4 Leçon 4

Nous continuons aujourd'hui nos efforts dans le but de décrire une sémantique aussi naturelle que possible de mini-Caml. Nous commençons par poursuivre la leçon 3, en donnant une sémantique dénotationnelle de mini-Caml en section 4.1. Nous verrons une sémantique déjà plus concrète en section 5.1, avec laquelle nous ferons le lien. Cette dernière sera une étape dans le processus qui nous permettra de définir un compilateur pour mini-Caml.

4.1 Sémantique dénotationnelle de mini-Caml

Nous allons définir une fonction $[\![\_]\!]_{\text{caml}}$ qui prend une expression ou un programme mini-Caml en entrée, et retourne sa valeur. Supposons que Val est un cpo contenant $[Val \to Val]$, à isomorphisme près, par exemple $\mathbb{P}\omega$.

Une expression mini-Caml dépend de variables, et l'on ne peut pas décider de la valeur des variables comme ça, ex abrupto. La fonction $[\![\_]\!]_{\text{caml}}$ va donc prendre un argument supplémentaire, un *environnement* $\rho \in Env = Var \to Val$, où Var est l'ensemble de toutes les variables. L'environnement dit, pour chaque variable x, quelle est la valeur $\rho(x)$ que nous lui donnons. La valeur d'une expression, par exemple `x+1`, sera interprétée dans un tel environnement : si $\rho(\mathtt{x}) = 3$, alors intuitivement la valeur $[\![\mathtt{x} + \mathtt{1}]\!]_{\text{caml}}\,\rho$ de `x+1` dans cet environnement sera 4.

Une expression mini-Caml dépend aussi de l'état courant de la mémoire... Oh, pas encore la mémoire au sens où nous l'avons vue en section 3.1.1, mais quelque chose de plus abstrait et en même temps de proche dans l'esprit. Le but sera de donner une sémantique aux expressions portant sur les références, $\mathtt{ref}\ M$, $!M$, $M{:=}N$.

Donnons-nous un ensemble *Addr* fini ou dénombrable quelconque. On appellera conventionnellement les éléments de *Addr* les *adresses*, et ceci devrait vous rappeler la section 3.1.1. Mais ici, nous allons être moins concrets : nous ne stockerons pas des octets à chaque adresse, mais des valeurs quelconques de notre cpo Val. De plus, nous allons considérer qu'il peut exister des adresses inutilisées, au sens où rien n'y est stocké. Une ***mémoire*** sera donc ici une fonction partielle de *Addr* vers Val. Mieux, on va même demander que les mémoires μ soient les éléments de l'ensemble $Mem = Addr \to_{fin} Val$ des fonctions *de domaine fini* de *Addr* vers Val. Ceci représentera qu'on ne peut stocker qu'une quantité finie d'information à tout moment.

Au total, la sémantique d'une expression mini-Caml M *dans un environnement* $\rho \in Env$ *et une mémoire* $\mu \in Mem$ sera un élément $[\![M]\!]_{\text{caml}}\,\rho\mu$ de $Mem \times Val$: une paire (μ', V) formée de la mémoire *après* l'exécution de l'expression, et de la valeur V retournée.

▸ **EXERCICE 4.1**
Montrer que, si Val est un cpo, alors Env et Mem aussi, pour l'ordre point à point.

4.1.1 Quel genre de cpo nous faut-il ?

Au vu des constructions de mini-Caml, on n'aura pas besoin d'un domaine Val qui contienne $[Val \to Val]$ (pour les fonctions), mais quelque chose ressemblant à $[Mem \times Val \to Mem \times Val]$: les fonctions mini-Caml démarrent elles-mêmes dans une certaine mémoire, et retournent une nouvelle mémoire. Ce ne sera pas suffisant. D'abord, il se peut qu'une telle fonction boucle, et ne retourne jamais. Pour représenter ce comportement, on va considérer qu'une fonction mini-Caml n'a pas pour valeur une fonction continue de $Mem \times Val$ vers $Mem \times Val$, mais vers $(Mem \times Val)_\perp$, où $A_\perp$ dénote le cpo A avec un nouvel élément $\perp$ ajouté en-dessous de tous les éléments de A :

Définition 4 (Relèvement) *Pour tout cpo A, soit $A_\perp$ le cpo union disjointe de A et de $\{\perp\}$, avec $x \leq y$ si et seulement si $x = \perp$ ou $x, y \in A$ et $x \leq y$ dans A.*

Vu sous l'angle mathématique, $[Mem \times Val \to Mem \times Val]$ est bien un cpo, mais il n'est en général pas pointé. Or le corollaire 3 demande à ce que l'on se place dans un cpo pointé pour garantir l'existence de points fixes, et nous aurons besoin de tels points fixes pour définir la sémantique de la construction `letrec` $f =$ `fun` $x \to M;;$. Le cpo $[Mem \times Val \to (Mem \times Val)_\perp]$, lui, est pointé, et nous l'utiliserons comme domaine sémantique des fonctions mini-Caml.

▸ **EXERCICE 4.2**
Montrer que $[Mem \times Val \to (Mem \times Val)_\perp]$ a un plus petit élément. Lequel est-ce ?

▸ **EXERCICE 4.3**
Soit A un sous-cpo de $\mathbb{P}\omega$, c'est-à-dire un sous-ensemble de $\mathbb{P}\omega$ qui, muni de l'ordre ($\subseteq$) de $\mathbb{P}\omega$, est un cpo. Montrer que l'ensemble formé, d'une part, des parties de la forme $\{0\} \cup \{n+1 | n \in x\}$ avec $x \in A$, d'autre part de l'ensemble vide, est un sous-cpo de $\mathbb{P}\omega$ isomorphe à $A_\perp$.

▸ **EXERCICE 4.4**
Soient A et B deux sous-cpo de $\mathbb{P}\omega$. Montrer que $i([A \to B])$, l'image par i du cpo $[A \to B]$, est un sous-cpo de $\mathbb{P}\omega$ isomorphe à $[A \to B]$; i et r ont été introduites en section 3.2.1. Indication : surtout n'utilisez pas la définition de i et de r, juste leurs propriétés de continuité et le fait que $r \circ i$ est l'identité.

Ensuite, mini-Caml permet de raisonner sur des références. Intuitivement, la valeur d'une référence sera une adresse, dans $Addr$. Nous aimerions donc pouvoir considérer que Val contient $Addr$, mais Val est un cpo et $Addr$ un ensemble sans structure d'ordre particulière. On utilise alors l'astuce suivante :

Lemme 10 *Pour tout ensemble X, X muni de l'égalité comme ordre partiel est un cpo. On dit qu'un tel cpo est* plat.

Autrement dit, $Addr$ vu comme cpo plat est un cpo dont tous les éléments sont deux à deux incomparables. Ceci correspond bien à l'intuition de Scott que les éléments d'un cpo sont des valeurs partielles : dans un cpo plat, toute valeur est totale.

▸ **EXERCICE 4.5**
Rappelons que $Addr$ est fini ou dénombrable. Montrer que $Addr$ est isomorphe à un sous-cpo de $\mathbb{P}\omega$. Montrer que $\mathbb{B} = \{0, 1\}$, $\mathbb{N}$, $\mathbb{Z}$ sont isomorphes à des sous-cpo de $\mathbb{P}\omega$.

Mini-Caml permet aussi de raisonner sur des n-uplets. On a donc besoin d'une notion de produit cartésien de cpo :

Lemme 11 *Pour tous cpo A_1, …, A_n, le* produit $A_1 \times \ldots \times A_n$ *est l'ensemble des n-uplets $(a_1, \ldots, a_n)$, $a_1 \in A_1$, …, $a_n \in A_n$, avec l'ordre composante par composante :*

$$(a_1, \ldots, a_n) \leq (a'_1, \ldots, a'_n) \textit{ ssi } a_1 \leq a'_1 \textit{ et } \ldots \textit{ et } a_n \leq a'_n$$

Il s'agit d'un cpo. De plus, les projections $\pi_i : A_1 \times \ldots \times A_n \to A_i$ sont continues. Enfin, si $f_1 : C \to A_1$, …, $f_n : C \to A_n$ sont des fonctions continues, alors la fonction qui à $c \in C$ associe $(f_1(c), \ldots, f_n(c))$ est continue.

▸ **EXERCICE 4.6**
Démontrer le lemme 11.

▸ **EXERCICE 4.7**
Soient A_1, …, A_n des sous-cpo de $\mathbb{P}\omega$. Montrer que $\{\langle i, k\rangle | 1 \leq i \leq n, k \in a_i\}$ est un sous-cpo de $\mathbb{P}\omega$ isomorphe à $A_1 \times \ldots \times A_n$.

De même que l'on dispose des produits, on a aussi les sommes, c'est-à-dire l'équivalent des unions *disjointes* :

Lemme 12 *Soit $(A_i)_{i \in I}$ une famille de cpo. Leur* somme*, ou* coproduit $\sum_{i \in I} A_i$ *est l'ensemble des couples (i, a) où $i \in I$ et $a \in A_i$, avec l'ordre somme défini par :*

$$(i, a) \leq (i', a') \textit{ ssi } i = i' \textit{ et } a \leq a' \textit{ dans } A_i$$

Il s'agit d'un cpo. De plus, les injections $\iota_i : A_i \to \sum_{j \in I} A_j$ sont continues. Enfin, si $f_i : A_i \to C$ sont une famille de fonctions continues de A_i dans un cpo C, alors la fonction de $\sum_{i \in I} A_i$ dans C qui à (i, a) associe $f_i(a)$ est continue.

▸ **EXERCICE 4.8**
Démontrer le lemme 12.

▸ **EXERCICE 4.9**
Soit I un ensemble d'entiers. Soit $(A_i)_{i \in I}$ une famille de sous-cpo de $\mathbb{P}\omega$. Montrer que l'ensemble des $\{\langle i, [e]\rangle | e \text{ fini } \subseteq a\}$, $i \in I$, $a \in A_i$, est un sous-cpo de $\mathbb{P}\omega$ isomorphe à $\sum_{i \in I} A_i$.

▸ **EXERCICE 4.10**
Généraliser le lemme 11 et l'exercice 4.7 aux cas de produits infinis de cpo indexés par I. En quoi ceci diffère-t-il (ou non) d'un espace de fonctions continues de I vers un certain cpo ?

Avec le produit et la somme, on peut définir le cpo Val^* des *listes* (ou *mots*, ou *suites*) d'éléments de Val :

$$Val = \sum_{i \in \mathbb{N}} \underbrace{Val \times \ldots \times Val}_{i \text{ fois}}$$

On peut aussi définir les sommes finies $A_1 + \ldots + A_n$, qui ne sont autres que $\sum_{i \in I} A_i$, avec $I = \{1, \ldots, n\}$.

Nous avons besoin d'un domaine Val qui contienne :
– $Mem \times Val \to (Mem \times Val)_\perp$ pour représenter les fonctions ;

- $Addr$ pour les références ;
- Val^* pour les n-uplets ;
- $\mathbb{Z}$ pour les entiers ;
- $\mathbb{B}$ pour les booléens.

De plus, on souhaite pouvoir faire la différence entre fonctions, références, n-uplets, etc. Nous serons donc intéressé par la somme de tous ces cpo.

Dans les sections qui suivent, nous supposerons donc que Val est un cpo tel que, à isomorphisme près :

$$\begin{aligned} Val \supseteq\ & [Mem \times Val \to (Mem \times Val)_\perp] \\ & +Addr \\ & +Val^* \\ & +\mathbb{Z} \\ & +\mathbb{B} \end{aligned} \tag{2}$$

▸ **EXERCICE 4.11**

En utilisant les exercices 4.3, 4.4, 4.5, 4.7, 4.9, montrer que $Val = \mathbb{P}\omega$ est une solution à l'inégalité ci-dessus.

4.1.2 Sémantique dénotationnelle des expressions

Le problème du domaine des valeurs Val étant réglé, passons à la sémantique des expressions. Nous allons définir la quantité $[\![M]\!]_{\text{caml}}\,\rho\mu \in (Mem \times Val)_\perp$ par *récurrence structurelle* sur l'expression M, c'est-à-dire que $[\![M]\!]_{\text{caml}}\,\rho\mu$ sera définie en fonction de données $[\![N]\!]_{\text{caml}}\,\rho\mu$, où N est une sous-expression directe de M.

Les esprits attentifs auront au passage remarqué que le résultat de $[\![M]\!]_{\text{caml}}\,\rho\mu$ n'est finalement pas dans $Mem \times Val$, mais dans $(Mem \times Val)_\perp$: il est en effet possible que, en présence de `letrec`, l'évaluation de M ne termine pas.

Ceci étant dit, la sémantique des variables est immédiate :

$$[\![x]\!]_{\text{caml}}\,\rho\mu \quad = \quad (\mu, \rho(x)) \tag{3}$$

Une variable a en effet pour valeur ce que l'environnement dit, et la mémoire est inchangée.

Pour la sémantique des fonctions et de leurs applications, on utilise le fait que (à isomorphisme près), $[Mem \times Val \to (Mem \times Val)_\perp]$ est un sous-cpo de Val :

$$\begin{aligned} [\![MN]\!]_{\text{caml}}\,\rho\mu \ &=_{def}\ \phi([\![N]\!]_{\text{caml}}\,\rho\mu') \quad \text{si } \phi \in [Mem \times Val \to (Mem \times Val)_\perp], \\ &\qquad\qquad \text{où } (\mu', \phi) =_{def} [\![M]\!]_{\text{caml}}\,\rho\mu \neq \perp \\ &\qquad\qquad \text{et } [\![N]\!]_{\text{caml}}\,\rho\mu' \neq \perp \\ &=_{def}\ \perp \qquad \text{sinon} \end{aligned} \tag{4}$$

$$\begin{aligned} [\![\texttt{fun}\ x \to M]\!]_{\text{caml}}\,\rho\mu \ &=_{def}\ (\mu, \phi) \\ &\qquad \text{où } \phi \text{ est défine par } \phi(\mu'', V) = [\![M]\!]_{\text{caml}}\,(\rho[x \mapsto V])\mu'' \end{aligned} \tag{5}$$

En d'autres termes, pour calculer la valeur de MN dans l'environnement ρ, et en partant de la mémoire M, on calcule la valeur $[\![M]\!]_{\text{caml}}\,\rho\mu$ de M. Ceci peut valoir $\perp$ (pas moyen d'obtenir la valeur de M en tant que fonction), auquel cas la valeur de MN est $\perp$ de nouveau. Ou bien ceci vaut un couple (μ', ϕ) formé d'une nouvelle mémoire après l'évaluation de M, et d'une valeur ϕ. Si ϕ n'est pas une fonction, c'est-à-dire pas un élément de $[Mem \times Val \to (Mem \times Val)_\perp]$, encore une fois la valeur de ϕ est $\perp$ (l'indéfini). Sinon, on applique la fonction ϕ à la valeur de N, si pas indéfinie. La sémantique de $\texttt{fun}\ x \to M$ devrait être plus déchiffrable. Notez cependant que la mémoire μ ne change pas lors de l'exécution de $\texttt{fun}\ x \to M$: le couple retourné est (μ, ϕ), avec le même μ qu'en paramètre. Mais la valeur ϕ elle-même est une fonction qui prendra l'état courant de la mémoire au moment où on l'appliquera, et pourra retourner une nouvelle mémoire.

▸ **EXERCICE 4.12**
Intuitivement, dans quel ordre la définition de $[\![MN]\!]_{\text{caml}}\,\rho\mu$ force-t-elle l'évaluation de MN ?

▸ **EXERCICE 4.13**
Montrer que les côtés droits des équations 4 et 5 sont bien définis. Notez en particulier que les injections canoniques sont toutes continues. Vérifiez soigneusement en particulier que la fonction ϕ de l'équation 5 est bien continue, en supposant que $[\![M]\!]_{\text{caml}}\,\rho'\mu'$ est une fonction continue en $\rho' \in Env$ pour tout $\mu' \in Mem$.

La construction suivante de mini-Caml est le `let` :

$$\begin{aligned}
[\![\texttt{let}\ x = M\ \texttt{in}\ N]\!]_{\text{caml}}\,\rho\mu &= [\![N]\!]_{\text{caml}}\,(\rho[x \mapsto V])\mu' \\
&\quad\ \text{si } (\mu', V) = [\![M]\!]_{\text{caml}}\,\rho\mu \neq \perp \\
&= \perp \text{ sinon}
\end{aligned} \tag{6}$$

▸ **EXERCICE 4.14**
Les constructions $\texttt{let}\ x = M\ \texttt{in}\ N$ et $(\texttt{fun}\ x \to N)M$ sont-elles équivalentes ? On dira que deux expressions M et N sont *équivalentes* si et seulement si $[\![M]\!]_{\text{caml}}\,\rho\mu = [\![N]\!]_{\text{caml}}\,\rho\mu$ pour tout $\rho \in Env$ et $\mu \in Mem$.

Abordons maintenant les références, ce pour quoi nous avons introduit les mémoires. Pour ceci :

Définition 5 (Stratégie d'allocation mémoire) *Une* stratégie d'allocation mémoire *est une fonction* $alloc$ *de l'ensemble* $\mathbb{P}_{fin}(Addr)$ *des ensembles finis d'adresses vers* $Addr_\perp$*, telle que* $alloc(S) \notin S$ *pour toute partie finie* S *de* $Addr$*.*

Fixons-nous dans la suite une stratégie d'allocation mémoire $alloc$.

En notant ϵ le n-uplet vide (dans $Val^* \subseteq Val$) :

$$\begin{aligned}
[\![\texttt{ref}\ M]\!]_{\text{caml}}\,\rho\mu &= (\mu'[a \mapsto V], a) \\
&\quad\ \text{où } (\mu', V) = [\![M]\!]_{\text{caml}}\,\rho\mu \neq \perp, a = alloc(\text{dom}\ \mu') \neq \perp \\
&= \perp \qquad \text{sinon}
\end{aligned} \tag{7}$$

$$[\![!M]\!]_{\text{caml}}\,\rho\mu = (\mu', \mu'(a)) \text{ si } a \in \text{dom}\ \mu',\ \text{où } (\mu', a) = [\![M]\!]_{\text{caml}}\,\rho\mu \neq \perp \tag{8}$$

$$
\begin{aligned}
& = \perp \quad \text{sinon} \\
[\![M{:=}N]\!]_{\text{caml}}\,\rho\mu & = (\mu''[a \mapsto V], \epsilon) \text{ si } a \in \text{dom } \mu'' \qquad (9) \\
& \text{ où } (\mu', a) = [\![M]\!]_{\text{caml}}\,\rho\mu \neq \perp,\ (\mu'', V) = [\![N]\!]_{\text{caml}}\,\rho\mu' \neq \perp \\
& = \perp \text{ sinon}
\end{aligned}
$$

L'effet de `ref` M est donc de calculer M, puis d'allouer une adresse non encore utilisée a, c'est-à-dire hors du domaine de le mémoire courante μ' (si c'est possible, c'est-à-dire si la stratégie retourne autre chose que $\perp$), et met à jour la mémoire de sorte que l'on trouve la valeur V de M à l'adresse a.

Les opérations sur les n-uplets sont maintenant faciles. Notons $V_1 \cdot \ldots \cdot V_n$ le n-uplet (le mot) formé des composantes V_1, ..., V_n. Nous n'utiliserons pas la notation $(V_1, \ldots, V_n)$, qui pourrait prêter à confusion avec la syntaxe $(M_1, \ldots, M_n)$ des n-uplets dans le langage mini-Caml.

$$
\begin{aligned}
[\![(M_1, \ldots, M_n)]\!]_{\text{caml}}\,\rho\mu & = (\mu_n, V_1 \cdot \ldots \cdot V_n) \qquad (10) \\
& \text{ où } \mu_0 = \mu, \text{ et pour tout } i,\ 1 \leq i \leq n, \\
& \qquad (\mu_i, V_i) = [\![M_i]\!]_{\text{caml}}\,\rho\mu_{i-1} \neq \perp \\
& = \perp \quad \text{sinon} \\
[\![\mathtt{proj}_i M]\!]_{\text{caml}}\,\rho\mu & = (\mu', V_i) \qquad (11) \\
& \text{ si } (\mu', V_1 \cdot \ldots \cdot V_i \cdot \ldots) = [\![M]\!]_{\text{caml}}\,\rho\mu \neq \perp \\
& = \perp \quad \text{sinon}
\end{aligned}
$$

Rappelons que $\mathbb{B} = \{0, 1\}$. La sémantique des autres constructions essentielles de mini-Caml, la séquence et la conditionnelle, est immédiate elle aussi :

$$
\begin{aligned}
[\![M; N]\!]_{\text{caml}}\,\rho\mu & = [\![N]\!]_{\text{caml}}\,\rho\mu' \qquad (12) \\
& \text{ où } (\mu', V) = [\![M]\!]_{\text{caml}}\,\rho\mu \neq \perp \\
& = \perp \quad \text{sinon} \\
[\![\mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ P]\!]_{\text{caml}}\,\rho\mu & = [\![N]\!]_{\text{caml}}\,\rho\mu' \qquad (13) \\
& \text{ où } (\mu', 1) = [\![M]\!]_{\text{caml}}\,\rho\mu \neq \perp \\
& = [\![P]\!]_{\text{caml}}\,\rho\mu'' \\
& \text{ où } (\mu'', 0) = [\![M]\!]_{\text{caml}}\,\rho\mu \neq \perp \\
& =_{def} \perp \quad \text{sinon}
\end{aligned}
$$

Nous ne donnerons pas la sémantique des autres instructions, qui se définissent au cas par cas, or nous n'avons pas défini précisément de quelles autres instructions nous disposons en mini-Caml (voir les trois petits points dans la ligne des opérations arithmétiques en figure 4). Les constructions essentielles ont été vues ci-dessus, de toute façon.

En supposant que toutes les constructions de mini-Caml ont été décrites (c'est-à-dire en ignorant les constructions arithmétiques, pas souci de simplicité), on peut montrer :

Théorème 13 (Bonne définition, continuité) *Les équations (3)–(13) sont une définition valide de la fonction* $[\![\_]\!]_{\text{caml}} : \mathbf{Caml} \times Env \times Mem \to (Mem \times Val)_\perp$. *De plus,* $[\![M]\!]_{\text{caml}}\,\rho\mu$ *est continue en* $\rho \in Env$ *et* $\mu \in Mem$.

Démonstration. Les deux résultats, bonne définition et continuité, se démontrent simultanément, par récurrence structurelle sur M. À noter qu'on a besoin de la continuité de $[\![M]\!]_{\text{caml}}$ ne serait-ce que pour montrer que le côté droit de l'équation (5) définissant la sémantique de $\texttt{fun}\ x \to M$ est bien défini, sans parler de continuité. □

▸ **EXERCICE 4.15**
Montrer qu'une fonction f de $X \times Y$ dans Z, où X, Y, Z sont des cpo, est continue si et seulement si elle est continue en chaque argument séparément. Autrement dit, f est continue de $X \times Y$ vers Z si et seulement si, pour tout $x \in X$, la fonction $y \mapsto f(x, y)$ est continue de Y vers Z, et pour tout $y \in Y$, la fonction $x \mapsto f(x, y)$ est continue de X vers Z. (On rappelle que ce n'est pas le cas pour les fonctions d'un produit de deux espaces topologiques quelconques dans un autre.)

▸ **EXERCICE 4.16**
Démontrez formellement le théorème 13. On pourra s'aider de l'exercice 4.15 pour le cas de l'équation (5).

4.1.3 Sémantique dénotationnelle des programmes

Nous avons donné la sémantique des expressions, il reste à donner celle des programmes. Les programmes mini-Caml sont juste des listes de définitions, récursives ($\texttt{letrec}\ f = \texttt{fun}\ x \to M;;$) ou non ($\texttt{let}\ x = M;;$).

La sémantique d'une telle définition d sera une donnée $[\![d]\!]^{\text{prog}}_{\text{caml}}\,\rho\mu$ dans $(Env \times Mem)_\perp$. En effet, soit la définition échoue (n'a aucun sens), et $[\![d]\!]^{\text{prog}}_{\text{caml}}\,\rho\mu$ vaudra $\perp$, par exemple, si M ne termine pas dans $\texttt{let}\ x = M;;$, soit la définition retourne un nouvel environnement ρ, enrichi de la définition du symbole f ou x, ainsi qu'une nouvelle mémoire au cas où le calcul l'aurait modifiée.

Posons $Fun = [Mem \times Val \to (Mem \times Val)_\perp]$. On rappelle que Fun est un cpo pointé (exercice 4.2). Par le corollaire 3, toute fonction continue F de Fun dans Fun a un plus petit point fixe, notons-le $lfp(F)$ ("least fixed point"). La définition suivante est donc légitime :

$$[\![\texttt{letrec}\ f = \texttt{fun}\ x \to M;;]\!]^{\text{prog}}_{\text{caml}}\,\rho\mu \quad = \quad (\rho[f \mapsto lfp(F)], \mu) \tag{14}$$

où $F : Fun \to Fun$ est la fonction qui à $\phi \in Fun$ associe l'unique ϕ' tel que $(\mu, \phi') = [\![\texttt{fun}\ x \to M;;]\!]_{\text{caml}}\,(\rho[f \mapsto \phi])\mu$: ϕ' est par construction un élément de Fun. De plus, F est continue par l'exercice 4.16.

Le cas du $\texttt{let}$ est plus simple, mais permet de changer la mémoire :

$$\begin{aligned} [\![\texttt{let}\ x = M;;]\!]^{\text{prog}}_{\text{caml}}\,\rho\mu \quad &= \quad (\rho[x \mapsto V], \mu') \\ & \qquad \text{où } (\mu', V) = [\![M]\!]_{\text{caml}}\,\rho\mu \neq \perp \\ &= \quad \perp \quad \text{sinon} \end{aligned} \tag{15}$$

Finalement, un programme étant une séquence de telles définitions, on pose :

$$[\![\epsilon]\!]^{\text{prog}}_{\text{caml}}\,\rho\mu \quad = \quad (\rho, \mu) \tag{16}$$

$$
\begin{aligned}
[\![d_1\ d_2]\!]^{\mathrm{prog}}_{\mathrm{caml}}\,\rho\mu &= [\![d_2]\!]_{\mathrm{caml}}\,\rho'\mu' \\
&\quad \text{où } (\rho', \mu') = [\![d_1]\!]_{\mathrm{caml}}\,\rho\mu \neq \bot \\
&= \bot \qquad \text{sinon}
\end{aligned}
\tag{17}
$$