Introduction à la sémantique des langages de programmation

Adrien Guatto

Compilation M1

Ce cours et les deux suivants

Quoi ?

- Mathématiser des concepts essentiels des langages de programmation.
- Implémenter ces concepts en OCaml, le cas échéant.

Pourquoi ?

- Acquérir les notions et la terminologie de base du domaine.
- Comprendre les sujets des prochains jalons du projet.
- Se préparer à vos futurs cours de sémantique (par exemple, au S2).

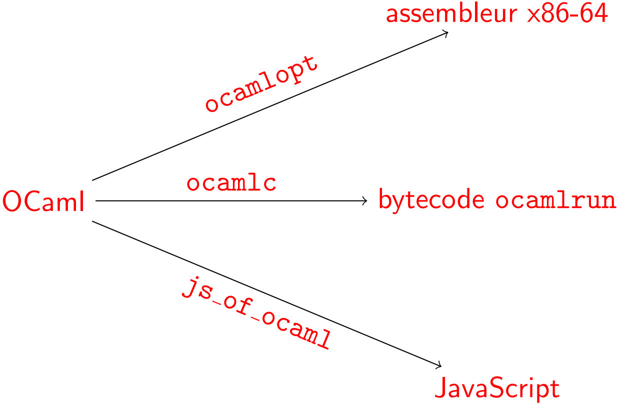
Quel rôle pour Marthe ?

- Beaucoup plus simple qu'OCaml, Java, Python, ou le λ-calcul.
- Exhibe pourtant certaines difficultés caractéristiques.
- Peut être progressivement étendu en un langage plus riche.

Cours 1

Syntaxe et sémantique de Marthe

Un compilateur traduit un langage source vers un langage cible.



Un compilateur doit avant tout être correct. Qu'est-ce que cela signifie ?

Correction des compilateurs, dans l'abstrait

Quelques notations :

- $|\mathcal{S}|$ désigne l'ensemble des *termes* du langage source,
- $|\mathcal{T}|$ désigne l'ensemble des termes du langage cible,
- $C : |\mathcal{S}| \to |\mathcal{T}|$ désigne le compilateur, fonction de $|\mathcal{S}|$ dans $|\mathcal{T}|$.

Intuitivement, on aimerait adopter une définition ressemblant à :

$$C \text{ est correct} \stackrel{\text{def}}{=} \forall M \in |\mathcal{S}|, M \text{ et } C(M) \text{ “font la même chose”}.$$

C'est un peu vague. Essayons :

$$C \text{ est correct} \stackrel{\text{def}}{=} \forall M \in |\mathcal{S}|, M \text{ et } C(M) \text{ ont le même résultat}.$$

Chaque langage $L \in \{\mathcal{S}, \mathcal{T}\}$ doit donc définir le résultat $R_L(M)$ de tout terme $M \in |L|$. On obtient donc, formellement :

$$C \text{ est correct} \stackrel{\text{def}}{=} \forall M \in |\mathcal{S}|, R_{\mathcal{S}}(M) = R_{\mathcal{T}}(C(M)).$$

La sémantique de Marthe (1/3)

Comment définir mathématiquement le langage Marthe, noté $\mathcal{M}$?

- $|\mathcal{M}|$ est l'ensemble des arbres de syntaxe abstraite de Marthe, décrits sous la forme d'une grammaire BNF au premier cours.

$$|\mathcal{M}| \ni M, N, P ::= x \mid \underline{n} \mid M \mathbin{\underline{+}} N \mid M \mathbin{\underline{*}} N \mid \underline{\sum}_{x=M}^{N} P$$

- Pour définir $R_{\mathcal{M}}$, on va reformuler notre interprète écrit en OCaml sous la forme d'un ensemble de triplets appelé *Eval*.

$$(M, \sigma, n) \in \mathit{Eval} \Leftrightarrow \text{“}M \text{ s'évalue en } n \text{ dans l'environnement } \sigma\text{”}$$

- En OCaml, un environnement est une liste de couples variable/entier. Et en sémantique ? Un choix commode : les fonctions partielles finies.

$$\mathit{Env} \stackrel{\text{def}}{=} \{\sigma : \mathit{Var} \rightharpoonup \mathbb{N} \mid \sigma^{-1}(\mathbb{N}) \text{ est fini}\}$$

- Le résultat d'un terme Marthe est l'ensemble des entiers vers lesquels il s'évalue dans l'environnement vide :

$$R_{\mathcal{M}}(M) \stackrel{\text{def}}{=} \{n \in \mathbb{N} \mid (M, \emptyset, n) \in \mathit{Eval}\}.$$

La sémantique de Marthe (2/3)

Que doit contenir notre ensemble de triplets $\mathit{Eval} \subseteq |\mathcal{M}| \times \mathit{Env} \times \mathbb{N}$?

- Le terme $\underline{n}$ doit s'évaluer en n dans tout σ.

$$\{(\underline{n}, \sigma, n) \mid n \in \mathbb{N}, \sigma \in \mathit{Env}\} \subseteq \mathit{Eval}$$

- Le terme x doit s'évaluer en v dans σ **si** $\sigma(x) = v$.

$$\{(x, \sigma, \sigma(x)) \mid x \in \mathit{Var}, \sigma \in \mathit{Env}\} \subseteq \mathit{Eval}$$

- Le terme $M \mathbin{\underline{+}} N$ doit s'évaluer en $m + n$ dans σ **si** M s'évalue en m dans σ et N s'évalue en n dans σ. De même pour $\underline{*}$.

$$\{(M \mathbin{\underline{+}} N, \sigma, m + n) \mid (M, \sigma, m), (N, \sigma, n) \in \mathit{Eval}\} \subseteq \mathit{Eval}$$
$$\{(M \mathbin{\underline{*}} N, \sigma, mn) \mid (M, \sigma, m), (N, \sigma, n) \in \mathit{Eval}\} \subseteq \mathit{Eval}$$

- **[Exercice]** Déterminer les contraintes pour $\underline{\sum}_{x=M}^{N} P$.

La sémantique de Marthe (3/3)

Au moins deux approches possibles pour $\sum_{x=M}^{N} P$.

1. Un seul cas : évaluer P sur la plage complète en une seule fois.

$$\left\{ \begin{pmatrix} \sum_{x=M}^{N} P, \\ \sigma, \sum_{m \leq i \leq n} p_i \end{pmatrix} \middle| \begin{array}{r} (M, \sigma, m), (N, \sigma, n) \in Eval, \\ \forall m \leq i \leq n, (P, \sigma[x \mapsto i], p_i) \in Eval \end{array} \right\} \subseteq Eval$$

2. Deux cas : évaluer P itération par itération.

$$\left\{ \left(\sum_{x=M}^{N} P, \sigma, 0\right) \middle| (M, \sigma, m), (N, \sigma, n) \in Eval, n < m \right\} \subseteq Eval$$

$$\left\{ \left(\sum_{x=M}^{N} P, \sigma, p + r\right) \middle| \begin{array}{r} (M, \sigma, m), (N, \sigma, n), (P, \sigma[x \mapsto m], p), \\ (\sum_{x=M+1}^{N} P, \sigma, r) \in Eval, m \leq n \end{array} \right\} \subseteq Eval$$

Les deux approches sont *mathématiquement* équivalentes.

Fabriquer la sémantique (1/2)

On a obtenu un ensemble de contraintes sur notre ensemble *Eval*.

$$\{(\underline{n}, \sigma, n) \mid n \in \mathbb{N}, \sigma \in Env\} \subseteq Eval \qquad (1)$$

$$\{(x, \sigma, \sigma(x)) \mid x \in Var, \sigma \in Env\} \subseteq Eval \qquad (2)$$

$$\{(M \mathbin{\underline{+}} N, \sigma, m + n) \mid (M, \sigma, m), (N, \sigma, n) \in Eval\} \subseteq Eval \qquad (3)$$

$$\{(M \mathbin{\underline{*}} N, \sigma, mn) \mid (M, \sigma, m), (N, \sigma, n) \in Eval\} \subseteq Eval \qquad (4)$$

$$\left\{ \left(\underline{\sum}_{x=M}^{N} P, \sigma, 0 \right) \middle| (M, \sigma, m), (N, \sigma, n) \in Eval, n < m \right\} \subseteq Eval \qquad (5)$$

$$\left\{ \begin{pmatrix} \underline{\sum}_{x=M}^{N} P, \\ \sigma, p + r \end{pmatrix} \middle| \begin{array}{r} (M, \sigma, m), (N, \sigma, n), (P, \sigma[x \mapsto m], p), \\ (\underline{\sum}_{x=M \mathbin{\underline{+}} \underline{1}}^{N} P, \sigma, r) \in Eval, m \leq n \end{array} \right\} \subseteq Eval \qquad (6)$$

Comment construire un ensemble *Eval* les respectant *exactement* ?

- En appliquant le théorème de Knaster-Tarski.
- En le construisant à la main, cf. transparent suivant.

Fabriquer la sémantique (2/2)

L'idée : construire une séquence croissante d'ensembles $Eval_k$ pour $k \in \mathbb{N}$.

$$
\begin{aligned}
Eval_0 &= \{(\underline{n}, \sigma, n) \mid n \in \mathbb{N}, \sigma \in Env\} \cup \{(x, \sigma, \sigma(x)) \mid x \in Var, \sigma \in Env\} \\
Eval_1 &= Eval_0 \cup \{(M \mathbin{\underline{+}} N, \sigma, m + n) \mid (M, \sigma, m), (N, \sigma, n) \in Eval_0\} \\
&\quad \cup \{(M \mathbin{\underline{*}} N, \sigma, mn) \mid (M, \sigma, m), (N, \sigma, n) \in Eval_0\} \\
&\quad \cup \left\{ \left(\underline{\textstyle\sum}_{x=M}^{N} P, \sigma, 0 \right) \middle| (M, \sigma, m), (N, \sigma, n) \in Eval_0, n < m \right\} \\
&\quad \cup \left\{ \begin{pmatrix} \underline{\sum}_{x=M}^{N} P, \\ \sigma, p + r \end{pmatrix} \middle| \begin{array}{c} (M, \sigma, m), (N, \sigma, n), (P, \sigma[x \mapsto m], p), \\ (\underline{\sum}_{x=M \mathbin{\underline{+}} \underline{1}}^{N} P, \sigma, r) \in Eval_0, m \leq n \end{array} \right\} \\
Eval_{k+1} &= Eval_k \cup \{(M \mathbin{\underline{+}} N, \sigma, m + n) \mid (M, \sigma, m), (N, \sigma, n) \in Eval_k\} \cup \ldots
\end{aligned}
$$

La séquence $(Eval_k)_{k \in \mathbb{N}}$ converge vers le $Eval$ qu'on cherche à construire. Il suffit donc de poser comme définition :

$$Eval \stackrel{\text{def}}{=} \bigcup_{k \in \mathbb{N}} Eval_k.$$

Présentation alternative de la sémantique

Il est commode d'adopter une notation infixe pour *Eval*.

$$\boxed{M; \sigma \Downarrow n} \stackrel{\text{def}}{=} (M, \sigma, n) \in \mathit{Eval} \Leftrightarrow \exists k \in \mathbb{N}, (M, \sigma, n) \in \mathit{Eval}_k$$

Ensuite, on peut montrer que $M; \sigma \Downarrow n$ est vrai si et seulement c'est la racine d'un arbre dont les noeuds sont étiquettés par les règles ci-dessous.

$$\frac{}{\underline{n}; \sigma \Downarrow n} \qquad \frac{}{x; \sigma \Downarrow \sigma(x)} \qquad \frac{M; \sigma \Downarrow m \qquad N; \sigma \Downarrow n}{M \mathbin{\underline{+}} N; \sigma \Downarrow m + n}$$

$$\frac{M; \sigma \Downarrow m \qquad N; \sigma \Downarrow n}{M \mathbin{\underline{*}} N; \sigma \Downarrow mn} \qquad \frac{M; \sigma \Downarrow m \qquad N; \sigma \Downarrow n}{\underline{\Sigma}_{x=M}^{N} P; \sigma \Downarrow 0}\ n < m$$

$$\frac{M; \sigma \Downarrow m \qquad N; \sigma \Downarrow n \qquad P; \sigma[x \mapsto m] \Downarrow p \qquad \underline{\Sigma}_{x=M \mathbin{\underline{+}} \underline{1}}^{N} P; \sigma \Downarrow r}{\underline{\Sigma}_{x=M}^{N} P; \sigma \Downarrow p + r}\ m \leq n$$

[Exercice] Construire un arbre de racine $\underline{\Sigma}_{x=\underline{1}}^{2} x \mathbin{\underline{*}} \underline{3}; \emptyset \Downarrow n$ (n au choix).

Jugements inductifs et sémantiques à grands pas

En pratique

Plutôt que de construire $(Eval_k)_{k \in \mathbb{N}}$ et $Eval$ sous forme ensembliste, on préfère définir directement la relation $M; \sigma \Downarrow n$ par des règles.

- On appelle une telle relation un *jugement inductif* et un arbre de racine $M; \sigma \Downarrow N$ une *dérivation* de *conclusion* $M; \sigma \Downarrow N$.
- Les deux définitions sont équivalentes car $(M, \sigma, n) \in Eval_k$ si et seulement si $M; \sigma \Downarrow n$ admet une dérivation de hauteur k.

- Une sémantique qui associe à un terme son résultat final est dite **"à grands pas"**. On en verra d'autres exemples.
- Celle que nous venons de définir est **déterministe**.

$$\text{Si } M; \sigma \Downarrow n \text{ et } M; \sigma \Downarrow m \text{ alors } m = n.$$

- **[Exercice]** Est-elle **totale** ?

$$\forall M \in |\mathcal{M}|, \forall \sigma \in Env, \exists n \in \mathbb{N}, M; \sigma \Downarrow n$$

Quand l'évaluation est-elle définie ?

- Il n'existe pas d'entier n tel que que $\mathtt{x}; \emptyset \Downarrow n$. Donc $R_{\mathcal{M}}(x) = \emptyset$.
- Notre sémantique définit donc une fonction partielle de $|\mathcal{M}| \times \mathit{Env}$ dans $\mathbb{N}$, que notre fonction OCaml `eval` implémente.
- **[Exercice]** Quels sont les termes M tels que $R_{\mathcal{M}}(M) = \emptyset$?
- Pour que l'évaluation d'un terme soit définie, il faut que la valeur de chacune de ses variables x soit définie. Donc, x doit :
  - soit appartenir à $\sigma^{-1}(\mathbb{N})$,
  - soit apparaître sous une construction $\underline{\Sigma}_{x=M}^{N}(-)$ (qui la "lie").

  Comment rendre ces intuitions précises ?

Variables libres et termes clos

- L'ensemble $FV(M)$ des *variables libres* d'un terme M est défini par récursion sur M.

$$
\begin{aligned}
FV(\underline{n}) &= \emptyset \\
FV(x) &= \{x\} \\
FV(M \mathbin{\underline{+}} N) &= FV(M) \cup FV(N) \\
FV(M \mathbin{\underline{*}} N) &= FV(M) \cup FV(N) \\
FV(\underline{\sum}_{x=M}^{N} P) &= FV(M) \cup FV(N) \cup (FV(P) \setminus \{x\})
\end{aligned}
$$

- Un terme M est dit *clos* si $FV(M) = \emptyset$ et *ouvert* sinon.
- **[Exercice]** $\underline{\sum}_{\mathrm{x}=\underline{1}}^{\underline{4}} \mathrm{x}$, $\underline{\sum}_{\mathrm{x}=\underline{1}}^{\underline{4}} \mathrm{y}$ et $(\underline{\sum}_{\mathrm{y}=\underline{1}}^{\underline{4}} \mathrm{y}) \mathbin{\underline{+}} \mathrm{y}$ sont-ils clos ou ouverts ?
- **[Exercice]** Programmer FV en OCaml.

Propriété

Il existe $n \in \mathbb{N}$ tel que $M; \sigma \Downarrow n$ si et seulement si $FV(M) \subseteq \sigma^{-1}(\mathbb{N})$.

Équivalence observationnelle de termes Marthe

Il semble raisonnable de considérer que deux termes Marthe sont équivalents lorsqu'ils calculent le même entier dans tout environnement.

$$M \equiv N \stackrel{\text{def}}{=} \forall \sigma \in \mathit{Env}, \forall n \in \mathbb{N},\ M; \sigma \Downarrow n \Leftrightarrow N; \sigma \Downarrow n$$

Que peut-on dire sur cette relation d'*équivalence observationnelle* ?

- Elle est clairement réflexive, symétrique et transitive.
- Elle valide les identités arithmétiques habituelles.

$$(M \underline{+} N) \underline{+} P \equiv M \underline{+} (N \underline{+} P) \qquad M \underline{+} N \equiv N \underline{+} M \qquad \ldots$$

- Elle valide certains renommages de variables.

$$\underline{\Sigma}_{\mathtt{x}=\underline{1}}^{\underline{9}} (\mathtt{x} \underline{*} \mathtt{z}) \stackrel{?}{\equiv} \underline{\Sigma}_{\mathtt{y}=\underline{1}}^{\underline{9}} (\mathtt{y} \underline{*} \mathtt{k}) \qquad \underline{\Sigma}_{\mathtt{x}=\underline{1}}^{\underline{9}} (\mathtt{x} \underline{*} \mathtt{z}) \stackrel{?}{\equiv} \underline{\Sigma}_{\mathtt{y}=\underline{1}}^{\underline{9}} (\mathtt{y} \underline{*} \mathtt{z})$$

$$\underline{\Sigma}_{\mathtt{x}=\underline{1}}^{\underline{9}} \underline{\Sigma}_{\mathtt{y}=\underline{1}}^{\underline{9}} (\mathtt{x} \underline{+} \mathtt{y}) \stackrel{?}{\equiv} \underline{\Sigma}_{\mathtt{y}=\underline{1}}^{\underline{9}} \underline{\Sigma}_{\mathtt{x}=\underline{1}}^{\underline{9}} (\mathtt{y} \underline{+} \mathtt{x}) \qquad \mathtt{x} \underline{+} \underline{\Sigma}_{\mathtt{x}=\underline{1}}^{\underline{9}} \mathtt{x} \stackrel{?}{\equiv} \mathtt{y} \underline{+} \underline{\Sigma}_{\mathtt{y}=\underline{1}}^{\underline{9}} \mathtt{y}$$

L'égalité à un renommage des variables liées près est une notion universelle dans les langages de programmation : l'α-conversion.

L'α-conversion : intuitions

Une *occurence* liée n'a pas d'identité : elle ne fait que référence à un lieur.

$$\underline{\sum}_{\mathtt{x}=\underline{0}}^{\underline{9}} \left(\mathtt{x} \mathbin{\underline{+}} \underline{\sum}_{\mathtt{x}=\underline{0}}^{\underline{9}} \mathtt{x}\right) \equiv_\alpha \underline{\sum}_{\mathtt{x}=\underline{0}}^{\underline{9}} \left(\mathtt{x} \mathbin{\underline{+}} \underline{\sum}_{\mathtt{y}=\underline{0}}^{\underline{9}} \mathtt{y}\right) \not\equiv_\alpha \underline{\sum}_{\mathtt{x}=\underline{0}}^{\underline{9}} \left(\mathtt{x} \mathbin{\underline{+}} \underline{\sum}_{\mathtt{y}=\underline{0}}^{\underline{9}} \mathtt{x}\right)$$

On peut donc voir les termes comme des graphes de liaison.

$$\underline{\sum}_{\square=\underline{0}}^{\underline{9}} \left(\square \mathbin{\underline{+}} \underline{\sum}_{\square=\underline{0}}^{\underline{9}} \square\right) \neq \underline{\sum}_{\square=\underline{0}}^{\underline{9}} \left(\square \mathbin{\underline{+}} \underline{\sum}_{\square=\underline{0}}^{\underline{9}} \square\right)$$

Les graphes de liaison :

- rendent l'α-conversion triviale ($M \equiv_\alpha N$ ssi $\mathrm{Gr}(M) = \mathrm{Gr}(N)$),
- assignent une identité uniquement aux occurences libres,

$$\mathrm{Gr}\left(\underline{\sum}_{\mathtt{x}=\underline{0}}^{\underline{9}} (\mathtt{x} \mathbin{\underline{+}} \mathtt{y})\right) = \underline{\sum}_{\square=\underline{0}}^{\underline{9}} (\square \mathbin{\underline{+}} \mathtt{y})$$

- sont utilisés en pratique dans les implémentations efficaces.

L'α-conversion : renommage des variabes libres

On définit $M[y/x]$, le renommage de x en y dans M, comme suit.

$$z[y/x] = \begin{cases} y \text{ si } z = x \\ z \text{ sinon} \end{cases}$$

$$\underline{n}[y/x] = \underline{n}$$
$$(M \mathbin{\underline{+}} P)[y/x] = M[y/x] \mathbin{\underline{+}} P[y/x]$$
$$(M \mathbin{\underline{*}} P)[y/x] = M[y/x] \mathbin{\underline{*}} P[y/x]$$
$$(\underline{\sum}_{z=M}^{N} P)[y/x] = \begin{cases} \underline{\sum}_{z=M[y/x]}^{N[y/x]} P \text{ si } z = x \\ \underline{\sum}_{z=M[y/x]}^{N[y/x]} P[y/x] \text{ sinon} \end{cases}$$

Cette définition est-elle raisonnable ? Non. **[Exercice]** Calculer :

$$(\underline{\sum}_{\mathtt{x}=\underline{1}}^{9} \mathtt{y})[\mathtt{z}/\mathtt{y}] = \underline{\sum}_{\mathtt{x}=\underline{1}}^{9} \mathtt{z} \qquad (\underline{\sum}_{\mathtt{x}=\underline{1}}^{9} \mathtt{x})[\mathtt{z}/\mathtt{x}] = \underline{\sum}_{\mathtt{x}=\underline{1}}^{9} \mathtt{x}$$

$$(\underline{\sum}_{\mathtt{x}=\underline{1}}^{9} \mathtt{y})[\mathtt{x}/\mathtt{y}] = \underline{\sum}_{\mathtt{x}=\underline{1}}^{9} \mathtt{x}$$

L'α-conversion : définition

Il ne faut pas *capturer* de variable libre lors d'un renommage :

$$(\underline{\Sigma}_{z=M}^{N} P)[y/x] = \begin{cases} \underline{\Sigma}_{z=M[y/x]}^{N[y/x]} P \text{ si } z = x \\ \underline{\Sigma}_{k=M[y/x]}^{N[y/x]} P[k/z][y/x] \text{ sinon} \end{cases}$$

où k est une variable *fraîche*, au sens où $k \notin FV(P) \cup \{y\}$.
L'α-conversion, notée $\equiv_\alpha$, est définie comme un jugement inductif.

$$\frac{}{x \equiv_\alpha x} \qquad \frac{}{\underline{n} \equiv_\alpha \underline{n}} \qquad \frac{M \equiv_\alpha M' \qquad N \equiv_\alpha N'}{M \mathbin{\underline{+}} N \equiv_\alpha M' \mathbin{\underline{+}} N'}$$

$$\frac{M \equiv_\alpha M' \qquad N \equiv_\alpha N'}{M \mathbin{\underline{*}} N \equiv_\alpha M' \mathbin{\underline{*}} N'} \qquad \frac{\begin{array}{cc} M \equiv_\alpha M' & N \equiv_\alpha N' \\ P[z/x] \equiv_\alpha P'[z/y] & z \notin FV(P) \cup FV(P') \end{array}}{\underline{\Sigma}_{x=M}^{N} P \equiv_\alpha \underline{\Sigma}_{y=M'}^{N'} P'}$$

[Exercice] Programmer renommage et test d'α-conversion en OCaml.

Substitutivité de l'équivalence observationnelle

Quand je programme, je peux toujours remplacer un fragment de code par un autre qui lui est équivalent. Comment rendre cette idée précise ?

$$(\underline{\Sigma}_{\mathtt{x}=\underline{1}}^{\underline{4}}\, \underline{12}) \,\underline{+}\, (\underline{3} \,\underline{*}\, \underline{2}) \equiv (\underline{\Sigma}_{\mathtt{x}=\underline{1}}^{\underline{4}}\, \underline{12}) \,\underline{+}\, (\underline{1} \,\underline{+}\, \underline{5})$$

Comment séparer la partie commune des deux termes équivalents ?

$$((\underline{\Sigma}_{\mathtt{x}=\underline{1}}^{\underline{4}}\, \underline{12}) \,\underline{+}\, \mathtt{y})[\underline{3} \,\underline{*}\, \underline{2}/\mathtt{y}] \equiv ((\underline{\Sigma}_{\mathtt{x}=\underline{1}}^{\underline{4}}\, \underline{12}) \,\underline{+}\, \mathtt{y})[\underline{1} \,\underline{+}\, \underline{5}/\mathtt{y}]$$

Ce $N[M/x]$ désigne N où M a été *substitué* aux occurences libres de x.

Substitutivité de l'équivalence

Pour tout M, N_1, N_2, x, si $N_1 \equiv N_2$ alors $M[N_1/x] \equiv M[N_2/x]$.

Il nous reste à définir formellement l'opération de substitution.

Substitution

La substitution généralisant le renommage, on imite sa définition.

$$
\begin{aligned}
y[M/x] &= \begin{cases} M \text{ si } y = x \\ y \text{ sinon} \end{cases} \\
\underline{n}[M/x] &= \underline{n} \\
(N \mathbin{\underline{+}} P)[M/x] &= N[M/x] \mathbin{\underline{+}} P[M/x] \\
(N \mathbin{\underline{*}} P)[M/x] &= N[M/x] \mathbin{\underline{*}} P[M/x] \\
\left(\underline{\sum}_{y=N}^{P} O\right)[M/x] &= \begin{cases} \underline{\sum}_{z=N[M/x]}^{P[M/x]} O \text{ si } y = x \\ \underline{\sum}_{z=N[M/x]}^{P[M/x]} O[z/y][M/x] \text{ sinon} \end{cases}
\end{aligned}
$$

où, dans la dernière clause, z est fraîche, i.e., $z \notin FV(O) \cup FV(M)$.

[Exercice] Programmer la substitution en OCaml.

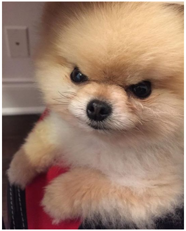
Bilan de cette séance

Un langage de programmation est défini par :

- sa syntaxe,
  - qui comprend des variables *libres* et des variables *liées*,
  - à laquelle on peut appliquer *renommage* et *substitution*,
  - sur laquelle on raisonne “à renommage des variables liées près”,
- sa sémantique,
  - une relation d’*évaluation* associant programmes et résultats (ici),
  - qu’on peut implémenter comme un interprète écrit en OCaml,
  - à partir de laquelle on peut définir l’*équivalence observationnelle*.

La prochaine séance adaptera ces concepts à une extension de Marthe.

Pour la prochaine fois



[Exercice] Formuler une sémantique équivalente de Marthe utilisant un jugement auxiliaire $P; \sigma; x; m; n \Downarrow_{\Sigma} p$ pour l'évaluation des sommes.

$$\frac{M; \sigma \Downarrow m \qquad N; \sigma \Downarrow n \qquad P; \sigma; x; m; n \Downarrow_{\Sigma} p}{\sum_{x=M}^{N} P; \sigma \Downarrow p}$$

Cette sémantique devra être équivalente aux deux autres.

(*Indice* : il s'agit de reformuler le code OCaml.)