La stratégie de la fourchette

Simon Castellan †

†: *ENS de Lyon, Laboratoire LIP, CNRS, Inria, UCBL, Université de Lyon*
46 allée d'Italie, 69364 Lyon cedex 07
simon.castellan@ens-lyon.fr

Résumé

Dans cet article, on se propose d'utiliser les développements récents de sémantique des jeux basées sur des ordres partiels [12, 3, 2] pour donner une sémantique dénotationnelle à une primitive de concurrence inspirée de l'appel système UNIX `fork`. Cet appel système, lors de son invocation, duplique le contexte d'exécution de son programme, et renvoie deux entiers distincts permettant au programme de savoir dans quelle dent de la fourche il se trouve.

La première contribution de cet article est de donner une sémantique opérationnelle faisant droit au parfum opérateur de contrôle de cette primitive qui manipule le contexte d'exécution. La seconde contribution de ce papier est l'élaboration d'une sémantique des jeux « vraiment concurrente » qui donne une représentation fine de l'exécution du langage.

Cette sémantique des jeux est prouvée correcte vis-à-vis de la sémantique opérationnelle et permet de faire un premier pas vers la modélisation de comportements concurrents complexes en utilisant les jeux concurrents.

1. Introduction

L'appel système UNIX L'appel système UNIX a été conçu historiquement comme seul moyen primitif de création de processus, en dupliquant le processus appelant (le père), ces deux copies poursuivent leur exécution et reprennent au retour de `fork`, mais voient deux valeurs de retour différentes. Ainsi une invocation typique de `fork` est :

```
if Unix.fork () = 0 then (* Code fils *) else (* Code père *)
```

Concurrence et valeurs de retour multiples La primitive `fork` permet donc la concurrence au sein d'un programme, mais de façon différente de ce qu'on rencontre d'habitude en sémantique. En effet, il s'agit d'une inversion de contrôle vis-à-vis de la composition parallèle $\cdot \parallel \cdot$ où c'est l'**appelant** qui décide d'exécuter plusieurs fois la même fonction avec des arguments différents. Avec `fork`, c'est l'**appelé** qui choisit d'exécuter plusieurs fois en parallèle la fonction à laquelle il est donné, en retournant plusieurs valeurs. Par exemple, la fonction suivante :

```
let deux_et_trois () = if Unix.fork () = 0 then 2 else 3
```

renvoie deux et trois en même temps, l'exécution du code `f (deux_et_trois ())` déclenchera l'exécution de f avec la valeur deux et la valeur trois en parallèle.

Non-déterminisme Dès qu'il est possible de faire communiquer les branches d'un `fork`, on peut créer des *races* qui créent du non-déterminisme de la façon suivante [1] :

1. Ici on utilise le système de fichiers pour coller aux implémentations car `fork` duplique la mémoire, mais le moyen de communication n'a pas d'importance pour la sémantique.

```
let choice (file: string) =
  write file 0;
  if Unix.fork () = 0 then (read file = 1) else (write file 1; exit 0)
```

`choice` renvoie vrai ou faux selon si la lecture est exécutée avant ou après l'écriture dans le processus père. L'`exit 0` permet de ne laisser qu'un seul processus en vie après exécution de `choice`. (Les nombres d'entrelacements des effets qui font que `choice` renvoie `tt` ou `ff` ne sont pas les mêmes mais ça ne nous concerne pas ici.)

Une sémantique opérationnelle et dénotationnelle On se propose dans cet article de donner une étude sémantique d'un langage fonctionnel disposant de cette primitive. Comme à notre connaissance il n'existe aucune modélisation sémantique de ce comportement précis de `fork`, on commence par donner une sémantique opérationnelle simple à petit pas de ce langage. Ensuite, et c'est le principal objectif, on cherche à en donner une sémantique dénotationnelle, et en particulier une sémantique des jeux. On a vu qu'un tel langage exhibera des comportements concurrents et non-déterministes. Nous souhaitons en donner une sémantique « vraiment concurrente », en d'autres termes, nous voulons que nos stratégies soient donc des ordres partiels enrichis avec des informations de non-déterminisme, ce que sont exactement les structures d'événements [13]. En effet, en mettant l'accent sur les causalités qui sous-tendent le comportement du programme, les structures d'événements permettent de distinguer clairement concurrence (absence de causalité et possible concomitance) du non-déterminisme (exécution conjointe impossible).

Le développement récent des sémantiques des jeux basées sur les structures d'événements [12, 2, 3] fournit un cadre idéal pour construire notre sémantique, cadre qui étend d'autres sémantiques des jeux vraiment concurrente [10, 6] avec du non-déterminisme. À noter aussi la sémantique des jeux pour des langages basés sur CCS en utilisant des pré-faisceaux [5]. Notre langage étend *Concurrent Idealized Algol* (*Idealized Algol* avec la possibilité d'exécuter deux commandes en parallèle) pour lequel un modèle pleinement abstrait vis-à-vis du *may testing* existe [7].

2. Le calcul de la fourche

Dans cette section on introduit le calcul de la fourche qui est notre langage idéalisé pour étudier la sémantique de `fork`. La première abstraction qu'on s'autorise est d'oublier la notion de *PID* et de considérer une version booléenne de `fork` :

```
let fourche () = Unix.fork () == 0
```

À l'aide cette primitive, on peut combiner de façon concurrente plusieurs termes en une *fourche* :

$$\begin{aligned}[t_1, \dots, t_n] = \texttt{if fourche () then } & t_1 \\ \texttt{else if fourche () then } & \dots \\ \texttt{else if fourche () then } & t_{n-1} \texttt{ else } t_n\end{aligned}$$

On choisit de prendre cette opération $[\cdot, \dots, \cdot]$ comme primitive de notre langage formel à la place de `fourche`, parce qu'elle permet de représenter explicitement une exécution de plusieurs processus en parallèle ce qui rend plus aisée la définition de la sémantique opérationnelle. On ne perd pas en expressivité puisque l'on a let `fourche` $() = [\texttt{ff}, \texttt{tt}]$. On ajoute aussi la fourche vide $[]$ qui représente l'arrêt du processus courant (le `exit 0` de la section précédente). On pense $[\_]$ comme un multi-ensemble de termes s'exécutant en parallèle. (On veut quand même pouvoir distinguer `tt` et $[\texttt{tt}, \texttt{tt}]$)

Sans mécanisme de communication entre les contextes, cette concurrence aurait les mêmes propriétés sémantiques que le non-déterminisme et ne serait donc pas très intéressante. Nous avons

ainsi choisi de baser notre calcul sur Idealized Algol [11], un λ-calcul simplement typé étendu avec des références, pour lequel de solides bases sémantiques existent (voir [1]).

2.1. Syntaxe et typage

Les grammaires des types et des termes sont données par

$$\begin{array}{lll} A, B ::= \texttt{nat} \mid \texttt{bool} \mid \texttt{unit} & & \text{(types de base)} \\ \quad \mid \texttt{var} & & \text{(références)} \\ \quad \mid A \Rightarrow B & & \text{(types flèches)} \end{array} \quad \left| \quad \begin{array}{ll} t, u ::= x \mid \lambda x.\ t \mid t\ u & \text{(λ-calcul simplement typé)} \\ \quad \mid c & \text{(constantes)} \\ \quad \mid \texttt{new}\ r := k\ \texttt{in}\ t & \text{(déclaration d'une référence)} \\ \quad \mid [t_1, \ldots, t_n] & \text{(fourches ($n \geq 0$))} \end{array} \right.$$

À noter que les fourches ne sont possibles que sur un type de base. On raisonne sur la syntaxe à associativité et commutativité près de la construction $[\cdot, \ldots, \cdot]$ plus l'équation $[x] = x$. On utilisera dans la suite la notation X, Y pour dénoter des types de base. Les constantes du langages sont :

- constantes des types de base : $\texttt{tt}, \texttt{ff}$ de type $\texttt{bool}$, et une constante $\bar{n}$ de type $\texttt{nat}$ pour chaque entier n et $()$ de type $\texttt{unit}$
- destructeurs des types de base : succ : $\texttt{nat} \Rightarrow \texttt{nat}$, pred : $\texttt{nat} \Rightarrow \texttt{nat}$, iszero : $\texttt{nat} \Rightarrow \texttt{bool}$, $\texttt{if \_ then \_ else \_} : \texttt{bool} \Rightarrow X \Rightarrow X$ et $\texttt{\_ ; \_} : \texttt{unit} \Rightarrow X \Rightarrow X$.
- point fixe : $Y : (A \Rightarrow A) \Rightarrow A$ pour tout type A
- opérations impératives : $\texttt{\_ := \_} : \texttt{var} \Rightarrow \texttt{nat} \Rightarrow \texttt{unit}$ et $\texttt{!\_} : \texttt{var} \Rightarrow \texttt{nat}$.

Les règles de typage sont celles données par le λ-calcul simplement typé avec les constantes mentionnées, plus la règle du *new* et des fourches :

$$\frac{\Gamma, r : \texttt{var} \vdash t : X}{\Gamma \vdash \texttt{new}\ r := k\ \texttt{in}\ t : X} \qquad \frac{\Gamma \vdash t_1 : X \quad \ldots \quad \Gamma \vdash t_n : X}{\Gamma \vdash [t_1, \ldots, t_n] : X}$$

Bien que les fourches aient un type de base, il est facile de définir les fourches pour les types flèches en définissant

$$[f, g] = \lambda x.[f\ x, g\ x]$$

2.2. Sémantique opérationnelle

On donne à présent une sémantique opérationnelle pour ce langage, à petit pas à cause des phénomènes de concurrence en s'inspirant des sémantiques données dans [1, 7]. Cette sémantique simule l'exécution des programmes par une machine séquentielle avec une mémoire infinie. La définition de la sémantique opérationnelle se fait en deux étapes :

- Une relation $\equiv$ qui est la plus petite congruence contenant $(\lambda x.\ t)\ u \equiv t[u/x]$ et $Y\ M \equiv M\ (Y\ M)$, qui permet la réduction aux types d'ordre supérieurs et ignore complètement les effets et les traite comme des constantes.
- Une relation $\Gamma \vdash t, \rho \to t', \rho'$ où Γ ne contient que des références, $\Gamma \vdash t, t' : X$ et $\rho, \rho' : \mathrm{dom}(\Gamma) \to \mathbb{N}$ représentent des environnements, qui réduit les effets

Les règles sont données en figure 1 et sont divisées en trois :

- les règles liées au fragment impératif du langage (basé sur *Idealized Algol*)
- la règle Duplication (et Effacement qui est l'équivalent pour $n = 0$) qui permet de propager la duplication quand on atteint un destructeur (représenté par les contextes d'évaluation). La duplication ne peut avoir lieu que sous un contexte linéaire de tête.
- la règle Préemption qui permet de choisir non-déterministiquement le prochain terme à évaluer (c'est la seule règle non-déterministe du langage)

Les *valeurs* d'un type de base X seront les termes de la forme $[x_1, \ldots, x_n]$ où les x_i sont des éléments de l'ensemble représenté par le type X.

$$\textsc{succ}\ \frac{}{\Gamma \vdash \mathrm{succ}\,\bar{n}, \rho \to \overline{n+1}, \rho} \qquad \textsc{pred}\ \frac{}{\Gamma \vdash \mathrm{pred}\,\overline{n+1}, \rho \to \bar{n}, \rho} \qquad \textsc{pred2}\ \frac{}{\Gamma \vdash \mathrm{pred}\,\bar{0}, \rho \to \bar{0}, \rho}$$

$$\textsc{iszero}\ \frac{}{\Gamma \vdash \mathrm{iszero}\,\bar{0}, \rho \to \mathtt{tt}, \rho} \qquad \textsc{iszero2}\ \frac{}{\Gamma \vdash \mathrm{iszero}\,\overline{n+1}, \rho \to \mathtt{ff},}$$

$$\textsc{if}\ \frac{}{\Gamma \vdash \mathtt{if\ tt\ then}\ t\ \mathtt{else}\ u, \rho \to t, \rho} \qquad \textsc{if2}\ \frac{}{\Gamma \vdash \mathtt{if\ ff\ then}\ t\ \mathtt{else}\ u, \rho \to u, \rho}$$

$$\textsc{Séq}\ \frac{}{\Gamma \vdash (); t, \rho \to t, \rho} \qquad \textsc{New1}\ \frac{\Gamma, r : \mathtt{var} \vdash t, \rho \to t', \rho'}{\Gamma \vdash \mathtt{new}\ r := \rho(r)\ \mathtt{in}\ t, \rho \setminus r \to \mathtt{new}\ r := \rho'(r)\ \mathtt{in}\ t', \rho' \setminus r}$$

$$\textsc{New2}\ \frac{r \notin t}{\Gamma \vdash \mathtt{new}\ r := k\ \mathtt{in}\ t, \rho \to t, \rho} \qquad \textsc{Déréf}\ \frac{\rho(r) = k}{\Gamma \vdash\ !r, \rho \to \bar{k}, \rho}$$

$$\textsc{Assign}\ \frac{}{\Gamma \vdash r := \bar{k}, \rho \to (), (r \mapsto k) :: \rho} \qquad \textsc{Congruence}\ \frac{\Gamma \vdash t, \rho \to t', \rho'}{\Gamma \vdash E(t), \rho \to E(t'), \rho'}$$

$$\textsc{Duplication}\ \frac{}{\Gamma \vdash E([t_1, \ldots, t_n]), \rho \to [E(t_1), \ldots, E(t_n)], \rho} \qquad \textsc{Effacement}\ \frac{}{\Gamma \vdash E([]), \rho \to [], \rho}$$

$$\textsc{Préemption}\ \frac{\Gamma \vdash t, \rho \to t', \rho'}{\Gamma \vdash [t_1, \ldots, t_i, t, t_{i+1}, \ldots, t_n], \rho \to [t_1, \ldots, t_i, t', t_{i+1}, \ldots, t_n], \rho'}$$

$$\beta\textsc{-réd}\ \frac{t \equiv t' \qquad \Gamma \vdash t, \rho \to u, \rho' \qquad u \equiv u'}{\Gamma \vdash t', \rho \to u', \rho'}$$

$$\begin{aligned} E ::= \quad & \textbf{contextes d'évaluation} \\ & \mid \_ \mid \mathrm{succ}\,E \mid \mathrm{pred}\,E \mid \mathtt{if}\ E\ \mathtt{then}\ t\ \mathtt{else}\ u \mid \mathrm{iszero}\,E \\ & \mid\ !E \mid x := E \mid E := t \mid E;\ t \end{aligned}$$

FIGURE 1 – Sémantique opérationnelle

2.3. Expressivité du langage

Dans cette partie, on s'intéresse à étudier l'expressivité du calcul de la fourche. Grâce au point fixe, on code la boucle `while` et les fonctions récursives usuelles sur les entiers.

Un langage parallèle À partir des fourches, on retrouve une composition parallèle de deux termes t, u de type `unit`, qui termine après que t et u ont terminé :

$$t \parallel u \;=\; \texttt{new } d \;:=\; 0 \texttt{ in [(}t\texttt{; } d \texttt{ := 1; [])} \quad , \quad \texttt{(}u\texttt{; while (}d \texttt{ == 0) \{ () \})]}$$

Cette opération satisfait :

$$\frac{}{\vdash () \parallel (), \rho \to^* (), \rho} \qquad \frac{\Gamma \vdash t, \rho \to t', \rho'}{\Gamma \vdash (t \parallel u), \rho \to (t' \parallel u), \rho'} \qquad \frac{\Gamma \vdash u, \rho \to u', \rho'}{\Gamma \vdash (t \parallel u), \rho \to (t \parallel u'), \rho'}$$

Un langage non-déterminisme Ce langage permet également d'exprimer le non-déterminisme, en effet l'opérateur de choix peut se coder ainsi comme on l'a vu à la section 1 :

$$\texttt{choix := new } s \;:=\; 0 \texttt{ in [!s == 1} \quad , \quad \texttt{(s := 1); []]}$$

L'opérateur `choix` se différencie de `fourche` : `choice` renvoie vrai **ou** faux, alors que `fourche` renvoie vrai **et** faux.

3. Une sémantique des jeux Hyland-Ong concurrente

Dans cette section, on présente la catégorie cartésienne fermée construite dans [2] qui ajoute aux jeux concurrents de [12] la possibilité de duplication des coups, et ainsi la possibilité de modéliser des langages avec multiples occurrences d'une même variable. Cette exposition laisse de côté une partie des détails techniques et ne présente que le strict nécessaire à la compréhension du modèle donné à la section suivante.

3.1. Types et arènes

Tout comme en sémantique des jeux traditionnelle, les types seront interprétés par des jeux à deux joueurs habituellement nommés Joueur/Opposant ou Programme/Environnement. Un jeu est donné par une collection de coups (ou événements), chaque coup se voyant attribuer une polarité positive ($\oplus$) ou négative ($\ominus$). Les règles du jeu sont spécifiées par une relation d'ordre $\leq$ sur les coups qui représente la *causalité*. Si $a \leq a'$, alors pour jouer le coup a', a doit déjà avoir été joué.

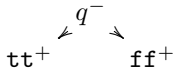
Si $a < a'$ sans événement entre les deux on dira que (a, a') est une causalité (immédiate) et on notera $a \rightarrowtriangle a'$. Nos types ayant une structure très simple, les jeux que nous devons considérer sont élémentaires et ont une structures d'*arène* :

***Definition* 1** (Arène)**.** — *Une arène est un ordre $(A, \leq)$ où chaque coup $a \in A$ a une polarité pol(a) à valeur dans $\{\ominus, \oplus\}$ satisfaisant les axiomes suivants :*

- Alternance : *Si on a $a \rightarrowtriangle a'$, alors a et a' ont une polarité différente.*
- Branchement forestier : *Si a n'est pas minimal, alors il existe un unique a_0 (le* justifieur *de a) tel que $a_0 \rightarrowtriangle a$ (en sémantique des jeux HO, on écrit alors $a_0 \vdash a$)*
- Finitude *L'ensemble $\{a' \mid a' \leq a\}$ (noté $[a]$ par la suite) est fini pour tout a*

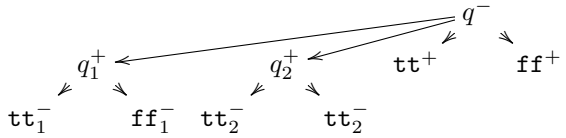
Une arène est de plus négative si tous ses événements minimaux sont négatifs.

La négativité est nécessaire parce que notre langage est en appel-par-nom : c'est à Opposant que revient la décision d'évaluer un terme, c'est pour cette raison que les types seront interprétés par des arènes négatives. Le type `bool` est par exemple interprété par l'arène suivante



Dans les dessins que nous faisons, la polarité des événements est indiquée en exposant et seule la causalité immédiate est représentée par une flèche (on retrouve l'ordre par clôture transitive). L'événement minimal est appelé q car on peut le voir comme une question que pose Opposant à Joueur, lui demandant sa valeur. Joueur peut alors jouer les coups `tt` et `ff` qui seront ainsi vus comme des réponses à la question initiale d'Opposant.

Étant données deux arènes A et B, on peut former comme d'habitude en sémantique des jeux les arènes $A \parallel B$ et $A \Rightarrow B$ qui représentent le produit et le type flèche de notre langage. Par exemple, voici l'arène $\texttt{bool}_1 \Rightarrow \texttt{bool}_2 \Rightarrow \texttt{bool}$ avec des indices aux différentes copies de `bool` pour plus de clarté :



Sur cette arène, Joueur a plus de possibilités : quand Opposant lui pose la question initiale, il peut ne pas lui répondre immédiatement et commencer par lui poser une question sur $\texttt{bool}_1$ ou $\texttt{bool}_2$ pour connaître la valeur de ses arguments.

Ces constructions sont définies de la manière suivante, où l'on utilise une notation additive pour les unions disjointes ensemblistes ($\cdot + \cdot$ et $\sum_{a \in A} \cdot$) et $\iota_1, \iota_2, \iota_a$ pour les injections.

***Definition* 2** (Arène produit et flèche)**.** — *Si A et B sont des arènes, on définit l'arène produit $A \parallel B$ par*

— Ordre causal *: $A + B$ avec l'ordre induit par A et B sans causalités supplémentaires*

— Polarités *: Induites par A et B*

et l'arène flèche $A \Rightarrow B$ par

— Ordre causal *: $\left(\sum_{b \in \min(B)} A\right) + B$, où $\min(B)$ est l'ensemble des coups minimaux de B, muni de l'ordre induit par A et les copies de B adjoint des causalités $\iota_2(b) \rightarrowtriangle \iota_1(\iota_b(a))$ pour a et b minimaux dans A et B,*

— Polarités *: $\iota_2(b) = pol(b)$ et $pol_1(\iota_b(a)) = pol(a)^\perp$ avec $\ominus^\perp = \oplus, \oplus^\perp = \ominus$.*

3.2. Termes et stratégies

Un terme de type A est interprété par une stratégie sur l'arène correspondante à A. Une stratégie joue en ajoutant des causalités absentes du jeu (avant de répondre $\texttt{tt}^+$, j'attends qu'Opposant m'ait répondu $\texttt{tt}_1^-$) et en ignorant des événements positifs (je ne joue jamais $\texttt{ff}^+$). Habituellement en sémantique des jeux séquentielle, on définit une stratégie comme un ensemble de séquences de coups expliquant comment la stratégie joue. Le séquencement induit un ordre total sur le déroulement de la partie, ce qui n'est pas adapté à un cadre « vraiment concurrent » : on souhaite qu'une stratégie puisse jouer deux coups en même temps (sans liens causaux), *ie.* que nos parties soient des ordres partiels. Nos stratégies forment alors des sous-ensembles de l'arène avec un ordre causal enrichissant celle de l'arène.

Un autre problème auquel on doit faire face dans notre langage est le non-déterminisme. Pour l'exprimer, l'ordre de la stratégie doit être enrichi par une relation de conflit binaire notée $a \sharp a'$ qui

signifie que deux coups ne seront jamais joués tous les deux dans une même exécution par la stratégie (bien sûr, a et a' ne doivent pas être reliés causalement pour que cela ait du sens), représentant un choix non-déterministe entre a et a'. Cela nous mène naturellement à la notion de structure d'événements :

***Definition* 3** (Structure d'événements)**.** — *Une structure d'événements est un triplet* $(S, \leq, \sharp)$ *où* $(S, \leq)$ *est un ordre partiel et* $\sharp \subseteq S^2$ *est une relation irréflexive symétrique satisfaisant*

- Finitude : *Pour* $s \in S$, $[s]$ *est fini*
- Héritage du conflit : *Si* $s \sharp s'$ *et* $s \leq s''$ *alors* $s'' \sharp s'$

Une arène est un cas particulier de structure d'événements sans conflits. Un sous-ensemble x clos vers le bas pour $\leq$ et dont les éléments sont deux-à-deux compatibles (c'est-à-dire pas en conflit) est appelé une configuration, l'ensemble des configurations finies de S sera noté $\mathscr{C}(S)$. Une stratégie sera donc une structure d'événements S où chaque événement sera étiqueté par un coup de A, *ie.* munie d'une fonction $\sigma : S \to A$. σ sera parfois utilisée pour faire référence à la structure d'événements S.

Pour que σ représente une stratégie valide, on impose les conditions suivantes :

Respect des règles du jeu σ doit respecter les causalités imposées par le jeu. Elle ne peut pas jouer un événement avant d'avoir joué son justifieur.

Injectivité locale Une configuration de S doit représenter une partie (qui peut être incomplète) du jeu. Hors, dans nos jeux, on ne peut jouer un événement qu'une seule fois, donc une configuration de S doit être en bijection avec sa projection dans le jeu. Une autre façon de voir ça est la suivante : si la partie courante est x, et que σ joue un événement s, on veut que cette extension soit projetée dans le jeu vers une extension de σx, sinon le coup joué par σ serait invisible dans le jeu.

Courtoisie Une stratégie n'est pas libre d'ajouter toutes les causalités immédiates qu'elle souhaite. Les seules causalités $a \rightarrowtriangle a'$ qu'elle peut ajouter sont celles où a est négatif et a' positif : attendre un coup Opposant avant de jouer un certain coup Joueur. En particulier elle ne peut pas faire attendre Opposant si le jeu ne le permet pas, ni imposer un ordre d'arrivée entre deux événements positifs si cet ordre n'est pas présent dans le jeu.

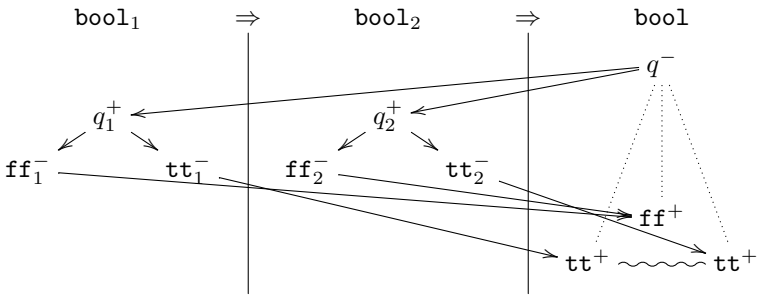
Réceptivité On interdit à la stratégie de modifier le comportement d'Opposant : si les règles du jeu autorisent Opposant à jouer un coup à la suite d'un état atteint par la stratégie, alors la stratégie ne peut l'en empêcher (le coup ne peut donc pas être absent de S).

En résumé, on donne la définition suivante des stratégies :

***Definition* 4** (Stratégie)**.** — *Une stratégie sur l'arène* $(A, \leq_A)$ *est une structure d'événements* $(S, \leq_S, \sharp)$ *avec une fonction* $\sigma : S \to A$ *satisfaisant les axiomes suivants :*

- Respect des règles : *Si* $x \in \mathscr{C}(S)$ *alors* $\sigma x \in \mathscr{C}(A)$.
- Injectivité locale : *Si* $s, s' \in S$ *sont tels que* $\sigma s = \sigma s'$ *alors* $s \sharp s'$.
- Courtoisie : *Si* $s \rightarrowtriangle_S s'$ *et* $\neg(\sigma s \rightarrowtriangle \sigma s')$ *alors* $(pol(\sigma s), pol(\sigma s')) = (\ominus, \oplus)$
- Réceptivité : *Pour* $x \in \mathscr{C}(S)$ *et* $a \in A$ *négatif tel que* $\sigma x \cup \{a\} \in \mathscr{C}(S)$ *alors il existe un unique* $s \in S$ *tel que* $\sigma s = a$ *et* $x \cup \{s\} \in \mathscr{C}(A)$.

À noter que ces conditions naturelles apparaissent aussi si l'on désire munir ces stratégies d'une structure de catégorie, comme ce qui est fait dans [12]. À titre d'exemple, voici une stratégie qui représente l'opération *parallel-or*. On dessine la structure d'événement étiquetée, $\rightarrow$ représente la causalité immédiate, $\sim$ le conflit, et $\cdots$ relie un coup à son justifieur, ou plus formellement, le coup qui est projeté vers le justifieur du premier événement dans le jeu.



La stratégie commence par interroger ses deux arguments *en parallèle*, car q_1 et q_2 sont causalement indépendants. Alors, selon la réponse d'Opposant [2] elle répond :

— On répond $\mathtt{ff}^+$ si Opposant a répondu $\mathtt{ff}_1^-$ **et** $\mathtt{ff}_2^-$

— On répond $\mathtt{tt}^+$ si Opposant a répondu $\mathtt{tt}_1^-$ **ou** $\mathtt{tt}_2^-$. On est obligé de faire deux copies du $\mathtt{tt}^+$ (en conflit car on ne veut répondre qu'une fois), car il y a deux raisons pour lesquelles on peut être amené à répondre $\mathtt{tt}^+$.

3.3. Indices de copie

Dans les jeux présentés ci-dessus, une stratégie peut jouer un coup une fois ou zéro fois. Ainsi une fonction ne peut appeler son argument qu'une seule fois et ne retourner qu'une seule fois une valeur donnée, ce qui nous empêche de modéliser le calcul de la fourche ou même le λ-calcul simplement typé. La solution que l'on adopte ici est inspirée des séquences de pointeurs des jeux HO, mais adaptée à un cadre concurrent. À partir d'une arène A, on définit une arène $!A$ où les coups de A sont dupliqués en profondeur. Pour chaque coup de $a \in A$, on duplique chacune des causes au sens large de a : un coup de $!A$ sera donné par un coup a de A et un choix d'un indice de copie pour chaque cause de a, c'est-à-dire une fonction $[a] = \{a' | a' \leq a\} \to \mathbb{N}$.

***Definition* 5** (Arène $!A$)**.** — *Si A est une arène, on définit une arène $!A$:*

— Ordre causal : *Fonctions de la forme $\alpha : [a] \to \mathbb{N}$ où $a \in A$ ($\alpha(a)$ est appelé l'*indice de α*), ordonné par l'inclusion des graphes, c'est-à-dire $(\alpha : [a] \to \mathbb{N}) \leq (\beta : [b] \to \mathbb{N})$ lorsque $a \leq b$ et pour $a' \leq a, \alpha(a) = \beta(a)$.*

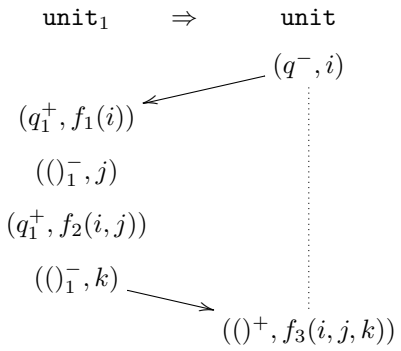
— Polarité *La polarité de $\alpha : [a] \to \mathbb{N}$ est la même que celle de a dans A*

Une stratégie jouant sur l'arène $!A$ peut jouer un coup de A plusieurs fois, ce qui permet de poser plusieurs fois la même question (ie. invoquer une variable plusieurs fois). Cependant, la réceptivité force alors la structure d'événements à être infinie. Ainsi, lorsque l'on dessine une telle stratégie, on ne dessine qu'une seule copie de chaque coup négatif avec pour indice de copie une variable. En faisant l'union de tous ces dessins, on retrouve la stratégie totale.

Si $a \in A$ a pour justifieur a_0, alors une fonction de choix $\alpha : [a] \to \mathbb{N}$ correspond exactement à un choix d'indice $\alpha(a)$ pour a et une fonction de choix pour a_0. Donc, quand on représente des éléments de $!A$, on a besoin de représenter uniquement leur indice à condition de représenter leur justifieur.

À titre d'exemple, voici une stratégie, correspondant au terme $\lambda x.\ x; x$.

2. Ou plutôt, selon *les réponses* d'Opposant car la stratégie décrite n'empêche pas Opposant de répondre vrai et faux à ses deux arguments (elle n'en a pas le droit par réceptivité de toute façon)



Pour alléger les dessins, un événement dessiné directement représenté au-dessus d'un autre est son seul antécédent causal immédiat, sans qu'on ait besoin de représenter la causalité explicitement.

Le choix des fonctions f_1, f_2, f_3 est laissé à la stratégie, cependant l'injectivité locale impose des contraintes. L'une d'elles est que f_2 doit dépendre injectivement de j. En effet, deux coups de $!A$ sont les mêmes si les fonctions de copie sont les mêmes. En particulier deux copies de q_1^+ sont identiques si leurs indices sont égaux et si l'indice de la copie de q^- qui les cause sont les mêmes (car $[q_1^-] = \{q^-, q_1^+\}$). Donc si Opposant joue deux copies de $()_1^-$ avec deux indices de copie j_1 et j_2 différents, alors pour que l'on réponde par des copies distinctes de q_1 il faut que $f_2(i, j_1) \neq f_2(i, j_2)$. Le même raisonnement tient pour f_3 qui doit être injective en j et k. Le choix $f_1(i) = 0, f_2(i, j) = j + 1$ et $f_3(i, j, k) = \langle j, k\rangle$ satisfait ces contraintes où $\langle \_ \rangle : \mathbb{N}^* \to \mathbb{N}$ est une injection.

Plus tard, on considérera les stratégies à choix d'indices de copie près, tant qu'un tel choix existe.

3.4. Cadre catégorique

Pour interpréter des langages fonctionnels dans ce cadre, on cherche à le munir d'une structure catégorique. Nos objets seront les arènes, et un morphisme d'une arène A vers une arène B sera une stratégie jouant sur une arène $B - A$ (en suivant la tradition de Conway et Joyal [8]). Dans notre cadre, on définit $B - A = !A^\perp \parallel !B$ où $A^\perp$ est l'arène A avec les polarités inversées. Même si A et B sont négatives, $B - A$ ne l'est pas forcément, il nous faut donc imposer que nos stratégies le soient, c'est-à-dire que ses coups minimaux soient dans B. On verra cependant qu'on peut quand même se ramener à une arène négative (cf. lemme 2) nécessaire afin d'établir la fermeture de la catégorie.

Composition Il nous faut décrire la composition d'une stratégie $\sigma : S \to (!A)^\perp \parallel !B$ avec une stratégie $\tau : T \to (!B)^\perp \parallel !C$. Pour ce faire, on procède comme en sémantique des jeux habituelle et on procède en deux étapes :

— On calcule d'abord l'interaction $\tau \circledast \sigma : T \circledast S \to !A \parallel !B \parallel !C$

— Ensuite on dissimule les événements de l'interaction projetés sur $!B$ en définissant la structure d'événements correspondant à la composition

$$T \odot S = \{p \in T \circledast S \mid (\tau \circledast \sigma)(p) \in !A \parallel !C\},$$

l'ordre causal et le conflit étant hérités de $T \circledast S$ (la définition formelle est donnée dans [2]).

La partie difficile est donc la définition de l'interaction. Elle est formellement définie au travers d'un *pullback* entre les stratégies $\sigma \parallel C$ et $A \parallel \tau$, mais on peut décrire plus simplement les configurations finies de $S \circledast T$:

***Lemme* 1.** — *Les configurations finies de $S \circledast T$ correspondent à l'ensemble des bijections φ de la forme (avec $x \in \mathscr{C}(S)$ et $y \in \mathscr{C}(T)$) telles que $\sigma x \cap !B = !B \cap \tau y$) :*

$$x \parallel (\tau y \cap !C) \cong \sigma x \parallel (\tau y \cap !C) = (\sigma x \cap !A) \parallel \tau y \cong (\sigma x \cap !A) \parallel \tau y$$

qui sont acycliques*, c'est-à-dire telle que la clôture transitive de la relation sur les couples* $(s, \varphi s)$ *donnée par « $s \leq s'$ ou $\varphi s \leq \varphi s'$ » soit un ordre, ce qui correspond à l'absence de* deadlocks *entre x et y.*

Les *deadlocks* peuvent apparaître lorsque dans l'arène B il y a deux coups concurrents a_0^- et a_1^+ de polarités opposées : σ jouant sur $!B$ peut ajouter la causalité $a_0^- \rightarrowtriangle a_1^+$, mais τ jouant sur $!B^\perp$ peut elle ajouter la causalité $a_1^+ \rightarrowtriangle a_0^-$. Intuitivement σ attend qu'Opposant joue a_0 pour jouer a_1 et τ attend qu'Opposant joue a_1 pour jouer a_0 : si on fait interagir σ avec τ rien ne se passe et l'interaction est vide (il n'y a pas de bijections acycliques).

Identité La stratégie jouant le rôle de l'identité dans notre cadre est comme traditionnellement en sémantique des jeux, une stratégie *copycat*. Pour la décrire, on donne une stratégie $\mathfrak{c}_A : \mathbb{CC}_A \to A^\perp \parallel A$, qui se borne à recopier de l'autre côté les coups négatifs joués par Opposant d'un côté. La structure d'événements $\mathbb{CC}_A$ est obtenue à partir de $A^\perp \parallel A$ en adjoignant les causalités immédiates $\iota_1(a) \rightarrowtriangle \iota_2(a)$ pour tout coup positif a de l'arène, et $\iota_2(a) \to \iota_1(a)$ pour tout coup négatif de l'arène.

3.5. Uniformité et équivalence faible

Malgré quelques contraintes engendrées par l'injectivité locale, une stratégie dispose d'une grande liberté dans le choix des indices de copie des coups positifs qu'elle joue. Il est alors naturel de vouloir identifier deux stratégies qui ne diffèrent que par leur choix d'indices de copie positifs (c'est aussi nécessaire pour obtenir une catégorie cartésienne fermée et donc un modèle du λ-calcul simplement typé).

Sans autres conditions sur les stratégies, il est clair qu'une telle notion ne pourra pas être préservée par composition. En effet, considérons la stratégie τ jouant sur l'arène $!\texttt{bool}^\perp \parallel !\texttt{bool}$ qui, informellement, évalue son argument, et si Opposant lui répond sur un indice de copie pair alors renvoie `tt` sinon renvoie `ff`. Considérons alors σ_1 et σ_2 qui jouent sur $!\texttt{bool}$ et qui répondent toutes les deux `tt` à la question initiale, mais σ_1 avec un indice pair et σ_2 avec un indice impair. Alors σ_1 et σ_2 devraient être considérées équivalentes mais $\sigma_1; \tau$ et $\sigma_2; \tau$ ne peuvent pas l'être car la première répond `tt` et la second répond `ff`.

Le problème est que τ se comporte différemment selon les indices des coups Opposant qu'elle reçoit. On cherche donc à le bannir au travers de la notion d'uniformité. On ne cherche pas à détailler la définition d'une stratégie uniforme ici, elle est donnée dans [2] et définie au travers de la notion de structure d'événements avec symétrie.

On utilisera donc les notions suivantes comme des boîtes noires dans la suite :

— La notion de « témoin d'uniformité » (*thin symmetry* dans [2]). Une stratégie uniforme (*∼-strategy* dans [2]) est une stratégie équipée d'un témoin d'uniformité ;

— La notion d'équivalence faible entre stratégies uniformes, notée $\sigma \simeq \tau$ qui est une congruence identifiant les stratégies différant uniquement par leur choix d'indice de copie ;

— La notion de stratégie uniforme *filaire* (qui sert un but similaire à la notion correspondante dans les jeux HO : garantir que nos stratégies soient des morphismes de comonoïdes).

À noter que étant donnée une stratégie, le témoin d'uniformité n'est pas unique, et que la notion d'équivalence faible est un peu plus subtil que la simple identification des stratégies à indices de copie positifs près, puisqu'elle s'assure également que les témoins d'uniformité sont compatibles.

Ces définitions mènent au théorème suivant :

***Théorème* 1** (de [2])**.** — *La structure suivante est une catégorie cartésienne fermée, notée* CHO *:*

— Objets *: Arènes négatives (le produit est donné par $\parallel$ et la flèche par la construction $\Rightarrow$ décrite à la section 3.1)*

— Morphismes de A vers B *: Stratégies négatives uniformes et filaires jouant sur $!A^\perp \parallel !B$, considérées à équivalence faible près*

Le produit de la catégorie est donnée par le produit d'arène $\cdot \parallel \cdot$ et l'exponentiation est donnée par la flèche sur les arènes $\cdot \Rightarrow \cdot$. La clôture est une conséquence du lemme suivant :

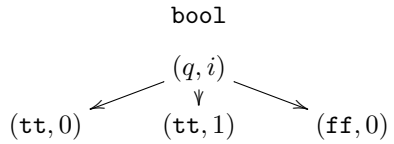
***Lemme* 2.** — *Il y a une bijection qui respecte la composition entre les stratégies uniformes filaires et négatives sur* $!A^{\perp} \parallel !B$ *et les stratégies uniformes et filaires sur* $!(A \Rightarrow B)$ *(qui sont nécessairement négatives).*

4. Un modèle du calcul de la fourche

En utilisant la technologie présentée à la section précédente, on s'attaque maintenant à la modélisation de notre langage à l'intérieur de CHO. Comme noté dans [2], cette catégorie est déjà un modèle de PCF. Comme cette interprétation de PCF coïncide avec celle des jeux HO usuels (coïncidence prouvée dans [2]), on ne s'étend pas dessus. Dans cette partie, on explique comment modéliser la concurrence et l'état partagé dans ce cadre.

4.1. Concurrence

Pour interpréter la concurrence de notre langage, il faut expliquer comment une stratégie peut renvoyer plusieurs résultats. En effet, $[\texttt{tt}, \texttt{tt}, \texttt{ff}]$ est une valeur de type `bool` qui retourne deux fois `tt` et une fois `ff`. Comme dans le jeu $!A$ on duplique en profondeur, les réponses (les coups `tt` et `ff`) sont aussi dupliquées et une stratégie peut jouer plusieurs copies différentes de `tt`. Ainsi, la stratégie suivante représente $[\texttt{tt}, \texttt{tt}, \texttt{ff}]$:



Ainsi, pour former $[\sigma_1, \ldots, \sigma_n]$ il suffit de mettre en parallèle les réponses possibles apportées par les σ_i. Il faut cependant faire attention aux indices de copie, et faire en sorte que les réponses jouées par les σ_i aient des indices différents. Notons S_i la structure d'événements correspondant à la stratégie S_i de sorte que $\sigma_i : S_i \to !A$. On cherche maintenant à construire $\sigma : S \to !A$ qui représente la mise en parallèle des σ_i en assimilant les coups négatifs. Pour cela, quitte à remplacer les σ_i par des σ'_i qui leur sont faiblement équivalentes, on peut faire les trois hypothèses suivantes :

— les événements minimaux (forcément négatifs) des S_i coïncident : si $s \in S_i$ est est minimal alors $s \in S_j$ et $\sigma_i s = \sigma_j s$ pour tout $j \leq n$
— les événements non minimaux des S_i sont disjoints : si $s \in S_i$ n'est pas minimal alors il n'appartient à aucun autre S_j
— les images positives des σ_i sont disjointes, si $s \in S_i$ et $s' \in S_j$ sont positifs, alors $\sigma s \neq \sigma s'$ (cela est possible en changeant les indices de copie jouées par les σ_i)

Dans ce cas, on peut former l'union $S = S_1 \cup \ldots \cup S_n$ qui hérite de la causalité et les conflits des S_i. La projection $\sigma : S \to A$ est alors définie par :

— si $s \in S$ est minimal, on pose $\sigma s = \sigma_1 s = \ldots = \sigma_n s$
— si $s \in S$ n'est pas minimal et appartient à S_k (k est unique par hypothèse), $\sigma s = \sigma_k s$

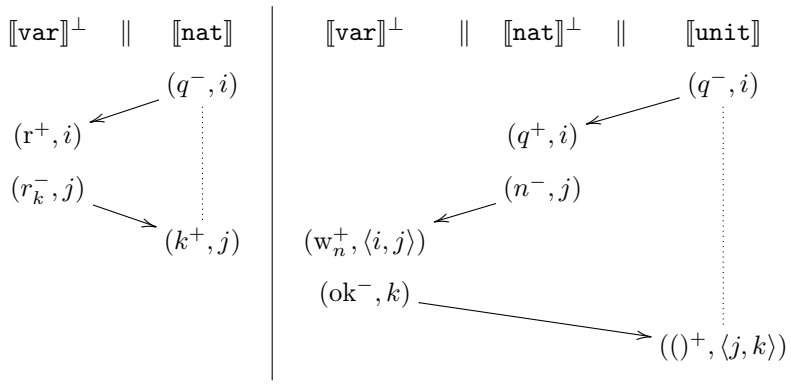
Par construction σ est bien localement injective et on peut vérifier qu'il existe un témoin d'uniformité lui donnant la structure d'une stratégie uniforme.

En notant $[\sigma_1, \ldots, \sigma_n]$ pour σ on sait à présent interpréter le terme $[t_1, \ldots, t_n]$ comme $[[\![t_1]\!], \ldots, [\![t_n]\!]]$. Le phénomène de duplication provient du fait que par réceptivité, une stratégie doit être prêt à réagir à un Opposant qui répond plusieurs valeurs en parallèle, et donc quand la stratégie demande la valeur de son argument, Opposant peut décider de démarrer plusieurs fils d'exécution en répondant plusieurs réponses différentes.

4.2. État

On s'attaque maintenant à la modélisation des primitives liées à l'état dans notre langage. Pour cela, on suit la méthodologie de [1]. Tout d'abord, il nous faut donner l'arène qui interprétera le type $[\![\texttt{var}]\!]$. Il s'agit de l'arène suivante :

Opposant peut soit demander à lire la référence (par le coup r), et dans ce cas Joueur doit lui répondre sa valeur, ou bien peut demander à écrire une valeur k (par le coup w_k) et dans ce cas Joueur répond pour lui signaler que l'écriture est terminée. En utilisant cette arène, on peut à présent donner des stratégies pour représenter les opérations d'assignement (assign : $[\![\texttt{var}]\!]^\perp \parallel [\![\texttt{nat}]\!]$), et de déréférenciation (get : $[\![\texttt{var}]\!]^\perp \parallel [\![\texttt{nat}]\!]^\perp \parallel [\![\texttt{nat}]\!]$), les voici :



Ces stratégies, comme remarqué dans [1] se comportent comme des programmes fonctionnels purs. Toute la complexité de l'état provient de l'interprétation du `new`. En effet, tant que la référence est abstraite (sans implémentation) comme une variable d'un contexte, les opérations de lecture ne font qu'envoyer des commandes lecture/écriture à la référence. Cependant, quand on créé vraiment la référence avec `new`, il faut fournir une implémentation de la référence et répondre de manière cohérente aux commandes envoyées par le terme. À cet effet, on cherche à décrire la sémantique de $\texttt{new}\ r := k\ \texttt{in}\ t$ comme suit :

$$[\![\Gamma \vdash \texttt{new}\ r := k\ \texttt{in}\ t : X]\!] = [\![\Gamma]\!] \xrightarrow{\langle id, \mathrm{ref}_k\rangle} [\![\Gamma, r : \texttt{var}]\!] \xrightarrow{[\![\Gamma, r:\texttt{var} \vdash t:X]\!]} [\![X]\!]$$

où ref_n sont toutes des pré-stratégies jouant sur $![\![var]\!]$ (pré-stratégie au sens où elles ne satisfont pas les axiomes de stratégies, on vérifiera par contre que la composition ci-dessus en est une), et qui représente une référence initialisée à n.

La stratégie ref_n est donc responsable d'implémenter la mémoire et de répondre aux requêtes du terme (la duplication des coups ici permet au terme de lancer plusieurs commandes). Ici, on définit une stratégie qui implémente une mémoire qui séquentialise les écritures et les lectures. Quand deux écritures concurrentes sont envoyées à la mémoire, un ordre est choisi non déterministiquement. Ainsi toutes les configurations de la stratégie seront des ordres totaux représentant une séquentialisation possible des accès mémoires reçus.

On cherche à construire la structure d'événements R_n sous-jacente à cette pré-stratégie. Les événements de R_n doivent retenir en mémoire l'histoire des opérations qui ont été faites sur la mémoire

afin d'ajouter les bons liens causaux entre lecture et écriture. La notion de séquence d'opérations est définie au travers de la grammaire suivante paramétrée par un ensemble d'indices qui sont les indices des coups négatifs déjà rencontrés (pour éviter des problèmes d'injectivité), et un entier qui représente la valeur initiale.

$$\begin{aligned} cell_n(l) := {} & (\mathrm{r}^-, i) \cdot (r_n^+, \langle l \rangle) \cdot (cell_n(l \cup \{2i\})) \\ & \mid (\mathrm{w}_k^-, i) \cdot (ok^+, \langle l \rangle) \cdot (cell_k(l \cup \{2i+1\})) \\ & \mid \emptyset \end{aligned}$$

$cell_n(l)$ représente donc un ensemble de séquences d'éléments de $!\llbracket \mathtt{var} \rrbracket$. On peut ensuite définir R_n ainsi :

— *Ordre causal* : l'ensemble des séquences non vides de $cell_n(\emptyset)$ ordonné par l'ordre préfixe
— *Conflit* : comme on désire que nos configurations soient des ordres totaux, on déclare que deux événements non comparables pour l'ordre préfixe sont en conflit.

On définit ensuite la fonction $\mathrm{ref}_n : R_n \to !\llbracket \mathtt{var} \rrbracket$ qui à une séquence associe son dernier élément. Ce n'est pas une stratégie, en effet, dans Cell_n on a des causalités non courtoises de la forme $w_k^- \cdot ok^+ \twoheadrightarrow w_k^- \cdot ok^+ \cdot r^-$. Elle n'est pas non plus réceptive (l'unicité n'est pas vérifiée). Cependant sa composition avec une stratégie en est une.

Lemme 3. — *Soit σ une stratégie uniforme jouant sur $\llbracket \Gamma \rrbracket^\perp \parallel \llbracket \mathtt{var} \rrbracket^\perp \parallel !\llbracket X \rrbracket$ où X est un type de base. Alors $\sigma \odot \langle id, ref_n \rangle$ est une stratégie uniforme.*

Démonstration. On rappelle le résultat de [12] : $\tau : T \to A^\perp \parallel B$ est une stratégie ssi $\mathbb{c}_A ; \tau ; \mathbb{c}_B$ est isomorphe à τ (deux stratégies sont isomorphes lorsque les structures d'événements le sont et que cet isomorphisme commute avec les projections sur le jeu). Puisque cell_k a 1 pour domaine, on a $\mathbb{c}_1 ; \mathrm{cell}_k$ et la composition vérifie clairement le critère énoncé plus haut. L'uniformité découle de celle de σ □

À remarquer qu'on se base ici sur le fait que l'on peut définir la composition entre pré-stratégies qui ne sont pas des stratégies (mais elles doivent être des morphismes de structures d'événements, voir [12] pour plus de détails). En général, la composition n'est pas filaire, cependant il est possible de la rendre filaire en enlevant les événements qui sont dans deux fils (ie. au dessus de deux événements minimaux). On écrit cette construction $\sigma \mapsto \sigma'$ qui prend une stratégie uniforme et négative et renvoie une stratégie qui est filaire en plus.

Au final on définit l'interprétation de l'opérateur new ainsi :

$$\llbracket \Gamma \vdash \mathtt{new}\ r := k\ \mathtt{in}\ t : X \rrbracket = \left(\llbracket \Gamma \rrbracket \xrightarrow{\langle id, \mathrm{ref}_k \rangle} \llbracket \Gamma, r : \mathtt{var} \rrbracket \xrightarrow{\llbracket \Gamma, r : \mathtt{var} \vdash t : X \rrbracket} \llbracket X \rrbracket \right)'$$

4.3. Correction du modèle

On a donc explicité comment interpréter notre langage à l'intérieur de CHO. On s'intéresse maintenant à la comparaison du modèle avec la réduction à petit pas du langage définie à la section 2.2. Si on essaye d'appliquer la méthode usuelle, à savoir qu'un pas dans le langage laisse les dénotations invariantes (à équivalence faible près), on se heurte à plusieurs problèmes. Le premier est qu'un jugement de la sémantique opérationnelle a la forme $\Gamma \vdash t, \rho \to t', \rho' : X$, et donc ce ne sont pas $\llbracket t \rrbracket$ et $\llbracket t' \rrbracket$ qu'il faut comparer mais $\llbracket (t, \rho) \rrbracket$ et $\llbracket (t', \rho') \rrbracket$. Cependant ce n'est pas défini !

En suivant la méthodologie de [1], on définit l'interprétation d'un environnement $\Gamma \vdash \rho$ comme une stratégie $\llbracket \rho \rrbracket : 1 \to \llbracket \Gamma \rrbracket$ donnée par $\mathrm{ref}_{\rho(r_1)} \parallel \ldots \parallel \mathrm{ref}_{\rho(r_n)}$ où $\Gamma = (r_1 : \mathtt{var}, \ldots, r_n : \mathtt{var})$. Là, deux problèmes supplémentaires surgissent :

— Notre langage étant non-déterministe, lorsque l'on fait une transition on s'engage potentiellement dans un choix non-déterministe. Par exemple, on a vu que `choice` se réduit vers `tt` et `ff`

qui ne sont pas équivalents. Une façon de résoudre ce problème est donc d'affaiblir ce que l'on demande, on demande donc que (t, ρ) puisse simuler (t', ρ').

— Contrairement à un cadre séquentiel et déterministe, les termes sont en compétition pour accéder à la mémoire. Quand (t, ρ) peut faire un pas vers (t', ρ') cela signifie dans n'importe quel contexte concurrent (à cause de la règle Préemption), t, ρ *peut* faire un pas vers t', ρ'. Pour cela, la preuve a besoin de l'invariant que t peut avancer dans *tout contexte concurrent.*

Dans [7], pour résoudre le premier problème, le résultat de correction est énoncé vis-à-vis de l'inclusion de trace. Comme on a un modèle plus fin, il nous fat une notion de simulation plus subtile, basée sur les morphismes de structures d'événements ouverts [9] :

***Definition* 6.** — *Une stratégie uniforme $\sigma : S \to !A$ simule une stratégie uniforme $\tau : T \to !A$ s'il existe une fonction $f : T \to S$ telle que*

— *f préserve et reflète l'ordre et les conflits, localement injective et vérifie une propriété de* lifting *des extensions :*

$$\forall x \in \mathscr{C}(T), \forall y \in \mathscr{C}(S), fx \subseteq y \Rightarrow \exists x' \in \mathscr{C}(T), fx' = y \wedge x \subseteq x'$$

— *si x est une configuration de T, la bijection dont le graphe est $\{(\tau s, \sigma(fs)) | s \in x\}$ forme un isomorphisme d'ordre entre les configurations de $!A$ τx et $\sigma(fx)$ qui préserve les étiquettes dans A.*

On peut maintenant énoncer le résultat :

***Théorème* 2.** — *Supposons qu'on ait une étape de réduction $\Gamma \vdash (t, \rho) \to (t', \rho') : X$.*

Alors pour toute stratégie uniforme σ jouant sur $!\llbracket\Gamma\rrbracket^{\perp} \parallel !\llbracket A\rrbracket$, la stratégie $\llbracket\rho\rrbracket; \langle\llbracket t\rrbracket, \sigma\rangle$ jouant sur $!\llbracket X\rrbracket \parallel !\llbracket A\rrbracket$ simule $\llbracket\rho'\rrbracket; \langle\llbracket t'\rrbracket, \sigma\rangle$.

Démonstration. (Ébauche) La première étape est de remarquer que $\equiv$ se traduit par l'équivalence faible ce qui permet de valider la règle β-réd. Les règles liées à PCF proviennent de simples calculs de compositions qui ne posent pas de problèmes.

Pour les règles de duplication et d'effacement, on remarque que les opérations $x : X, y : X \vdash [x, y] : X$ et $\vdash [] : X$ induisent des stratégies $X \parallel X \to X$ et $1 \to X$ qui forment une structure de monoïde sur les arènes correspondant aux types de base. On montre ensuite que tous les contextes d'évaluation sont des morphismes de monoïdes ce qui permet de valider ces deux règles.

L'invariant permet de valider facilement la règle de préemption. Pour le fragment impératif, les règles concernant le new sont triviales. Les règles Assign et Déréf sont là où l'on a besoin des simulations. En effet, pour maintenir l'invariant dans tout contexte concurrent, on ne peut prouver qu'une simulation en lieu d'une équivalence faible.

□

La quantification sur σ représente le contexte concurrent évoqué ci-dessus et est nécessaire pour que l'induction fonctionne pour le cas de la règle Préemption.

Conclusion

On a présenté un langage de programmation idéalisé pour représenter l'expressivité calculatoire offerte par `fork` ainsi qu'une sémantique opérationnelle et dénotationnelle reliées par un théorème de correction, en montrant ainsi que les jeux concurrents sont un bon cadre pour construire des modèles de langages concurrents complexes. Dans le futur, on souhaiterait relier plus précisément les deux sémantiques en particulier relier la sémantique dénotationnelle avec une forme de bisimulation. Pour cela, il faut permettre au modèle d'être conscient des *deadlocks* par exemple via l'étude des configurations stoppantes faite dans [4].

Le modèle obtenu ici permet aussi de comparer l'expressivité de différents langages en comparant les classes de stratégies obtenues. Par exemple, si on montre que la classe des stratégies engendrées à Idealized Algol et la composition parallèle vérifie une propriété que ne vérifie pas `fourche`, alors on peut en déduire que ce dernier est strictement plus expressif.

Remerciements Merci à Pierre Clairambault et Olivier Laurent pour leurs nombreux conseils et relectures.

Références

[1] S. Abramsky and G. McCusker. Full abstraction for idealized algol with passive expressions. volume 227, pages 3–42. 1999.

[2] S. Castellan, P. Clairambault, and G. Winskel. Concurrent hyland-ong games, 2014.

[3] S. Castellan, P. Clairambault, and G. Winskel. Symmetry in concurrent games. In *LICS 2014*. IEEE Computer Society, 2014.

[4] S. Castellan, J. Hayman, M. Lasson, and G. Winskel. Strategies as concurrent processes. In *MFPS*, LNCS. Springer, 2014.

[5] C. Eberhart, T. Hirschowitz, and T. Seiller. Fully-abstract concurrent games for pi. *CoRR*, abs/1310.4306, 2013.

[6] C. Faggian and M. Piccolo. Partial orders, event structures and linear strategies. In *TLCA '09*, volume 5608 of *LNCS*. Springer, 2009.

[7] D. R. Ghica and A. S. Murawski. Angelic semantics of fine-grained concurrency, 2007.

[8] A. Joyal. Remarques sur la théorie des jeux à deux personnes. *Gazette des Sciences Mathematiques du Québec 1(4)*, pages 46 – 52, 1977.

[9] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Inf. Comput.*, 127(2) :164–185, 1996.

[10] P. Melliès and S. Mimram. Asynchronous games : Innocence without alternation. In *CONCUR 2007 - Concurrency Theory, 18th International Conference*, pages 395–411, 2007.

[11] J. C. Reynolds. Idealized algol and its specification logic. *Tools and notions for program construction*, pages 121–161, 1982.

[12] S. Rideau and G. Winskel. Concurrent strategies. In *Logic in Computer Science (LICS), 2011 26th Annual IEEE Symposium on*, pages 409–418. IEEE, 2011.

[13] G. Winskel. Event structures. In *Petri Nets : Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, 8.-19. September 1986*, pages 325–392, 1986.