerne pas la commande C.

Peter O'Hearn remarqua que la logique de séparation pouvait être adaptée afin de vérifier des programmes concurrents, en utilisant le fait que deux programmes qui opèrent sur deux partions séparées de la mémoire ne peuvent interférer. Cela permet de raisonner sur la mise en parallèle de deux programmes de manière compositionnelle. L'utilité de la logique de séparation découle du fait que, dans les programmes concurrents, le danger vient des parties de la mémoire auxquelles deux processus concurrents peuvent accéder

en même temps, avec au moins l'un de ces deux accès en écriture, ce que le produit de séparation peut exprimer élégamment. Les deux processus communiquent alors grâce à des ressources partagées, décrites par des prédicats. On peut donc voir les prédicats de la logique comme des ressources qui sont échangées par les processus, ce qui permet de formaliser le concept informel de possession (*ownership*) de la mémoire. Une des difficultés conceptuelles est de définir une logique de séparation concurrente qui soit correcte, et d'établir rigoureusement sa correction. Dans un entretien[1] qui a suivi la remise du Prix Gödel en mai 2016, Peter O'Hearn raconte que devant la difficulté technique et conceptuelle de la tache, il fit appel à Stephen Brookes pour prouver la correction de sa logique. Il donna une démonstration en 2004[5], en utilisant une sémantique dénotationnelle où les commandes sont interprétées par un ensemble de traces. Brookes explique dans un article publié sept ans plus tard[4] que sa logique était incorrecte à cause de conditions de bord erronées sur les variables. Il y a donc une difficulté inhérente à définir une bonne logique de séparation concurrente et à établir sa preuve.

Il existe aujourd'hui de nombreuses preuves de correction, notamment par François Pottier[12], qui la ramène de manière syntaxique à un théorème de *subject reduction*, et par Viktor Vafeiadis[14], qui développe une méthode sémantique (opérationnelle cette fois), à l'instar de Brookes.

Nous mettons en place dans ce rapport de stage une nouvelle méthode de démonstration de la correction de la logique de séparation concurrente. Cette méthode est inspirée par la sémantique des jeux des langages de programmation, et elle a pour avantage d'être formulée dans le même cadre de théorie des jeux que celui utilisé pour établir la correction des logiques de spécification telle que la logique monadique du second ordre (MSO) ou le mu-calcul modal. La démonstration s'appuie sur une sémantique dénotationnelle et interactive du langage impératif et concurrent. Dans cette sémantique, chaque programme du langage est interprété par un ensemble de traces d'exécution, où chaque trace alterne entre une transition du code et une transition de l'environnement. Cette interprétation interactive permet de formaliser de manière claire l'idée fondamentale de "possession" qui gouverne la logique de séparation concurrente : si un joueur "possède" de la mémoire, il est le seul à pouvoir la modifier.

Pour formaliser cette notion de possession dans notre cadre sémantique, nous définissons un "jeu de séparation" où la mémoire est coupée en deux parties, la partie qui appartient au code, et la partie qui appartient à l'environnement. Chacun des deux joueurs Adam et Eve ne peut changer que la partie qui lui appartient. Lors de l'interaction entre code et environnement définie par une trace d'exécution, Adam et Eve s'assignent pour tache de séparer l'espace mémoire et de justifier le triplet de Hoare PCQ donné par la logique de séparation. Dans notre approche, le théorème de correction de la logique de séparation concurrente se ramène à la traduction d'un arbre de dérivation de la logique en une stratégie gagnante pour Eve dans

ce jeu de séparation. Comme nous l'expliquerons à la fin du rapport, notre approche permet d'établir de manière conceptuellement rigoureuse l'absence d'accès concurrents dans un programme vérifié par la logique de séparation concurrente.

Plan du rapport Après avoir défini le langage impératif et concurrent sur lequel on se repose (section 2) et avoir introduit la logique de séparation concurrente (sections 3.1 et 3.2), nous mettons en place notre formalisme basé sur les jeux (section 3.3). Ensuite, nous prouvons la correction de la logique (sections 3.4 à 3.10). Enfin, avant de conclure, nous traitons l'aspect concurrent et les accès concurrents dans la section 4.

2 Un langage impératif concurrent et sa sémantique

2.1 Un langage impératif concurrent

Grammaire On utilise un langage impératif concurrent très simple, dérivé de celui utilisé dans [14]. Le déréférencement d'une adresse est noté $[E]$, et la mise en parallèle de deux programmes est notés $C_1 \parallel C_2$. Sa grammaire est la suivante :

$$
\begin{aligned}
B &::= \mathbf{true} \mid \mathbf{false} \mid B \wedge B \mid B \vee B \mid E = E' \\
E &::= 0 \mid 1 \mid \cdots \mid x \mid E + E \mid E - E \mid E * E \\
C &::= x \coloneqq E \mid x \coloneqq [E] \mid [E] \coloneqq [E] \\
&\quad \mid \mathbf{while}\, B\, \mathbf{do}\, C \mid \mathbf{select}[S_1, S_2](C_1, C_2) \\
&\quad \mid \mathbf{resource}\, r\, \mathbf{in}\, C \mid \mathbf{when}\, B\, \mathbf{with}\, r\, \mathbf{do}\, C \\
&\quad \mid C; C \mid C \parallel C
\end{aligned}
$$

où $r \in \mathrm{Res}$, $x \in \mathrm{Var}$, des ensembles (disjoints) infini dénombrable de noms, et S_1, S_2 sont des prédicats de la logique de séparation, dont la syntaxe est décrite dans la section 3.1.

Définitions préliminaires La mémoire est composée de deux parties : le tas et la pile. La pile associe aux variables leurs valeurs, et le tas associe aux adresses des valeurs. Dans un souci de régularité, on s'efforcera de traiter ces deux sortes de mémoires de manière uniforme. Pour les traiter de manière uniforme tout en étant capable de prouver autant de programmes qu'avec des prémisses auxiliaires de la forme «les variables écrites par la commande C_1 ne sont ni lues ni écrites par C_2» dans les règles d'inférence, il est nécessaire d'utiliser des permissions. Elles permettent d'internaliser ces conditions dans la logique de séparation[2, 11]. Le sujet sera abordé avec plus de détail dans

la section 3.1. Pour l'instant, il suffit de savoir que les permissions forment un monoïde commutatif partiel.

Définition 1 (Les états)**.** *On fixe* Loc *un ensemble infini d'adresses mémoires,* Val *un ensemble de valeurs et* $\mathbb{P}$ *un monoïde des permissions, avec* $\text{Loc} \subseteq \text{Val}$ *et* $\mathbb{N} \subseteq \text{Val}$. Un état *est la donnée de* $\sigma = (s, h)$*, où la pile* s *est une fonction partielle* $s : \text{Var} \rightharpoonup_{fin} \text{Val} \times \mathbb{P}$ *et le tas* $h : \text{Loc} \rightharpoonup_{fin} \text{Val} \times \mathbb{P}$ *(où* $A \rightharpoonup_{fin} B$ *dénote une fonction partielle finie de* A *vers* B*). On note* $\text{vdom}(\sigma)$ *le domaine de définition de* s*, et* $\text{hdom}(\sigma)$ *celui de* h*.*

L'opération la plus importante sur les états est le produit $*$, qui généralise l'union disjointe en permettant le partage de variables et d'adresses, grâce aux permissions.

Définition 2 (Produit d'états)**.** *Soient* $\sigma_1 = (s_1, h_1)$ *et* $\sigma_2 = (s_2, h_2)$ *deux états.*

L'état produit est défini si :

> *pour tout* $l \in \text{dom}(h_1) \cap \text{dom}(h_2)$*, si on note* $h_1(l) = (v_1, p_1)$ *et* $h_2(l) = (v_2, p_2)$*,* $v_1 = v_2$ *et* $p_1 \cdot p_2$ *est défini, et de même pour toutes les variables* x *et les piles* s_1 *et* s_2*.*

La fonction partielle h *de leur produit est une fonction partielle définie sur* $\text{dom}(t_1) \cup \text{dom}\,(t_2)$ *:*

$$l \mapsto \begin{cases} h_1(l) & \text{si } l \in \text{dom}(h_1) \text{ et } l \notin \text{dom}(h_2) \\ h_2(l) & \text{si } l \in \text{dom}(h_2) \text{ et } l \notin \text{dom}(h_1) \\ (v, p) & \text{si } h_1(l) = (v, p_1),\ h_2(l) = (v, p_2) \text{ et } p = p_1 \cdot p_2 \end{cases}$$

et de même pour la pile.

Il est aussi souvent utile de parler de sous-état, mais la définition naïve d'inclusion des graphes de s et h n'est pas satisfaisante, car elle ne permet pas de dire que deux processus peuvent lire une même variable en même temps, on définit donc l'inclusion d'état par :

$$\sigma \subseteq \sigma' \overset{\text{def}}{\iff} \exists \sigma_1.\ \sigma' = \sigma * \sigma_1$$

Les opérations arithmétiques E induisent une fonction d'évaluation $E : \text{Etat} \to \text{Val} + \textbf{abort}$, qui échoue lorsqu'une variable libre de E n'est pas dans le domaine de la pile ($\text{fv}(E) \not\subseteq \text{vdom}(\sigma)$), ce qui est dénoté par **abort**. Cette fonction est définie de manière évidente.

2.2 Instructions élémentaires

La sémantique d'une commande de notre langage sera exprimée sous la forme d'un ensemble de traces composées d'instructions élémentaires, qui sont de la forme suivante :

— $x \coloneqq E$
— $x \coloneqq [E]$
— $[E] \coloneqq E$
— $P(r)$
— $V(r)$
— **nop**
— $x \coloneqq \mathbf{alloc}(E)$
— $\mathbf{dispose}(E)$

Les instructions $P(r)$ et $V(r)$ servent à prendre et à rendre le verrou r.

Nous aurons aussi besoin d'une relation d'évaluation des instructions élémentaires, afin de pouvoir parler d'une instruction qui échoue.

Définition 3. *Soit m une instruction élémentaire, et σ un état. On définit la relation $\sigma \overset{m}{\rightsquigarrow} \sigma'$, qui exprime l'action d'une instruction élémentaire sur le tas. Nous donnons les principales règles ci-dessous.*

$$\frac{E(\sigma) = v}{\sigma \overset{x:=E}{\rightsquigarrow} \sigma[x \coloneqq v]} \qquad \frac{E(\sigma) = \textbf{\textit{abort}}}{\sigma \overset{x:=E}{\rightsquigarrow} \textbf{\textit{abort}}}$$

$$\frac{E_1(\sigma) = l \quad E_2(\sigma) = v \quad l \in \mathrm{hdom}(\sigma)}{\sigma \overset{[E_1]:=E_2}{\rightsquigarrow} \sigma[l \coloneqq v]}$$

$$\frac{E_1(\sigma) = \textbf{\textit{abort}} \;\; ou \;\; E_2(\sigma) = \textbf{\textit{abort}} \;\; ou \;\; E_1(\sigma) \notin \mathrm{hdom}(\sigma)}{\sigma \overset{[E_1]:=E_2}{\rightsquigarrow} \textbf{\textit{abort}}}$$

$$\frac{E(\sigma) = v}{\sigma \overset{x:=\textbf{\textit{alloc}}(E)}{\rightsquigarrow} \sigma[x := l] \uplus [l := v]}$$

Si $x \in \mathrm{dom}(\sigma)$, $\sigma[x := v]$ est la fonction partielle qui se comporte comme σ, sauf en x où elle vaut v. Si $l \notin \mathrm{dom}(\sigma)$, on note l'ajout d'une nouvelle paire (x, v) à une fonction partielle $\sigma \uplus [l := v]$.

On définit, pour chaque instruction élémentaire m, $\mathrm{lock}^+(m)$ et $\mathrm{lock}^-(m)$ par $\mathrm{lock}^+(P(r)) = \{r\}$, $\mathrm{lock}^-(V(r)) = \{r\}$, et $\mathrm{lock}^+(m) = \mathrm{lock}^-(m) = \emptyset$ pour les autres instructions.

2.3 Sémantique d'une commande

Nous allons définir la sémantique de notre langage en le compilant vers un ensemble de traces d'exécutions, dans chacune desquelles l'environnement peut agir. Chaque trace commence par une transition de l'environnement, et ensuite le code et l'environnement jouent un coup chacun. Un coup de l'environnement modélise un nombre arbitraire d'instructions, c'est pourquoi il peut modifier l'état à sa guise en une seule transition.

Plus précisément, une *trace* est un chemin dont les sommets sont des états, de taille paire ou infinie, alternant entre une transition de l'environnement et une transition du code, et où l'environnement commence. On les

note :

$$\sigma_1 \xrightarrow{env} \sigma_2 \xrightarrow{m_1} \sigma_3 \xrightarrow{env} \sigma_4 \xrightarrow{m_2} \dots$$

Pour les définitions qui suivent, nous utiliserons la définition équivalente suivante :

Définition 4 (Trace)**.** *Soit $n \in \mathbb{N} \cup \{\infty\}$ et $\bar{t}$ une application*

$$\bar{t} : \{1 \dots n\} \to \text{Etat} \times \text{Instr} \times \text{Etat}$$

telle que pour tout $i \in \{1 \dots n\}$, $f(i) = (\sigma, m, \sigma')$ vérifie $\sigma \rightsquigarrow^{m} \sigma'$.

Une trace *est un couple $(\sigma_1, \bar{t})$, avec $\sigma_1 \in$ Etat. Elle est dite* finie *ou* infinie *selon que n l'est.*

Si t est une trace, on appelle n sa taille, qu'on note $|t|$. Les deux définitions sont équivalentes, la trace suivante

$$\sigma_1 \xrightarrow{env} \sigma_2 \xrightarrow{m_1} \sigma_3 \xrightarrow{env} \sigma_4 \xrightarrow{m_2} \dots$$

correspond à $(\sigma_1, \bar{t})$, avec $\bar{t}(i) = (\sigma_{2i}, m_i, \sigma_{2i+1})$.

Définition 5 (Système de transition)**.** Un système de transition T *est la donnée de $(|T|, (T_F, T_A, T_{DNF}))$, où $|T|$ est un ensemble de traces, et (T_F, T_A, T_{DNF}) est un triplet de parties de $|T|$, qui déterminent le* statut *des traces, respectivement les traces qui terminent, celles qui aboutissent à une erreur, et les traces qui ne terminent pas (soit à cause d'un* deadlock*, soit à cause d'une boucle infinie).*

On définit deux opérations sur les systèmes de transitions qu'on vient de définir, qui serviront à définir la composition séquentielle et la composition parallèle de deux commandes.

Définition 6 (Concaténation)**.** *Soient T et T' deux systèmes de transitions. On appelle $T \cdot T'$ l'ensemble de traces enrichi dont l'ensemble sous-jacent est $\{t \cdot t' \mid t \in T \text{ et } t' \in T'\}$, où $t \cdot t'$ est défini par :*

- ***Si** t **ne termine pas,** alors $t \cdot t' = t$ et $t \cdot t'$ a le même statut que t.*
- ***Si** t **termine,** en notant $t = (\sigma_1, \bar{t})$ et $t' = (\sigma'_1, \overline{t'})$, $t \cdot t'$ n'est défini que lorsque $\bar{t}(|t|)$ est de la forme $(\_, \_, \sigma'_1)$. Dans ce cas, $t \cdot t' = (\sigma_1, \overline{t''})$, avec $\overline{t''}$ défini sur $\{1 \dots n + n'\}$ par*

$$\overline{t''}(i) = \begin{cases} \bar{t}(i) & \text{si } i \leq n \\ \overline{t'}(n+i) & \text{sinon} \end{cases}$$

Entrelacements Soient t, t' deux traces. On appelle *ordonnancement de t et t'* une bijection croissante

$$\tau : \{1 \dots |t|\} + \{1 \cdots |t'|\} \to \{1 \dots |t| + |t'|\}$$

On note ι_l et ι_r les injections canoniques dans la somme.

Définition 7 (Entrelacement, version préliminaire)**.** *Soient $T = (|T|, (T_F, T_A, T_{DNF}))$ et $T' = (|T'|, (T'_F, T'_A, T'_{DNF}))$ deux systèmes de transitions. L'ensemble sous-jacent du* produit parallèle (préliminaire) de T et de T' *est :*

$$\left|T \parallel T'\right| = \bigcup_{t \in |T|, t' \in |T'|} t \parallel t'$$

où le produit parallèle de deux traces est décrit ci-dessous.

Soient $t = (\sigma_1, \bar{t}), t' = (\sigma'_1, \overline{t'})$ deux traces, et τ un ordonnancement de t et de t'. L'entrelacement $\tau(t, t')$ de t et de t' n'est défini que si $\sigma_1 = \sigma'_1$.

Dans ce cas, cet état initial commun est celui de l'entrelacement, et sa fonction sous-jacente est définie grâce à :

$$f(i) = \begin{cases} \bar{t}(j) & si\ \tau^{-1}(i) = \iota_l(j) \\ \bar{t'}(j) & si\ \tau^{-1}(i) = \iota_r(j) \end{cases}$$

Dans le cas où l'une des deux traces ne termine pas normalement, il faut couper f au niveau du premier problème. Posons $|t|_C = |t|$ si t ne termine pas normalement ($t \in T_A \cup T_{DNF}$) et $|t| = \infty$ sinon (idem pour t'). Alors on pose $i_C = \min(\tau(\iota_l(|t|_C)), \tau(\iota_r(|t'|_C)))$. L'entrelacement est alors $(\sigma_1, f_{|i_C})$, et son statut est le même que la trace qui a donnée sa valeur à i_C.

Il manque quelque chose à cette définition : elle ne respecte pas la discipline des verrous. Le théorème de correction de notre logique doit nous garantir que le programme ne rencontre pas d'erreur pendant son exécution, même s'il va bloquer, simplement éliminer les traces qui ne respectent pas cette discipline n'est donc pas une solution satisfaisante. Il faut donc les couper au moment où elles tentent de prendre un verrou qu'elles ont déjà pris (au moment où elles *deadlock*).

Pour formaliser cela, on utilise une fonction auxiliaire, définie pour $i \in \{1 \dots |\tilde{t}|\}$, et qui indique, à chaque point de la trace, quels sont les verrous qui sont pris (par le code), ou bien `deadlock` si le code tente de prendre un verrou qui est déjà pris :

$$\alpha(i) = \begin{cases} \emptyset & \text{si } i = 0 \\ \texttt{deadlock} & \text{si } \mathrm{lock}^+(m_i) \cap \alpha(i-1) \neq \emptyset \text{ ou } \mathrm{lock}^-(m_i) \not\subseteq \alpha(i-1) \\ \alpha(i-1) \setminus \mathrm{lock}^-(m_i) \cup \mathrm{lock}^+(m_i) & \text{sinon} \end{cases}$$

On peut alors définir une trace qui respecte la discipline induite par les verrous.

Définition 8 (Trace disciplinée)**.** *Soit t une trace. Soit $i_0 = min\{i \mid \alpha(i) = \texttt{deadlock}\}$. Si $i_0 \in \mathbb{N}$, la trace disciplinée de t ne termine pas, et elle est égale au préfixe de t qui est de taille i_0, plus formellement, si $t = (\sigma_1, \bar{t})$, la trace disciplinée de t est $(\sigma_1, \bar{t}_{|i_0-1})$, si $i_0 = \infty$, alors son statut est le même que t, sinon elle est dans T_{DNF}.*

On peut alors enfin définir l'entrelacement de deux systèmes de transitions.

Définition 9 (Entrelacement)**.** *Soient T, T' deux systèmes de transitions. Notons $\tilde{T}$ leur entrelacement préliminaire. Alors l'*entrelacement de T et de T' *est le système de transitions T où on a remplacé chaque trace par sa trace disciplinée.*

Sémantique par traces des commandes La sémantique $[\![C]\!]$ de la commande C est un système de transitions, que nous allons définir par induction sur la structure de C.

On peut définir les commandes qui correspondent directement à des instructions élémentaires à partir de la relation $\sigma \overset{m}{\rightsquigarrow} \sigma'$: si C correspond à m (ie. $C = x \coloneqq E$ et $m = x \coloneqq E$, ou bien **skip** et **nop**), on définit :

$$[\![C]\!] = \{\sigma_1 \xrightarrow{env} \sigma_2 \xrightarrow{m} \sigma_3 \xrightarrow{env} \sigma_4 \mid \ \sigma_2 \overset{m}{\rightsquigarrow} \sigma_3 \ \} \cup$$
$$\{\sigma_1 \to \sigma_2 \mid \ \sigma_2 \overset{m}{\rightsquigarrow} \textbf{abort} \ \}$$

où les traces du premier ensemble terminent (sont dans T_F), et celles du second sont dans T_A.

Pour définir les traces de certaines commandes simples, on définit, pour tout prédicat P sur les paires d'états, les deux fonctions auxiliaires suivantes :

$$\textbf{succès}_m(P) = \{\sigma_1 \xrightarrow{env} \sigma_2 \xrightarrow{m} \sigma_3 \xrightarrow{env} \sigma_4, \text{ pour tout } \sigma_1, \sigma_2, \sigma_3, \sigma_4 \text{ tels que } P(\sigma_2, \sigma_3)\}$$
$$\textbf{bloqué}(P) = \{\sigma_1 \xrightarrow{env} \sigma_2, \text{ pour tout } \sigma_1, \sigma_2 \text{ tels que } \neg P(\sigma_2)\}$$

La différence étant que les traces de $\textbf{succès}_m(P)$ sont dans la partie T_F, et celles de $\textbf{bloqué}(P)$ sont dans T_{DNF}.

La sémantique des autres commandes est :

- $[\![C \parallel C']\!] \coloneqq [\![C]\!] * [\![C']\!]$,
- $[\![C; C']\!] \coloneqq [\![C]\!] \cdot [\![C']\!]$,
- $[\![\textbf{when } B \textbf{ with } r \textbf{ do } C]\!] = \textbf{succès}_{P(r)}(\sigma_2 = \sigma_3 \wedge \sigma_2 \vDash B) \cdot [\![C]\!] \cdot \textbf{succès}_{V(r)}(\sigma_2 = \sigma_3)$,
- $[\![\textbf{resource } r \textbf{ in } C]\!] = \{\text{cache}_r(t) \mid t \in [\![C]\!]\}$, où $\text{cache}_r(t)$ est égale à la trace t où on a remplacé tous les $P(r)$ et les $V(r)$ par des **nop**,

— $[\![\mathbf{select}[S_1, S_2](C_1, C_2)]\!] \coloneqq [\![S_1]\!] \cdot [\![C_1]\!] \cup [\![S_2]\!] \cdot [\![C_2]\!] \cup \mathbf{bloqué}\{\neg\sigma_2 \vDash S_1 \wedge \neg\sigma_2 \vDash S_2\}$, où $[\![S]\!] \coloneqq \{\sigma_1 \xrightarrow{env} \sigma_2 \xrightarrow{\mathbf{nop}} \sigma_2 \xrightarrow{env} \sigma_3,$ pour tout $\sigma_1, \sigma_2, \sigma_3$ avec que $\sigma_2 \vDash S\}$,

— $[\![\mathbf{while}\, B\, \mathbf{do}\, C]\!] \coloneqq Y(\lambda x.[\![B]\!] \cdot [\![C]\!] \cdot x)$, où Y est l'opérateur de plus petit point fixe.

La structure de contrôle **if** est alors du sucre syntaxique : on définit $\mathbf{if}\, B\, \mathbf{then}\, C_1\, \mathbf{else}\, C_2$ par $\mathbf{select}[\overline{B}, \overline{\neg B}](C_1, C_2)$, qui est défini dans la suite (section 3.1).

Le lemme suivant exprime que l'environnement effectue toutes les transitions possibles à chaque fois que c'est son tour.

Lemme 1 (complétude de $[\![C]\!]$)**.** *Soit C une commande, et $t \in [\![C]\!]$. Supposons que t soit de la forme*

$$\cdots \sigma_{2i} \xrightarrow{m_i} \sigma_{2i+1} \xrightarrow{env} \sigma_{2i+2} \cdots$$

Alors, pour tout état $\overline{\sigma_{2i+2}}$, il existe une trace $\overline{t} \in [\![C]\!]$ qui est égale à t jusqu'à σ_{2i+1} et dont la transition suivante est $\sigma_{2i+1} \xrightarrow{env} \overline{\sigma_{2i+2}}$.

Démonstration. Par induction sur la définition de $[\![C]\!]$. □

3 La logique de séparation concurrente, et une preuve de correction

Dans cette section, nous introduisons la syntaxe et la sémantique de la logique de séparation concurrente, puis nous prouvons qu'elle est correcte en reformulant la preuve de correction de la logique de séparation concurrente de Vafeiadis [14], en exprimant la correction de la logique par l'existence d'une stratégie gagnante dans un jeu.

3.1 Les formules de le logique de séparation concurrente

Les formules de la *logique de séparation concurrente* sont les suivantes :

$$\begin{aligned} P, Q, R, J ::= {} & \mathbf{true} \mid \mathbf{false} \mid P \vee Q \mid P \wedge Q \mid \neg P \mid P \Rightarrow Q \mid \forall X.P \mid \mathrm{Own}_p(x) \\ & \mid \exists X.P \mid \mathbf{emp} \mid E_1 \mapsto^p E_2 \mid P * Q \mid P \mathrel{-\!\!*} Q \end{aligned}$$

où x est une variable de programme, et X une variable logique. De plus, on note B les formules qui sont dans le fragment booléen (variables et connecteurs booléens). Les constructions $\mathrm{Own}_p(x)$ n'est pas standard. Elle sert à la gestion des variables.

La gestion des «variables comme ressources» Afin d'éviter d'utiliser des conditions de bord pour réguler l'utilisation des variables de programme, à la suite de Bornat *et al* [2, 11], on ajoute à nos formules les prédicats $\mathrm{Own}_p(x)$, où $p \in \mathbb{P}$ est une permission, et $x \in \mathrm{Var}$ une variable de programme. Cela nous permet de gérer les variables *à l'intérieur* de la logique, au lieu de d'ajouter des prémisses auxiliaires pour les variables.

Nous reprenons la définition de [11].

Définition 10 (Permission)**.** *Un monoïde de permission $\mathbb{P}$ est un monoïde commutatif partiel régulier tel qu'il existe un plus grand élément $\top$, qu'on appelle la* permission totale*, tel que $\forall x \in \mathbb{P}.\ \top \cdot x$ n'est pas défini.*

L'idée est qu'il faut avoir la permission totale pour avoir le droit d'écrire, alors que n'importe quelle permission permet de lire. Dans ce cas, on peut utiliser la notation $\mathrm{Own}\_(x) = \exists P.\, \mathrm{Own}_P(x)$. La correction vient du fait que si quelqu'un dispose de la permission totale, alors personne d'autre n'a de permission vers cette partie de la mémoire.

Sémantique de la logique de séparation concurrente On reprend la sémantique de Bornat *et al* [11]. On définit une relation $\sigma \vDash P$, qu'on lit «σ satisfait P», où σ est un état, et P un prédicat.

$$
\begin{aligned}
\sigma \vDash \mathrm{Own}_p(x) &\iff \exists v \in \mathrm{Val}, \sigma(x) = (v, p) \\
\sigma \vDash E_1 = E_2 &\iff [\![E_1]\!] = [\![E_2]\!] \wedge \mathrm{fv}(E_1 = E_2) \subseteq \mathrm{vdom}(h) \\
\sigma \vDash P \Rightarrow Q &\iff (\sigma \vDash P) \Rightarrow (\sigma \vDash Q) \\
\sigma \vDash P \wedge Q &\iff \sigma \vDash P \text{ et } \sigma \vDash Q \\
\sigma \vDash P * Q &\iff \exists \sigma_1 \sigma_2,\ \sigma = \sigma_1 * \sigma_2 \text{ et } \sigma_1 \vDash P \text{ et } \sigma_2 \vDash Q
\end{aligned}
$$

On remarque qu'on a alors $\sigma \vDash \mathrm{Own}_p(x) * \mathrm{Own}_{p'}(x) \Leftrightarrow \sigma \vDash \mathrm{Own}_{pp'}(x)$, pour tout σ.

3.2 Le système de preuve

La logique de séparation concurrente étant une extension de la logique de Hoare, la plupart de ses règles sont les mêmes. Une différence importante est que dans le cas de la logique de séparation concurrente, les triplets de Hoare sont adjoint d'un contexte Γ qui associe aux noms de ressources leurs invariants (des prédicats de la logique). L'idée est que chaque ressource est un morceau d'état qui satisfait cet invariant et est protégé par un verrou, afin d'éviter des accès concurrents. Un certain nombres d'entre elles sont énoncées ci-dessous. Les règle qui correspondent aux constructions séquentielles du

langage ressemblent à celles de la logique de Hoare, par exemple :

$$\frac{}{\Gamma \vdash \{\mathrm{Own}_\top(x) * X = E\}\, x \coloneqq E\, \{\mathrm{Own}_\top(x) * x = X\}}\ \textsc{Aff}$$

$$\frac{\Gamma \vdash \{P\}\, C\, \{Q\} \quad \Gamma \vdash \{Q\}\, C'\, \{R\}}{\Gamma \vdash \{P\}\, C; C'\, \{R\}}\ \textsc{Seq}$$

$$\frac{S_1 \preccurlyeq_S P \quad S_2 \preccurlyeq_S P \quad \Gamma \vdash \{P \wedge S_1\}\, C_1\, \{Q\} \quad \Gamma \vdash \{P \wedge S_2\}\, C_2\, \{Q\}}{\Gamma \vdash \{P\}\ \mathbf{select}[S_1, S_2](C_1, C_2)\, \{Q\}}\ \textsc{Conj}$$

$$\frac{\Gamma \text{ est précis} \quad \Gamma \vdash \{P_1\}\, C\, \{Q_1\} \quad \Gamma \vdash \{P_2\}\, C\, \{Q_2\}}{\Gamma \vdash \{P_1 \wedge P_2\}\, C\, \{Q_1 \wedge Q_2\}}\ \textsc{Conj}$$

La règle pour la conjonction est intéressante, car il faut que le contexte Γ soit précis ; ce qui signifie que toutes les formules de Γ le sont.

Définition 11 (Formule précise)**.** *Soit P une formule de la logique de séparation concurrente. Alors P est précise si, pour tout état σ, il existe au plus un sous-état σ' de σ tel que $\sigma' \vDash P$.*

Enfin, il y a de nouvelles règles qui correspondent le fragment concurrent du langage :

$$\frac{\Gamma, r : J \vdash \{P\}\, C\, \{Q\}}{\Gamma \vdash \{P * J\}\ \mathbf{resource}\, r\, \mathbf{in}\, C\, \{Q * J\}}\ \textsc{Res}$$

$$\frac{B \preccurlyeq_S P \quad \Gamma \vdash \{(P * J) \wedge B\}\, C\, \{Q * J\}}{\Gamma, r : J \vdash \{P\}\ \mathbf{when}\, B\, \mathbf{with}\, r\, \mathbf{do}\, C\, \{Q\}}\ \textsc{When}$$

$$\frac{\Gamma \vdash \{P_1\}\, C_1\, \{Q_1\} \quad \Gamma \vdash \{P_2\}\, C_2\, \{Q_2\}}{\Gamma \vdash \{P_1 * P_2\}\, C_1 \parallel C_2\, \{Q_1 * Q_2\}}\ \textsc{Par}$$

Dans la règle When, il faut que dès lors que $\sigma \vDash P$, P ait suffisamment de permissions pour pouvoir lire les variables libres de B.

Définition 12. *On dit que P* détermine spatialement *Q, et on note $Q \preccurlyeq_S P$ lorsque, pour tout état σ_C,*

$$\sigma_C \vDash P \quad \Longrightarrow \quad \forall \sigma_F,\ \sigma_C * \sigma_F \vDash Q \Leftrightarrow \sigma_C \vDash Q$$

Une condition suffisante pour que $B \preccurlyeq_S P$ Si E est un ensemble de variables, on écrit $E \sqsubseteq P$ si $\forall h, h \vDash P \Rightarrow E \subseteq \mathrm{vdom}(h)$. Si B est une formule booléenne et si $\mathrm{fv}(B) \sqsubseteq P$, alors P détermine spatialement $\overline{Q}$, avec $\overline{Q} = (\Pi_{x \in \mathrm{fv}(Q)}\, \mathrm{Own}\_(x)) \wedge Q$.

3.3 Jeux

Soit Γ un contexte, C une commande, un triplet de la logique de séparation concurrente $\Gamma \vdash \{P\}\, C\, \{Q\}$ et $t \in [\![C]\!]$.

Nous allons définir deux graphes.

Définition 13 (Graphe des états séparés)**.** *Ses* sommets *sont tous les couples*

$$(\sigma_C, \sigma_F) \in \text{Etat} \times \text{Etat}$$

tels que $\sigma_C * \sigma_F$ est bien défini, et ses arcs *sont de deux types :*

- *les arcs Eve, de la forme $(\sigma_C, \sigma_F) \xrightarrow{\sigma_U} (\sigma'_C, \sigma_U * \sigma_F)$,*
- *les arcs Adam, de la forme $(\sigma_C, \sigma_F) \xrightarrow{\sigma_L} (\sigma_C * \sigma_L, \sigma'_F)$.*

Définition 14 (Graphe des états)**.** *L'ensemble des sommets de ce graphe est l'ensemble* Etat *des états, et ses arcs sont de deux types :*

- *les arcs Code, de la forme $\sigma \xrightarrow{m} \sigma'$, où m est une instruction élémentaire, et $\sigma \overset{m}{\rightsquigarrow} \sigma'$,*
- *les arcs Environnement, de la forme $\sigma \xrightarrow{env} \sigma'$, pour tout σ and σ'.*

Une trace d'exécution (définition 4), est alors exactement un chemin alterné dans le graphe des états qui commence par un coup Environnement.

Les étiquettes σ_L et σ_U sont les morceaux d'états qui sont échangés à travers les verrous, ils doivent donc satisfaire les invariants associés à ces verrous. Pour exprimer cela, on introduit un nouveau graphe, où les sommets contiennent les prédicats que (σ_C, σ_F) doivent satisfaire, et les arcs sont étiquetés par les prédicats que σ_L et σ_U doivent satisfaire.

Définition 15 (Graphe des prédicats séparés)**.** *Les sommets du* graphe des prédicats séparés *sont les paires $\{P, Q\}$ de prédicats, et ses arcs sont de deux types :*

- *les transitions Joueur, étiquetées par un prédicat,*
- *les transitions Opposant, étiquetées par un prédicat.*

On étend la relation $\vDash$ aux états et aux prédicats séparés de manière naturelle :

$$(\sigma_C, \sigma_F) \vDash \{P, Q\} \overset{def}{\iff} \sigma_C \vDash P \text{ et } \sigma_F \vDash Q.$$

On appelle *jeu de séparation* un chemin alternant (fini ou infini) du graphe des prédicats séparés commençant par Opposant.

La donnée de $\Gamma \vdash \{P\}\, C\, \{Q\}$, et $t \in [\![C]\!]$ définit un jeu de séparation, dont les étiquettes sont données en remplaçant chaque instruction m par $\Gamma(\text{lock}^-(m))$ et chaque étiquette *env* par $\Gamma(\text{lock}^+(m))$, où m est l'instruction qui vient juste après cette étiquette ; et dont tous les sommets sont $\{\mathbf{true}, \mathbf{true}\}$, sauf le premier, qui est $\{P, \mathbf{true}\}$, et le dernier (s'il existe), qui est $\{Q, \mathbf{true}\}$. Note : $\text{lock}^+(m)$ et $\text{lock}^-(m)$ sont toujours des singletons

ou l'ensemble vide. On convient que $\Gamma(\{r\}) = \Gamma(r)$ et $\Gamma(\emptyset) = \mathbf{emp}$ pour tout contexte Γ.

Le jeu associé à un triplet et à une trace donne les règles du jeu que doivent suivre Adam et Eve. En effet, un jeu de séparation définit un ensemble de trajectoires alternantes dans le graphe des états séparés, c'est l'ensemble des chemins du graphe des états séparés qui réalisent le jeu de séparation induit par le triplet et la trace.

Définition 16 (Réalisation)**.** *Un arc* $(\sigma_C, \sigma_F) \xrightarrow{\sigma} (\sigma'_C, \sigma'_F)$ *du graphe des états séparés* réalise *un arc* $\{P, Q\} \xrightarrow{I} \{P', Q'\}$ *du graphe des prédicats séparés lorsque :*

- *le premier est un arc Adam (respectivement Eve), et le second est un arc Opposant (respectivement Joueur),*
- σ *satisfait l'invariant* I*,*
- *chaque état séparé satisfait le prédicat séparé qui lui correspond (ie.* $(\sigma_C, \sigma_F) \vDash \{P, Q\}$*, et* $(\sigma'_C, \sigma'_F) \vDash \{P', Q'\}$*).*

On étend la relation de réalisation aux paires de chemins alternés par recollement.

On peut alors définir les notions de stratégie et de stratégie gagnante.

Définition 17 (Stratégie)**.** Une stratégie pour un jeu de séparation *est la donnée d'un ensemble de chemins alternants dans le graphe des états séparés, qui réalisent chacun un préfixe fini de longueur paire du jeu.*

Une stratégie est dite gagnante *lorsqu'elle a réponse à tout coup d'Adam.*

Un triplet $\Gamma \vdash \{P\}\, C\, \{Q\}$ induit un ensemble de jeux de séparations : l'union de tous les jeux de séparation induits par le triplet et t, pour tout $t \in [\![C]\!]$.

Théorème 1 (Correction)**.** *Soit un triplet de la logique de séparation concurrente* $\Gamma \vdash \{P\}\, C\, \{Q\}$*, et* π *une preuve de ce triplet.*

Alors π *définit un stratégie gagnante pour chacun des jeux induit par* $\Gamma \vdash \{P\}\, C\, \{Q\}$*.*

On prouve ce théorème en montrant séparément que chaque règle d'inférence est correcte, dans la suite de cette section.

Pour obtenir l'équivalent du résultat classique de la correction partielle des logiques de Hoare, il faut se restreindre au cas où il n'y a aucune interférence de l'environnement.

Corollaire 1. *Lorsque* Γ *est vide, toute trace d'exécution de* C *qui termine lancée dans un état séparé vérifiant* $\{P, \textbf{\textit{false}}\}$ *termine dans un état séparé vérifiant* $\{Q, \textbf{\textit{true}}\}$*, pour autant que l'environnement reste passif.*

Démonstration. Soit une trace $t \in [\![C]\!]$ où l'environnement reste passif.

Comme l'environnement est passif, Adam a une contre-stratégie gagnante, qui consiste à rejouer la paire d'états précédente. Donc par le théorème 2, Eve a toujours une réponse. Par suite, le jeu de séparation induite par t est réalisé tout entier par un chemin dans le graphe des états séparés, et donc l'état final satisfait bien $\{Q, \mathbf{true}\}$. □

3.4 Cas de la séquence (;)

Si on dispose de stratégies gagnantes pour $\Gamma \vdash \{P\}\, C\, \{Q\}$ et pour $\Gamma \vdash \{Q\}\, C'\, \{R\}$, la stratégie qui consiste à jouer comme la première au début, puis à jouer comme la seconde ensuite, est une stratégie gagnante.

Lemme 2. *Soient C et C' deux commandes, P, Q, R des prédicats, et $\tilde{t} \in [\![C; C']\!]$.*

Supposons de plus que l'on dispose de stratégies gagnantes pour tous les jeux induits par $\Gamma \vdash \{P\}\, C\, \{Q\}$ et par $\Gamma \vdash \{Q\}\, C'\, \{R\}$.

Alors il existe une stratégie gagnante pour le jeu induit par $\Gamma \vdash \{P\}\, C; C'\, \{R\}$ et $\tilde{t}$.

Démonstration. Par définition de $[\![C; C']\!]$, il existe $t \in [\![C]\!]$ et $t' \in [\![C']\!]$ tels que $\tilde{t} = t \cdot t'$. Par hypothèse, on dispose de stratégies gagnantes $\texttt{strat}$ pour le jeu G, induit par $\Gamma \vdash \{P\}\, C\, \{Q\}$ et t, et $\texttt{strat}'$ pour G', celui induit par $\Gamma \vdash \{Q\}\, C'\, \{R\}$ et t'.

On pose

$$\texttt{strat}'' = \texttt{strat} \cup \{p \cdot p' \mid p \in \texttt{strat}, p' \in \texttt{strat}' \text{ et } p \text{ réalise tout le jeu associé à } t\},$$

où le chemin $p \cdot p'$ n'est défini que si p est fini, et si son point d'arrivée est égal au point de départ de p' (comme dans la définition 6). Nous prétendons que $\texttt{strat}''$ est une stratégie gagnante du jeu associé à $\tilde{t}$.

En effet, les préfixes pairs du jeu induit par la concaténation $G \cdot G'$ de G et G' sont :

$$G \cup \{G \cdot g' \mid g' \text{ est un préfixe de taille paire pair de } G'\},$$

donc $\texttt{strat}''$ est une stratégique gagnante de ce jeu, et ce jeu est strictement plus difficile (moins de contrainte pour Adam, et plus de contraintes pour Eve) que le jeu associé à $\tilde{t}$, qui est égal à $G \cdot G'$ sauf au point de jonction de G et G', où on remplace $\{Q, \mathbf{true}\}$ par $\{\mathbf{true}, \mathbf{true}\}$. □

3.5 Cas du produit parallèle

Sans surprise, la règle pour le produit parallèle est celle dont la preuve est la plus compliquée. L'idée de la preuve est que si l'ordonnancement τ fait jouer la trace t au i-ème coup, alors la stratégie pour le produit parallèle joue comme la stratégie pour t, en laissant la partie de σ_C qui correspond à la trace t' intacte.

Lemme 3 (Produit parallèle)**.** *Soient C_1 et C_2 deux commandes, et $\tilde{t} \in [\![C_1 \parallel C_2]\!]$.*

Supposons que l'on dispose de stratégies gagnantes pour tous les jeux induits par $\Gamma \vdash \{P_1\}\, C_1\, \{Q_1\}$ et par $\Gamma \vdash \{P_2\}\, C_2\, \{Q_2\}$.

Alors il existe une stratégie gagnante pour le jeu induit par $\Gamma \vdash \{P_1 * P_2\}\, C_1 \parallel C_2\, \{Q_1 * Q_2\}$.

Démonstration. Par définition de $[\![C_1 \parallel C_2]\!]$, on dispose de $t_1 \in [\![C_1]\!]$, $t_2 \in [\![C_2]\!]$ et d'un ordonnancement τ tels que $\tau(t, t') = \tilde{t}$.

On introduit au préalable le graphe des états triplement séparés, qui est analogue au graphe des états séparés. Il permet d'expliciter le fait que la partie de l'état qui appartient au code peut être séparé un deux parties, un qui correspond à C_1 et l'autre à C_2.

Définition 18 (Graphe des états triplement séparés)**.** *Ses états sont les triplets $(\sigma_1, \sigma_2, s_F)$, tels que le produit des trois composante est bien défini. Il y a quatre sortes d'arcs :*

- *Les arcs Eve 1, de la forme $(\sigma_1, \sigma_2, \sigma_F) \xrightarrow[1]{\sigma_U} (\sigma_1', \sigma_2, \sigma_F * \sigma_U)$,*
- *Les arcs Eve 2, de la forme $(\sigma_1, \sigma_2, \sigma_F) \xrightarrow[2]{\sigma_U} (\sigma_1, \sigma_2', \sigma_F * \sigma_U)$,*
- *Les arcs Adam 1, de la forme $(\sigma_1, \sigma_2, \sigma_F) \xrightarrow[1]{\sigma_L} (\sigma_1 * \sigma_L, \sigma_2, \sigma_F')$,*
- *Les arcs Adam 2, de la forme $(\sigma_1, \sigma_2, \sigma_F) \xrightarrow[2]{\sigma_L} (\sigma_1, \sigma_2 * \sigma_L, \sigma_F')$.*

Dans ce graphe, nous ne considérerons que les chemins tels qu'un coup d'Adam j est suivi par un coup de Eve j, tel qu'un coup Eve est suivi par un coup Adam, et qui commencent par un coup Adam.

Nous allons définir deux opérations utiles sur les chemins de ce graphe : la projection et l'épissage (*splicing* en anglais). On peut projeter un chemin de ce graphe sur un chemin du graphe des états séparés, en oubliant le numéro du joueur (mais pas s'il est Eve ou Adam), en en transformant les états par $(\sigma_1, \sigma_2, \sigma_2) \mapsto (\sigma_1 * s_2, \sigma_F)$. Remarquons que cette fonction envoie des chemins (au sens du paragraphe précédent) sur des chemins alternés pairs qui commencent par un coup d'Adam. Grâce à cette projection, on peut étendre à ce graphe et à ses chemins les notions de réalisation et de stratégies.

Étant donné un chemin du graphe des états triplement séparés, on peut le découper en deux chemins du graphe des états séparés. La différence avec l'épissage (de l'ARN par exemple) est qu'ici les deux chemins qui en résultent sont utiles, on les appelle donc les exons.

Définition 19 (Epissage)**.** *Soit p un chemin dans le graphe des états triplement séparés. Découpons le en suites maximales telles que les coups ont le même numéro (1 ou 2), notons cette décomposition $p = r_1 r_2 \cdots r_n$. Ainsi, les r_i sont tous de tailles paires.*

Définissions l'épissage par induction sur n : si $n = 0$, p est vide, et ses deux exons sont vides aussi. Soit $n \geq 0$ et $p = r_1 r_2 \cdots r_n r_{n+1}$. Notons (s_1, s_2)

les deux exons de $r_1 r_2 \cdots r_n$. Supposons, par exemple, que le numéro de r_n est 1, et donc que celui de r_{n+1} est 2. Alors l'épissage de p est défini comme suit.

- *Projetons r_{n+1} à droite dans le graphe des états séparés, en oubliant le numéro, et en transformant $(\sigma_1, \sigma_2, \sigma_F) \mapsto (\sigma_1, \sigma_2 * \sigma_F)$, notons r ce projeté. Notons $(\sigma_1, \sigma_F) \xrightarrow{\sigma_L} (\sigma_1, \sigma'_F)$ la première transition de r. Alors on ajoute à s_2 le chemin r, où on a remplacé au début de r, (σ_1, σ_F) par la dernière position de s_2. Le nouveau s_2 est bien un chemin alterné et de taille paire du graphe des états séparés.*
- *On ne touche pas à s_1.*

Nous allons construire par induction sur n une stratégie gagnante (de chemins du graphe des états triplement séparés) pour le préfixe de taille $2n$ de $\tilde{G}$, le jeu associé à $\tilde{t}$, telle que les couples d'exons de chaque chemin de cette stratégie sont dans $\texttt{strat}_1 \times \texttt{strat}_2$.

Initialisation Dans le cas du jeu vide, le chemin qui le réalise est réduit à un état. Par hypothèse, l'état de départ (σ_C, σ_F) vérifie $P_1 * P_2$, donc on peut décomposer σ_C en $\sigma_1 * \sigma_2$ tels que $\sigma_i \vDash P_i$, pour $i = 1, 2$. On relève donc ce chemin en $(\sigma_1, \sigma_2, \sigma_F)$.

Hérédité Soit g un préfixe de taille paire du jeu induit par $\tilde{t}$. Par hypothèse d'induction, on dispose d'un ensemble `strat` de chemins du graphe des états triplement séparés qui réalisent les préfixes pairs de g et qui forment une stratégie gagnante pour g. De plus, chacun de ces chemins peut être épissé en deux chemins, l'un réalisant un préfixe de G_1, et l'autre réalisant un préfixe de G_2.

Soit p un de ces chemins, s_1 et s_2 ses deux exons. Soit $j = 0$ ou 1 tel que la prochaine instruction du code vient de t_j. Notons le dernier état de p : $(\sigma_1, \sigma_2, \sigma_F)$. On considère un coup d'Adam en réponse à la projection de p dans le graphe des états (doublement) séparés, notons le $(\sigma_1 * \sigma_2, \sigma_F) \xrightarrow{\sigma_L} (\sigma_1 * \sigma_2 * \sigma_L, \sigma'_F)$.

On relève le coup d'Adam dans notre graphe des états triplement séparés en $(\sigma_1, \sigma_2, \sigma_F) \xrightarrow[1]{\sigma_L} (\sigma_1 * \sigma_L, \sigma_2, \sigma'_F)$, si $j = 1$, et en $(\sigma_1, \sigma_2, \sigma_F) \xrightarrow[1]{\sigma_L} (\sigma_1, \sigma_2 * \sigma_L, \sigma'_F)$ si $j = 2$.

Supposons, par exemple, que dans $\tilde{t}$, c'est au joueur $j = 1$ de s'exécuter. On s'intéresse donc à ce que joue $\texttt{strat}_1$ après $s_1 \xrightarrow{\sigma_L} (\sigma_1, \sigma_2 * \sigma'_F)$. Cela a du sens, car par définition de l'épissage, la première composante du dernier coup de s_1 est bien σ_1. Notons cette réponse $(\sigma_1 * \sigma_L, \sigma_2 * \sigma_F) \xrightarrow{\sigma_U} (\sigma'_1, \sigma_2 * \sigma_F * \sigma_U)$. On la relève dans notre nouveau graphe en $(\sigma_1 * \sigma_L, \sigma_2, \sigma_F) \xrightarrow[1]{\sigma_U} (\sigma'_1, \sigma_2, \sigma_F * \sigma_U)$.

Supposons que le chemin p qu'on vient de construire est aussi long que le jeu induit par $\tilde{t}$. Le chemin qu'on vient d'étendre est un chemin du graphe

des états triplement séparés, et il s'épisse bien un couple (s_1, s_2) de chemins de $\texttt{strat}_1 \times \texttt{strat}_2$. Comme $|p| = |s_1| + |s_2|$, s_1 et s_2 réalisent tout G_1 et G_2 respectivement. Par suite, si on note leurs dernier sommets respectifs $(\sigma_{1,)}$ et $(\sigma_{2,)}$, le dernier sommet du chemin du graphe des états triplement séparés qu'on vient de construire est de la forme $(\sigma_1, \sigma_2, \sigma_F)$, donc $(\sigma_1 * \sigma_2, \sigma_F) \vDash \{Q_1 * Q_2, \mathbf{true}\}$. □

3.6 Cas de la conjonction

Dans le cas de la conjonction, on dispose de deux stratégies portant sur les mêmes traces, une pour chaque prémisse de la règle Conj : $\Gamma \vdash \{P_1\}\, C\, \{Q_2\}$, et $\Gamma \vdash \{P_2\}\, C\, \{Q_2\}$. On va montrer que, comme, par hypothèse, Γ ne contient que des formules précises, il n'y a, à chaque position, qu'un seul coup possible et donc les deux stratégies sont nécessairement égales. Cette stratégie commune est gagnante pour les jeux induits par la conjonction.

Lemme 4. *Soient C une commande, Γ un environnement de formules précises, P_1, P_2, Q_1, Q_2 des formules de la logique de séparation et $t \in [\![C]\!]$.*

Supposons qu'il existe des stratégies gagnantes pour tous les jeux induits par $\Gamma \vdash \{P_1\}\, C\, \{Q_1\}$ et par $\Gamma \vdash \{P_2\}\, C\, \{Q_2\}$.

Alors il existe une stratégie gagnante pour le jeu induit par $\Gamma \vdash \{P_1 \wedge P_2\}\, C\, \{Q_1 \wedge Q_2\}$ et t.

Démonstration. Par hypothèse, on dispose de deux stratégies, $\texttt{strat}_1$ et $\texttt{strat}_2$, pour les jeux induits par $\Gamma \vdash \{P_1\}\, C\, \{Q_1\}$ et t et par $\Gamma \vdash \{P_2\}\, C\, \{Q_2\}$ et t respectivement.

Comme les prédicats de Γ sont précis, Adam et Eve n'ont qu'une seule manière de jouer. En effet à chaque étape, il n'y a qu'un seul choix de σ_U ou de σ_L d'après la précision ; et comme le monoïde des permissions est régulier, les joueurs peuvent jouer au plus un sommet à chaque coup. Par suite les deux stratégies $\texttt{strat}_1$ et $\texttt{strat}_2$ sont égales, et il est immédiat de voir que cette stratégie unique est aussi gagnante pour les jeux induits par $\Gamma \vdash \{P_1 \wedge P_2\}\, C\, \{Q_1 \wedge Q_2\}$. □

3.7 Cas de resource

Si on dispose d'une stratégie gagnante pour le jeu induit par $\Gamma, r : J \vdash \{P\}\, C\, \{Q\}$ et une trace $t \in [\![C]\!]$, et qu'on interdit à Adam de toucher la ressource r, alors soit r est prise par le code, soit il existe un morceau d'état qui vérifie J que personne ne touche. La stratégie qui explicite cela est une stratégie gagnante pour le jeu $\Gamma \vdash \{P * J\}\, \mathbf{resource}\, r \in C\, \{Q * J\}$ et $\text{cache}_r(t)$ (nous rappelons que $\text{cache}_r(t)$, définie à la fin de la section 2.3, cache la ressource r en remplaçant, dans t, $P(r)$ et $V(r)$ par **nop**).

Lemme 5. *Soient* Γ *un environnement,* r *un nom de ressource,* J, P, Q *des prédicats de la logique de séparation,* C *une commande, notons* $C' = \textbf{\textit{resource}}\, r\, \textbf{\textit{in}}\, C$ *et* $t' \in [\![C']\!]$.

Supposons que l'on dispose d'une stratégie gagnante pour tous les jeux induits par $\Gamma, r : J \vdash \{P\}\, C\, \{Q\}$.

Alors il existe une stratégie gagnante pour le jeu G' *induit par* $\Gamma \vdash \{P * J\}\, C'\, \{Q * J\}$ *et* t'.

Démonstration. Par définition de $[\![\cdot]\!]$, il existe $t \in [\![C]\!]$ tel que $t' = \text{cache}_r(t)$. Soit `strat` une stratégie gagnante du jeu G induit par $\Gamma, r : J \vdash \{P\}\, C\, \{Q\}$ et t.

Comme dans la preuve du lemme pour la règle Par, on définit un nouveau graphe des états séparés où il y a deux types de sommets : les sommet de la forme (σ_C, σ_F), et ceux de la forme $(\sigma_C, \sigma_J, \sigma_F)$, tels que les deux produits des composantes sont définies, et tels que $\sigma_J \vDash J$. Les premiers représentent les états où la ressource r est prise par le code, et les seconds représentent ceux où elle n'est pas prise. Ces sommets sont des triplets pour refléter que l'environnement ne peut plus y toucher.

Les arcs sont les suivants :

— les arcs Eve r, de la forme $(\sigma_C, \sigma_F) \xrightarrow{\sigma_U} (\sigma'_C, \sigma_U * \sigma_F)$, lorsque r est prise par le code ;
— les arcs Adam r, de la forme $(\sigma_C, \sigma_F) \xrightarrow{\sigma_L} (\sigma_C * \sigma_L, \sigma'_F)$, lorsque r est prise par le code ;
— les arcs Eve $\not r$, de la forme $(\sigma_C, \sigma_J, \sigma_F) \xrightarrow{\sigma_U} (\sigma'_C, \sigma_J, \sigma_U * \sigma_F)$, lorsque r n'est pas prise par le code ;
— les arcs Adam $\not r$, de la forme $(\sigma_C, \sigma_F) \xrightarrow{\sigma_L} (\sigma_C * \sigma_L, \sigma_J, \sigma'_F)$, lorsque r n'est pas prise par le code ;
— les arcs Eve $V(r)$, de la forme $(\sigma_C, \sigma_F) \xrightarrow{\textbf{emp}} (\sigma'_C, \sigma_J, \sigma_F)$, lorsque r est en train d'être rendue par le code ;
— les arcs Adam $P(r)$, de la forme $(\sigma_C, \sigma_J, \sigma_F) \xrightarrow{\textbf{emp}} (\sigma_C * \sigma_J, \sigma'_F)$, lorsque r est en train d'être prise par le code.

On définit la projection des chemins pairs alternés (Adam - Eve) en envoyant $(\sigma_C, \sigma_J, \sigma_F)$ vers $(\sigma_C * \sigma_J, \sigma_F)$ et en conservant les sommets qui sont des couples et les étiquettes des arcs. Grâce à cette projection, on peut étendre les notions de réalisation et de stratégie à des ensembles de tels chemins.

On va construire une relation $u' \sim u$, où u' est un chemin du graphe défini ci-dessus qui réalise un préfixe de G' et u est un chemin de `strat` (donc un chemin du graphe des états séparés de la définition 13), tels que u' et u soient de même taille. Cette dernière propriété nous permet de faire correspondre le $i$ème sommet de u' avec le $i$ème sommet de u. De plus, si $u' \sim u$, alors pour chaque sommet n' de u', en notant n l'état de u qui correspond à n', si la ressource r est prise dans n, alors n' est de la forme $(\sigma_C, \sigma_J, \sigma_F)$ et $n = (\sigma_C, \sigma_J * \sigma_F)$ et sinon n' est un couple et $n = n'$. Cette relation nous

permettra de définir une stratégie gagnante pour G'. Nous allons la définir par induction sur le nombre de coups d'Eve dans u.

Pour le chemin composé d'un unique sommet : par hypothèse, ce sommet est de la forme $(\sigma_C * \sigma_J, \sigma_F)$, avec $\sigma_C \vDash P$, $\sigma_J \vDash J$, et la ressource r n'y est pas prise. On pose $(\sigma_C, \sigma_J, \sigma_F) \sim (\sigma_C * \sigma_J, \sigma_F)$.

Supposons qu'on dispose de $u' \sim u$. u réalise donc un préfixe g de G. On considère les deux transitions de t qui suivent le préfixe de t qui correspond à g. Notons les $\sigma \xrightarrow{env} \sigma' \xrightarrow{m} \sigma''$. Il y a plusieurs cas :

— Si $m = P(r)$, alors par bon parenthésage des verrous, r n'est pas pris à cet endroit (ce serait un *deadlock*), donc le dernier sommet de u' est de la forme $(\sigma_C, \sigma_J, \sigma_F)$. Notons $(\sigma_C * \sigma_J, \sigma_F) \xrightarrow{\mathbf{emp}} (\sigma_C * \sigma_J, \sigma'_F)$ le coup d'Adam face à la projection de u (projection qu'on a définie ci-dessus). Notons $\overline{u'}$ la concaténation de u' et de $(\sigma_C, \sigma_J, \sigma_F) \xrightarrow{emp} (\sigma_C * \sigma_J, \sigma'_F) \xrightarrow{\mathbf{emp}} (\sigma_C * \sigma_J, \sigma'_F)$ et notons $\overline{u}$ la concaténation de u et de $(\sigma_C, \sigma_F * \sigma_J) \xrightarrow{\sigma_J} (\sigma_C * \sigma_J, \sigma_F)$. Ces deux concaténations sont bien définies. Alors on peut poser $\overline{u'} \sim \overline{u}$.

— Si $m = V(r)$, la stratégie de G nous donne un morceau d'état σ_J, dont on se sert pour étendre $\sim$, comme dans le cas précédent.

— Si m est une autre instruction et que r n'est pas prise à la fin de u, alors, en réponse à la projection de u', Adam joue un coup de la forme $(\sigma_C * \sigma_J, \sigma_F) \xrightarrow{\sigma_L} (\sigma_C * \sigma_L * \sigma_J, \sigma'_F)$. On étend la relation $\sim$ en utilisant la réponse de `strat` à u suivi de $(\sigma_C, \sigma_J * \sigma_F) \xrightarrow{\sigma_L} (\sigma_C * \sigma_L, \sigma_J * \sigma_F)$.

— Dernier cas, si m n'est ni $P(r)$ ni $V(r)$ et que r est prise à la fin de u. Alors on joue exactement comme `strat`, quand elle répond à u suivi du coup d'Adam dans le jeu G'.

Si le jeu G est fini et qu'il est réalisé tout entier par u, et que $u' \sim u$, d'après le bon parenthésage de t, r n'est pas prise à la fin de t, donc le dernier sommet de u' est un triplet $(\sigma_C, \sigma_J, \sigma_F)$, sa projection est $(\sigma_C * \sigma_J, \sigma_F)$, et le dernier sommet de u, qui vérifie $\{Q, \mathbf{true}\}$, est égal à $(\sigma_C, \sigma_J * \sigma_F)$, donc le dernier sommet de u' vérifie bien $\{Q * J, \mathbf{true}\}$.

La définition nous fournit aussi une manière de construire `strat'`, qui est la projection des éléments de l'ensemble $\{u' \mid u' \sim u, u \in \texttt{strat}\}$, en effet, on voit bien que cette procédure donne un réponse à toutes les coups d'Adam dans le jeu G'. □

3.8 Cas de **when** B **with** r **do** C

Par définition de $[\![\cdot]\!]$, si cette commande s'exécute, c'est que l'état courant vérifie B, et le verrou r est libre, donc c'est un point de départ valide des jeux induits par $\Gamma \vdash \{(P * J) \wedge B\}\, C\, \{Q\}$, qui sont précisément les jeux pour lesquels on dispose de stratégies gagnantes.

Lemme 6. *Soient Γ un environnement, r un nom de ressource, J, P, Q des prédicats de la logique de séparation, C une commande et B une formule booléenne telle que $B \preccurlyeq_S P$. Notons $C' = \textbf{\textit{when}}\, B\, \textbf{\textit{with}}\, r\, \textbf{\textit{do}}\, C$.*

Soit $t' \in [\![C']\!]$, on a donc $t' = t_1 \cdot t \cdot t_2$, avec $t \in [\![C]\!]$, et $t_1 \in \textbf{\textit{succès}}_{P(r)}(\sigma_2 = \sigma_3 \wedge \sigma_2 \vDash B)$ et $t_2 \in \textbf{\textit{succès}}_{V(r)}(\sigma_2 = \sigma_3)$.

Supposons de plus qu'on dispose d'une stratégie gagnante `strat` *pour le jeu G induit par $\Gamma \vdash \{(P * J) \wedge B\}\, C\, \{Q * J\}$ et par t.*

Alors il existe une stratégie gagnante pour le jeu G' induit par $\Gamma, r : J \vdash \{P\}\, C'\, \{Q\}$ et t'.

Démonstration. Les réalisateurs du préfixe vide de G' sont les états séparés de la forme (σ_C, σ_F), avec $\sigma_C \vDash P$. La première instruction du code étant $P(r)$, le premier coup d'Adam doit être de la forme $(\sigma_C, \sigma_F) \xrightarrow{\sigma_J} (\sigma_C * \sigma_J, \sigma'_F)$, avec $\sigma_J \vDash J$, et Eve répond en rejouant le même sommet. De plus, par définition de $[\![\cdot]\!]$, on a $\sigma_C * \sigma_F \vDash B$, et donc $\sigma_C * \sigma_J \vDash B$ (car $B \preccurlyeq_S P$ et $\sigma_C \vDash P$).

Comme la trace est de la forme $t' = t_1 \cdot t \cdot t_2$, et qu'à la fin de t_1 l'état vérifie $\{(P * J) \wedge B, \mathbf{true}\}$, la stratégie qui consiste à jouer comme au paragraphe précédent, puis comme `strat`, est une stratégie gagnante pour tous les préfixes de G', sauf G' tout entier. A la fin, comme le dernier noeud de G est $\{Q * J, \mathbf{true}\}$, le dernier état vérifie $Q * J$, et donc Eve peut réponde au dernier coup d'Adam en rendant à l'environnement σ_J, avec $\sigma_J \vDash J$, car la dernière instruction de t' est $V(r)$. □

3.9 Cas de la règle de Frame

Si on dispose de stratégies gagnantes pour les jeux induits par $\Gamma \vdash \{P\}\, C\, \{Q\}$, et qu'on part d'un état satisfaisant $P * R$, tout au long de l'exécution de C, le morceau d'état σ_R qui correspond à R est dans la première composante des états séparés, donc Adam ne peut pas le toucher, et Eve peut jouer comme si σ_R était dans la seconde composante, car on peut simuler la réponse de la stratégie pour $\Gamma \vdash \{P\}\, C\, \{Q\}$ où σ_R est dans la seconde composante et où Adam ne touche jamais σ_R.

Lemme 7. *Soient Γ un environnement, C une commande, P, Q, R des prédicats de la logique de séparation, et $t \in [\![C]\!]$. Supposons que l'on dispose de* `strat`*, une stratégie gagnante pour G, le jeu induit par $\Gamma \vdash \{P\}\, C\, \{Q\}$ et t.*

Alors il existe une stratégie gagnante `strat'` *pour le jeu induit par $\Gamma \vdash \{P * R\}\, C\, \{Q * R\}$ et t.*

Démonstration. On va construire `strat'` par induction sur les préfixes pairs de G', telle que, dans chaque chemin qui réalise un de ces préfixes, tous les états sont de la forme $(\sigma_C * \sigma_R, \sigma_F)$, avec $\sigma_R \vDash R$ qui reste le même au cours de tout le chemin, et tel que, de plus, si on applique la transformation suivante sur les états sans changer les étiquettes aux chemins de `strat'`, ils sont dans `strat`. Cette transformation, qu'on note T, est $(\sigma_C * \sigma_R, \sigma_F) \mapsto (\sigma_C, \sigma_R * \sigma_F)$.

Si le préfixe est vide, on dispose d'un état séparé qui réalise $\{P*J, \mathbf{true}\}$. Il est donc de la forme $(\sigma_C * \sigma_R, \sigma_F)$, avec $\sigma_C \vDash P$ et $\sigma_R \vDash R$.

Si on dispose d'un chemin p qui réalise un préfixe de G (qui n'est pas G tout entier), par hypothèse le dernier sommet de p est de la forme $(\sigma_C * \sigma_R, \sigma_F)$, donc le prochain coup c d'Adam est de la forme $(\sigma_C * \sigma_R, \sigma_F) \xrightarrow{\sigma_L} (\sigma_C * \sigma_L * \sigma_R, \sigma'_F)$. Notons $(\sigma_C * \sigma_L, \sigma'_F * \sigma_R) \xrightarrow{\sigma_U} (\sigma'_C, \sigma'_F * \sigma_U * \sigma_R)$ la réponse de `strat` à l'image par T de $s \cdot c$. Alors `strat`$'$ utilise l'image inverse par T de la réponse de `strat`. □

3.10 Cas du select

Au premier coup de la stratégie pour **select**, par hypothèse, on sait que $\sigma_C \vDash S_1$ (ou bien, de manière symétrique, $\sigma_C \vDash S_2$), et donc on peut laisser la main a une stratégie gagnante pour $\Gamma \vdash \{P \wedge S_1\}\, C\, \{Q\}$.

Lemme 8. *Soient Γ un environnement, S_1, S_2, P, Q des prédicats de la logique de séparation tels que $S_1 \preccurlyeq_S P$ et $S_2 \preccurlyeq_S P$, et C_1, C_2 des commandes. Notons $C' = \textbf{\textit{select}}[S_1, S_2](C_1, C_2)$ et $t' \in [\![C']\!]$.*

Supposons qu'on dispose de stratégie gagnantes, pour tous les jeux induits par $\Gamma \vdash \{P * S_1\}\, C_1\, \{Q\}$ et $\Gamma \vdash \{P * S_2\}\, C_2\, \{Q\}$.

Alors il existe une stratégie gagnante pour le jeu induit par $\Gamma \vdash \{P\}\, C'\, \{Q\}$ et t'.

Démonstration. Par définition de $[\![\cdot]\!]$, il existe $t_1 \in [\![C_1]\!]$, (ou, de manière symétrique, $t_2 \in [\![C_2]\!]$) telle que :

$$t' = \sigma_{-1} \xrightarrow{env} \sigma_0 \xrightarrow{\mathbf{nop}} \sigma_1 \xrightarrow{env} \sigma_2 \cdot t_1$$

avec $\sigma_0 = \sigma_1$ et $\sigma_1 \vDash B$.

Soit $\texttt{strat}_1$ une stratégie gagnante pour le jeu induit par $\Gamma \vdash \{P \wedge S_1\}\, C_1\, \{Q\}$ et t_1. La stratégie qui consiste à se comporter comme $\texttt{strat}_1$ après avoir répondu par l'identité au premier coup d'Adam est une stratégie gagnante. □

4 Concurrence

L'objectif de cette dernière partie du rapport est d'établir un résultat plus fin sur les programmes bien typés par la logique de séparation concurrente : leur seule source de non-déterminisme est l'ordre dans lequel sont ordonnancés les sections critiques. Un corollaire important sera que l'exécution d'un programme bien typé ne contient pas d'accès concurrents. Notre approche fondée sur la théorie des jeux nous permet d'énoncer précisément cette propriété dans le cadre des traces concurrentes, dans la tradition de Mazurkiewicz.

Nous allons montrer que si deux instructions s'exécutent en parallèle, alors elles commutent entre elles lorsque l'environnement ne les en empêchent pas. En effet, si l'environnement touche certaines ressources en prenant un verrou, il peut empêcher des instructions en apparence concurrentes de commuter. C'est pourquoi nous avons besoin de suivre les ressources à la trace, de manière interactive, et avec plus de précision qu'en section 3. Dans ce nouveau cadre, Adam devra donc déclarer quelles ressources l'environnement a touchées lors de la transition qu'il cherche à justifier (ou à relever) dans la logique. De surcroit, les sommets du graphe des états séparés contiendront une nouvelle information : les états associés aux ressources qui ne sont pas verrouillées — on demandera que ces états vérifient les invariants imposés par le contexte Γ.

4.1 Traces concurrentes

Pour avoir une définition rigoureuse de l'exécution parallèle de deux instructions, nous allons utiliser des traces concurrentes, qu'on peut voir comme des traces munies d'une relation d'ordre.

Définition 20 (Traces concurrente)**.** *Une* trace concurrente *est la donnée d'une une trace t, au sens de la définition 4, et $\leq_t$ un ordre (partiel) sur $\{1 \dots |t|\}$ qui est compatible avec l'ordre usuel sur les entiers ($\leq_t \subseteq \leq$).*

La définition d'un *système de transition concurrent* est similaire à la définition dans le cas non concurrent (définition 5), sauf que les traces sont maintenant concurrentes.

Définition 21 (Système de transition concurrent)**.** Un système de transition *T est donc la donnée de $(|T|, (T_F, T_A, T_{DNF}))$, où $|T|$ est un ensemble de traces concurrentes, et (T_F, T_A, T_{DNF}) est un triplet de parties de $|T|$, qui déterminent le* statut *des traces, respectivement les traces qui terminent, celles qui aboutissent à une erreur, et les traces qui ne terminent pas.*

On étend la concaténation et le produit parallèle aux systèmes de transitions concurrents.

Concaténation Lorsqu'on concatène deux traces concurrentes t et t', chaque instruction de t est $\leq_{t \cdot t'}$ à chaque instruction de t'.

Produit parallèle Intuitivement, deux instructions de deux traces qu'on entrelace sont dites parallèles lorsque qu'elles ne sont pas synchronisées par une paire de $V(r)$ et $P(r)$. Plus formellement, si τ est un ordonnancement de deux traces t et t', c'est à dire que τ est une fonction bijective croissante de $\{1 \dots |t|\} + \{1 \dots |t'|\}$ vers $\{1 \dots |t| + |t'|\}$, et $\tilde{t} = \tau(t, t')$, on définit $\leq_{\tilde{t}}$ comme le plus petit ordre contenant :

— *Les ordres des traces* t *et* t', c'est à dire, plus formellement, que pour tout i, j tels que $i \leq_t j$, $\tau(\iota_l(i)) \leq_{\tilde{t}} \tau(\iota_l(j))$, et de même pour t' : si $i \leq_{t'} j$ alors $\tau(\iota_r(i)) \leq_{\tilde{t}} \tau(\iota_r(j))$;

— *L'ordre induit par la synchronisation* : si i et j sont tels que $\tau(\iota_l(i)) \leq_{\mathbb{N}} \tau(\iota_r(j))$ et si $m_i = V(r)$ et $m_j = P(r)$, alors $\tau(\iota_l(i)) \leq_{\tilde{t}} \tau(\iota_r(j))$. Et de même en échangeant les rôles des injections ι_l et ι_r.

Note : la première relation est le produit parallèle des deux relations d'ordre, et la seconde relation est la relation *synchronizes-with*, utilisée notamment dans le standard $C11$.

4.2 Nouvelle définition des jeux

Dans cette section, nous adaptons les jeux définis à la section 3.3 au cadre concurrent qu'on a décrit dans la section précédente. Désormais, les états sont séparés en trois parties, celle du code, celle de l'environnement, et les parties qui sont protégées par les verrous libres (qui ne sont pris par personne).

Soit Γ un contexte, C une commande, un triplet de la logique de séparation concurrente $\Gamma \vdash \{P\}\, C\, \{Q\}$ et $t \in [\![C]\!]$, une trace concurrente.

Définition 22 (Graphe des états séparés)**.** *Ses* sommets *sont tous les triplets*

$$(\sigma_C, \sigma_\Gamma, \sigma_F) \in \text{Etat} \times (\text{Res} \rightharpoonup_{fin} \text{Etat}) \times \text{Etat}$$

tels que $\sigma_C * \Pi_{r \in \text{dom}(\sigma_\Gamma)} \sigma_\Gamma(r) * \sigma_F$ *est bien défini, et ses* arcs *sont de deux types :*

— *les arcs Eve, de la forme* $(\sigma_C, \sigma_\Gamma, \sigma_F) \xrightarrow{r:\sigma_U} (\sigma'_C, \sigma_\Gamma \uplus [r \mapsto \sigma_U], \sigma_F)$,

— *les arcs Adam, de la forme* $(\sigma_C, \sigma_\Gamma, \sigma_F) \xrightarrow[\rho]{r:\sigma_L} (\sigma_C * \sigma_L, \sigma'_\Gamma, \sigma'_F)$, *si* $r \notin \text{dom}(\sigma'_\Gamma)$, *et* $\forall r' \in \text{dom}(\Gamma) \setminus (\rho \cup \{r\}), \sigma_\Gamma(r') = \sigma'_\Gamma(r')$,

où les σ *sont des états,* r *est un nom de ressource, ou, si aucune* r *ressource n'est échangée,* $r = \emptyset$ *et* σ_U *ou* σ_L *est* ***emp****, et* ρ *est un ensemble fini de ressources.*

L'étiquette ρ est l'ensemble des ressources que l'environnement a touché pendant sa transition. Comme dans le cas non concurrent, on définit le *graphe des prédicats séparés*, dont les sommets sont les triplets $\{P, \Delta, Q\}$ de prédicats, et Δ est un contexte, et qui dispose de deux types de transitions :

— Les transitions Joueur, étiquetées par un couple nom de ressource - prédicat,

— les transitions Opposant, étiquetées par un couple nom de ressource - prédicat.

On étend la relation $\vDash$ aux états et aux prédicats séparés de manière naturelle :

$$(\sigma_C, \sigma_\Gamma, \sigma_F) \vDash \{P, \Delta, Q\} \overset{\text{def}}{\iff} \sigma_C \vDash P \text{ et } \sigma_F \vDash Q \text{ et } \forall r \in \text{dom}(\sigma_\Gamma), \sigma_\Gamma(r) \vDash \Delta(r).$$

On appelle *jeu de séparation* un chemin alternant (fini ou infini) du graphe des prédicats séparés commençant par Opposant.

La donnée de $\Gamma \vdash \{P\}\, C\, \{Q\}$, et $t \in [\![C]\!]$ définit un jeu de séparation, dont les étiquettes sont données en remplaçant chaque instruction m par $\Delta(\text{lock}^-(m))$ et chaque étiquette *env* par $\text{lock}^+(m) : \Delta(\text{lock}^+(m))$, où m est l'instruction qui vient juste après cette étiquette ; et dont tous les sommets sont $\{\mathbf{true}, \Gamma, \mathbf{true}\}$, sauf le premier, qui est $\{P, \Delta, \mathbf{true}\}$, et le dernier (s'il existe), qui est $\{Q, \Delta, \mathbf{true}\}$.

On adapte naturellement les définitions de la réalisation et des stratégies.

Définition 23 (Réalisation)**.** *Un arc* $(\sigma_C, \sigma_\Gamma, \sigma_F) \xrightarrow{r:\sigma} (\sigma'_C, \sigma'_\Gamma, \sigma'_F)$ *du graphe des états séparés* réalise *un arc* $\{P, \Delta', Q\} \xrightarrow{r:I} \{P', \Delta, Q'\}$ *du graphe des prédicats séparés lorsque :*

- *le premier est un arc Adam (respectivement Eve), et le second est un arc Opposant (respectivement Joueur),*
- σ *satisfait l'invariant* I*,*
- *chaque état séparé satisfait le prédicat séparé qui lui correspond (ie.* $(\sigma_C, \sigma_\Gamma, \sigma_F) \vDash \{P, \Delta, Q\}$*, et* $(\sigma'_C, \sigma'_\Gamma \sigma'_F) \vDash \{P', \Delta', Q'\}$*).*

On étend la relation de réalisation aux paires de chemins alternés par recollement.

On peut alors définir les notions de stratégie et de stratégie gagnante comme dans la section précédente. Un triplet $\Gamma \vdash \{P\}\, C\, \{Q\}$ induit un ensemble de jeux de séparation : l'union de tous les jeux de séparation induits par le triplet et t, pour tout $t \in [\![C]\!]$.

Le théorème de correction reste vrai dans notre nouveau cadre.

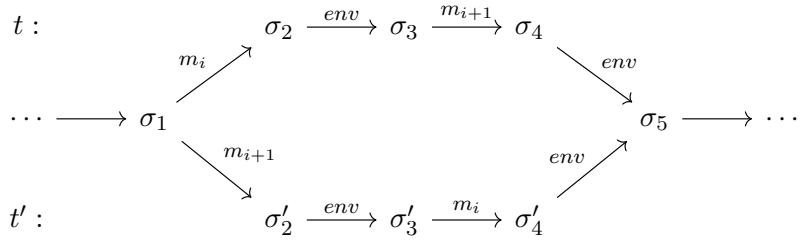
Théorème 2 (Correction (séquentielle))**.** *Soit un triplet de la logique de séparation concurrente* $\Delta \vdash \{P\}\, C\, \{Q\}$*, et* π *une preuve de ce triplet.*

Alors π *définit un stratégie gagnante pour chacun des jeux induit par* $\Gamma \vdash \{P\}\, C\, \{Q\}$*.*

Grâce à ce pistage plus fin des ressources, on va pouvoir exprimer qu'Eve doit être, en un certain sens que nous allons préciser, uniforme, c'est à dire que si m_i et m_{i+1} s'exécutent en parallèle, Eve relève m_i et m_{i+1} de manières semblables, que m_i s'exécute avant m_{i+1} ou le contraire. Pour parler d'uniformité, il faut pourvoir comparer les stratégies associées à différentes traces de la même commande : on appelle *stratégie globale pour* $\Gamma \vdash \{P\}\, C\, \{Q\}$ une fonction `STRAT` telle que pour tout $t \in [\![C]\!]$, $\texttt{STRAT}(t)$ est une stratégie pour le jeu induit par $\Gamma \vdash \{P\}\, C\, \{Q\}$ et t. Naturellement, on dit que `STRAT` est gagnante si pour tout $t \in [\![C]\!]$, $\texttt{STRAT}(t)$ est gagnante.

Définition 24 (instructions parallèles)**.** *Soient t une trace, et m_i, m_j deux instructions de t. On dit que m_i et m_j sont parallèles, et on note $m_i \# m_j$ si $\neg(m_i \leq m_j) \wedge \neg(m_j \leq m_i)$.*

Définition 25 (Échange)**.** *Soient t une trace, et m, m_{i+1} deux instructions parallèles et consécutives de t. Alors toute trace t' comme dans la figure ci-dessous est dite* une trace où m_i et m_{i+1} sont échangées dans t.



(1)

Soient $\Gamma \vdash \{P\}\, C\, \{Q\}$ un triplet, et $t \in [\![C]\!]$, d'après le théorème de correction, il existe une stratégie gagnante `strat` du jeu induit par ce triplet et t. Soient m_i et m_{i+1} deux instructions consécutives et parallèles de t. Notons les coups d'Eve et d'Adam qui les concernent :

$$(\sigma_C^{(1)}, \sigma_\Gamma^{(1)}, \sigma_F^{(1)}) \xrightarrow{\sigma_U} (\sigma_C^{(2)}, \sigma_\Gamma^{(2)}, \sigma_F^{(2)}) \xrightarrow[\rho_1]{\sigma_L} (\sigma_C^{(3)}, \sigma_\Gamma^{(3)}, \sigma_F^{(3)}) \xrightarrow{\sigma'_U} (\sigma_C^{(4)}, \sigma_\Gamma^{(4)}, \sigma_F^{(4)}) \xrightarrow[\rho_2]{\sigma'_L} (\sigma_C^{(5)}, \sigma_\Gamma^{(5)}, \sigma_F^{(5)}) \tag{2}$$

Supposons de plus que les différentes transitions touchent des verrous différents : ρ_1, ρ_2, $\mathrm{lock}(m_i)$, et $\mathrm{lock}(m_{i+1})$ sont deux-à-deux disjoints, avec $\mathrm{lock}(m) = \mathrm{lock}^+(m) \cup \mathrm{lock}^-(m)$.

Définition 26 (Voisinage)**.** *Soient* `STRAT` *une stratégie globale pour $\Gamma \vdash \{P\}\, C\, \{Q\}$, $t \in [\![C]\!]$, m une instruction de t et $\mathcal{O}$ un état. On dit que $\mathcal{O}$ est un voisinage pour m et* `STRAT` *si, pour tout R tel que $\mathcal{O} * R$ est défini, il existe $\mathcal{O}'$ tel que $\mathcal{O} * R \rightsquigarrow^{m} \mathcal{O}' * R$, en particulier, il n'y a pas d'erreur.*

De plus, à chaque fois que `STRAT` *doit réaliser une transition de cette forme dans une trace t' qui est une trace où m et m' (ou m' et m) ont été échangées dans t (au sens de la définition 25), il existe un chemin du graphe des états séparés où Eve joue un coup de la forme :*

- Si m n'est ni de la forme $P(r)$ ni de la forme $V(r)$, $(\mathcal{O} * R, \sigma_\Gamma, \sigma_F) \xrightarrow{r:\sigma_U} (\mathcal{O}' * R, \sigma'_\Gamma, \sigma_F)$,
- Si $m = R(v)$, $(\mathcal{O} * R, \sigma_\Gamma, \sigma_F) \xrightarrow{r:\sigma} (\mathcal{O}'' * R, \sigma_\Gamma \uplus [r \mapsto \sigma], \sigma_F)$, *avec* $\mathcal{O} = \sigma_1 * \mathcal{O}''$,
- Si $m = P(r)$, $(\mathcal{O} * R, \sigma_\Gamma, \sigma_F) \xrightarrow{\emptyset:\boldsymbol{emp}} (\mathcal{O} * R, \sigma_\Gamma, \sigma_F)$.

Comme on l'a dit plus haut, on veut que les stratégies induites par les preuves soient uniformes, plus précisément, pour toute trace t, et toute instruction m de t, la stratégie doit associer à m un voisinage $\mathcal{O}^t(m)$ tels que, si on a deux traces t et t' comme représentées à la figure (1), alors on veut que $\mathcal{O}^t(m_i) = \mathcal{O}^{t'}(m_i)$ et $\mathcal{O}^t(m_{i+1}) = \mathcal{O}^{t'}(m_{i+1})$, c'est à dire qu'Eve n'est pas sensible à l'ordre dans lequel elle voit deux instructions parallèles.

On va montrer que dans ce cas, on peut faire commuter les instructions m_i et m_{i+1} au niveau des traces.

Lemme 9. *Soit une trace $t \in [\![C]\!]$, un triplet valide $\Gamma \vdash \{P\}\, C\, \{Q\}$, `STRAT` une stratégie gagnante pour les jeux induits par ce triplet et des instructions m_i et m_{i+1} de t comme dans la figure 2. Alors il existe des voisinages $\mathcal{O}^t(m_i) \subseteq \sigma_C^{(1)}$ et $\mathcal{O}^t(m_{i+1}) \subseteq \sigma_C^{(3)}$ des instructions m_i et m_{i+1} dans t tels que leur produit est défini.*

Démonstration. On reprend la stratégie `STRAT` qui a été définie dans la section 3. La règle la plus importante est Par (traité en dans la section 3.5), car c'est le produit parallèle qui fait apparaître les instructions parallèles.

Reprenons donc la preuve du lemme 3 où est construite la stratégie dans le cas où la commande C est de la forme $C_1 \parallel C_2$ et la dernière règle utilisée dans la preuve est la règle Par. On dispose donc d'une stratégie, c'est à dire d'un ensemble de chemin, dans le graphe des états triplements séparés (bien que dans le cas concurrent, il possède quatre composantes, puisque qu'il y a σ_Γ en plus).

Soient donc $\tilde{t}$ un entrelacement de $t_1 \in [\![C_1]\!]$ et de $t_2 \in [\![C_2]\!]$, et soit m une instruction de $\tilde{t}$. Dans $\tilde{t}$, il y a donc une transition de la forme $\sigma \xrightarrow{m} \sigma'$. Soit p un chemin qui réalise un préfixe de t qui contient m. Notons le coup qui réalise la transition étiquetée par m (si, par exemple, m provient de la trace t) :

$$(\sigma_1, \sigma_2, \sigma_\Gamma, \sigma_F) \xrightarrow[1]{r:\sigma_U} (\sigma_1', \sigma_2, \sigma_\Gamma, \sigma_F)$$

Nous prétendons que σ_1 est un voisinage pour m dans t. En effet, si on échange m avec une autre instruction, parallèle et telle que m et m' sont consécutives, dans le graphe des états triplement séparés, dans les deux cas, la première composante sera la même, et donc la stratégie se comportera de la même manière.

De plus, si m_i et m_{i+1} sont deux instructions parallèles et consécutives, alors leurs voisinages (qu'on vient de définir) sont compatibles, par définition des noeuds du graphe des états triplement séparés.

Pour toutes les autres règles, le résultat découle du fait que les voisinages en entrés le sont toujours en sortie. En effet, soit on concatène des traces, ce qui séquentialise (il n'y a donc pas plus d'instructions parallèles), soit on modifie les chemins du graphe des états séparés (par exemple pour la règle Par), mais à chaque fois, les voisinages sont conservés. □

Grâce à ce lemme, on pourra montrer qu'une commande bien typée s'exécute toujours sans accès concurrent. Tout d'abord, définissons ce qu'est qu'un accès concurrent.

Définition 27 (Accès concurrent)**.** *Soit C une commande, et $t \in [\![C]\!]$ une trace concurrente. t contient un accès concurrent lorsque qu'il existe deux instructions m_i et m_j (distinctes) telles que : (i) m_i et m_j sont parallèles, (ii) m_i écrit dans une variable ou une adresse mémoire qui est lue ou écrite par m_j, ou le contraire.*

Corollaire 2. *Soit $\vdash \{P\}\, C\, \{Q\}$ un triplet valide, et $t \in [\![C]\!]$ où l'environnement reste passif. Soient m_i et m_{i+1} deux instructions parallèles consécutives dans t.*

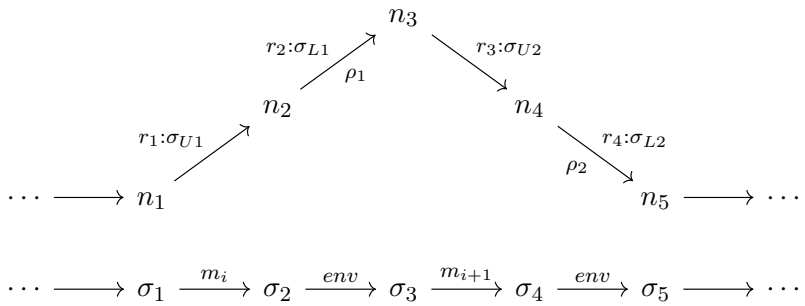
Alors m_i et m_{i+1} n'effectuent pas d'accès concurrents

Démonstration. Soient m_i et m_{i+1} deux instructions parallèles et consécutives de t. Alors d'après le lemme 9, on dispose de deux voisinages $\mathcal{O}_1$ et $\mathcal{O}_2$ de m_i et de m_{i+1} dans t respectivement. De plus leurs produit est défini. Il est aisé de voir que si, par exemple, m_i écrit dans $l \in \operatorname{hdom} \mathcal{O}_1$ (et nécessairment, si m_i écrit dans une adresse, elle doit être dans son voisinage), $l \notin \operatorname{hdom} \mathcal{O}_2$. Il n'y a donc pas d'accès concurrent. □

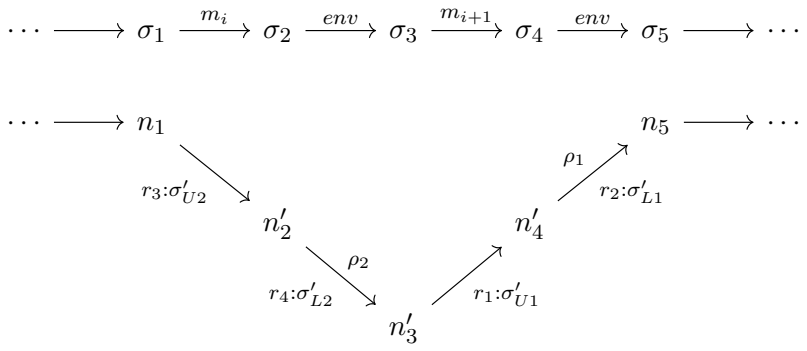
On peut alors montrer le résultat souhaité de déterminisme dans les jeux.

Théorème 3 (Commutation)**.** *Soient $\Gamma \vdash \{P\}\, C\, \{Q\}$ un triplet valide, $t, t' \in [\![C]\!]$ comme dans la figure 1. Appelons G et G' les jeux associés à ce triplet et, respectivement, aux traces t et t'.*

Pour tout chemin p du graphe des états séparés qui réalise un préfixe g (de taille au moins 4) de G de la forme :



de trace sous-jacente t. Notons g' le préfixe de G' qui a la même taille que g. Alors il existe un chemin p' qui réalise t' qui est égal à p sauf sur les quatre coups qui réalisent m_i, m_{i+1} et les coups de l'environnement qui les suivent. Le chemin p' est décrit par la figure suivante.



De plus, on dispose de voisinages $\mathcal{O}^t(m_i)$ et $\mathcal{O}^t(m_{i+1})$ dans t et $\mathcal{O}^{t'}(m_i)$ et $\mathcal{O}^{t'}(m_{i+1})$ dans t' tels que $\mathcal{O}^t(m_i) = \mathcal{O}^{t'}(m_i)$ et $\mathcal{O}^t(m_{i+1}) = \mathcal{O}^{t'}(m_{i+1})$.

Démonstration. On écrit, pour chaque noeud de p, $n_i = (\sigma_C^{(i)}, \sigma_\Gamma^{(i)}, \sigma_F^{(i)})$.

Supposons dans un premier temps que ni m_i ni m_{i+1} ne sont de la forme $V(r)$ ou de la forme $P(r)$.

D'après le lemme 9, il existe des voisinages $\mathcal{O}_i \subseteq \sigma_C^{(1)}$ de m_i et $\mathcal{O}_{i+1} \subseteq \sigma_C^{(3)}$ de m_{i+1}. On peut donc écrire $\sigma_C^{(1)} = \mathcal{O}_i * R_1$, $\sigma_C^{(3)} = \sigma_C^{(2)} = O'_1 * R_1$, $\sigma_C^{(3)} = \mathcal{O}_{i+1} * R_2$, $\sigma_C^{(4)} = \sigma_C^{(5)} = O'_2 * R_2$.

On a : $\mathcal{O}_i * \mathcal{O}_{i+1} \subseteq \sigma_C^{(i)}$, ie : $\exists$R, $\mathcal{O}_i * \mathcal{O}_{i+1} * R = \sigma_C^{(i)}$.

Donc on a :

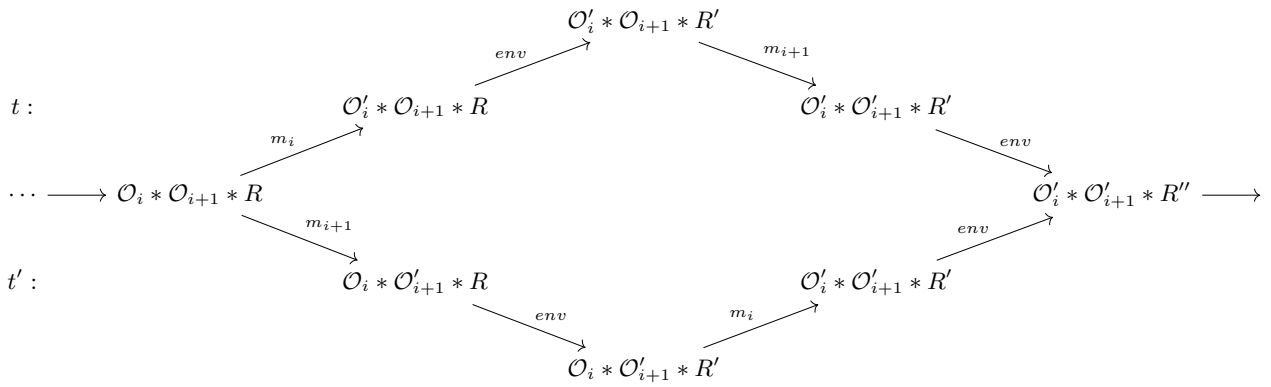
$$\mathcal{O}_i * \mathcal{O}_{i+1} * R \overset{m_{i+1}}{\rightsquigarrow} \mathcal{O}_i * O'_{i+1} * R$$

puis

$$\mathcal{O}_i * O'_{i+1} * R \overset{m_i}{\rightsquigarrow} O'_i * O'_{i+1} * R \ .$$

De plus, par définition des voisinages, chacune des ces deux relations sont vraies pour tout sous-état R qui est compatible avec $\mathcal{O}_i * \mathcal{O}_{i+1}$.

Par complétude (lemme 1), $[\![C]\!]$ contient la trace égale à t en tout points, sauf sur les transitions m_i et m_{i+1} où elle est définie par :



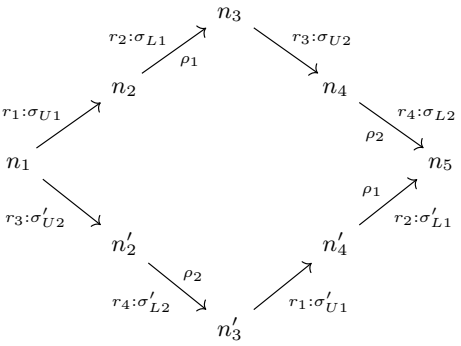
On peut vérifier qu'il est possible de réaliser la trace t', étant donné que la structure des états dans t et t' se reflète dans le graphe des états séparés.

Grâce à la dernière partie de la définition d'un voisinage (définition 26), la manière qu'on a de réaliser t' est la même que celle de $\texttt{STRAT}(t')$.

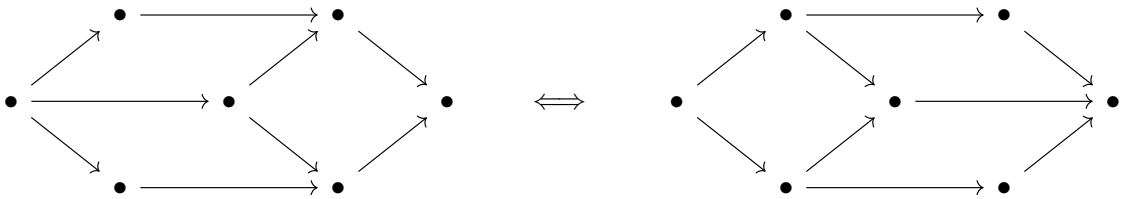
Supposons maintenant que $m_i = V(r)$ ou $m_{i+1} = V(r)$. Considérons par exemple la première alternative. Alors, par définition de deux instructions parallèles, m_{i+1} n'est pas une instruction de la forme $P(r)$ ou $V(r)$. Le lemme 9 nous indique l'on dispose de voisinages pour $V(r)$ et pour m_{i+1} dont le produit est défini. Alors comme dans le cas précédent, on peut faire commuter les deux instructions.

Enfin, dans le cas où l'une des instruction est de la forme $P(r)$, comme par hypothèse l'environnement ne change pas le tas associé à la ressource r (car $r \notin \rho_1 \cup \rho_2$), on sait qu'Adam n'a d'autre choix que de jouer $\sigma'_{U2} = \sigma_{U2}$. □

Dans ce cas, on dit que la figure suivante est une *tuile*.



La stratégie vérifie l'axiome du cube, où chaque parallélogramme est une tuile comme ci-dessus, et les bords (sommets et arrêtes) sont les mêmes.



Par suite, on peut étendre l'absence d'accès concurrents des paires d'instructions parallèles adjacentes à toutes les paires d'instructions parallèles.

5 Conclusion et travaux futurs

Le formalisme que nous avons proposé dans ce rapport nous parait très prometteur, notamment car il fait un lien entre les traditions des logiques de programmes et du model-checking.

Tout d'abord, nous comptons formaliser la preuve décrite dans ce rapport dans un assistant de preuve. L'ambition est que la preuve soit modulaire, et formalise tout à la fois la logique de séparation concurrente et les jeux de model-checking. Il existe des travaux qui s'en approchent, par exemple, cette année, [6] décrit une formalisation en Coq des automates bidirectionnels et [15] formalise les automates temporisés.

Une autre direction sera de développer notre méthode de preuve, pour pouvoir traiter les variantes d'ordres supérieures de la logique de séparation concurrente. Ces variantes permettent de quantifier sur les triplets de Hoare, ce qui est utile pour spécifier les fonctions d'ordre supérieure (pour spécifier ce qu'on suppose sur les paramètres formels qui sont des fonctions).

La sémantique de notre langage est *séquentiellement cohérente*, c'est à dire que le résultat d'un calcul est le même que si les instructions des différents processus s'étaient exécutés séquentiellement dans une certain ordre. Néanmoins, ce modèle n'est pas réaliste : en pratique, la présence des caches dans les processeurs empêche cette propriété d'être satisfaite. On se trouve en face de modèles de mémoires faibles. Le problème de trouver des logiques pour raisonner dans ces modèles mémoire est encore ouvert. Notre formalisme pourrait être bien adapté : par exemple, Boudol et Petri [3] et Jeffrey [7] formalisent ces modèles mémoires au moyen de structures d'évènements.

Une manière de voir notre formalisme est qu'une stratégie jouée par Eve et/ou Adam est un raffinement de la trace d'exécution jouée par le code et/ou l'environnement. On se trouve donc dans un système de raffinement de type, dont l'originalité est d'être à deux Joueurs, et de nature interactive. Il serait intéressant de relier nos travaux et ceux de Melliès et Zeilberger sur les raffinements de types [9].

Enfin, ce pont entre les logiques de programmes et le model-checking nous laisse entrevoir la possibilité de combiner logique de séparation concurrente et logiques de spécifications inductives et coinductives.

Références

[1] Luca Aceto. « Interview with Stephen Brookes and Peter W. O'Hearn Recipients of the 2016 Godel Prize ». In : *Bulletin of EATCS* 1.119 (2016).

[2] Richard Bornat, Cristiano Calcagno et Hongseok Yang. « Variables as Resource in Separation Logic ». In : *Electr. Notes Theor. Comput. Sci.* 155 (2006), p. 247–276.

[3] Gérard Boudol et Gustavo Petri. « Relaxed Memory Models : An Operational Approach ». In : POPL '09. New York, NY, USA : ACM, 2009, p. 392–403.

[4] Stephen Brookes. « A Revisionist History of Concurrent Separation Logic ». In : *Electronic Notes in Theoretical Computer Science* 276 (2011), p. 5–28.

[5] Stephen Brookes. « A semantics for concurrent separation logic ». In : *International Conference on Concurrency Theory*. Springer. 2004, p. 16–34.

[6] Christian Doczkal et Gert Smolka. « Two-Way Automata in Coq ». In : ITP '16. Cham : Springer International Publishing, 2016, p. 151–166.

[7] A. S. A. Jeffrey et J. Riely. « On Thin Air Reads : Towards an Event Structures Model of Relaxed Memory ». In : *Proc. IEEE Logic in Computer Science*. 2016.

[8] Cliff B Jones. « Specification and Design of (Parallel) Programs. » In : *IFIP congress*. T. 83. 1983, p. 321–332.

[9] Paul-André Melliès et Noam Zeilberger. « Functors Are Type Refinement Systems ». In : POPL '15. 2015, p. 3–16.

[10] Susan Owicki et David Gries. « An axiomatic proof technique for parallel programs I ». In : *Acta Informatica* 6.4 (1976), p. 319–340.

[11] Matthew J. Parkinson, Richard Bornat et Cristiano Calcagno. « Variables as Resource in Hoare Logics ». In : *21th IEEE Symposium on Logic in Computer Science (LICS 2006), Proceedings*. 2006, p. 137–146.

[12] François Pottier. « Syntactic soundness proof of a type-and-capability system with hidden state ». In : *J. Funct. Program.* 23.1 (2013), p. 38–144.

[13] John C Reynolds. « Separation logic : A logic for shared mutable data structures ». In : *Logic in Computer Science, 2002. Proceedings. 17th Annual IEEE Symposium on*. IEEE. 2002, p. 55–74.

[14] Viktor Vafeiadis. « Concurrent Separation Logic and Operational Semantics ». In : *Electr. Notes Theor. Comput. Sci.* 276 (2011), p. 335–351.

[15] Simon Wimmer. « Formalized Timed Automata ». In : ITP '16. Cham, 2016, p. 425–440.