Introduction à la sémantique des langages de programmation

Adrien Guatto & Yann Régis-Gianas

Cours de Compilation 2019-2029
Master 1 Université Paris Diderot
(version du 7 octobre 2019)

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 ?

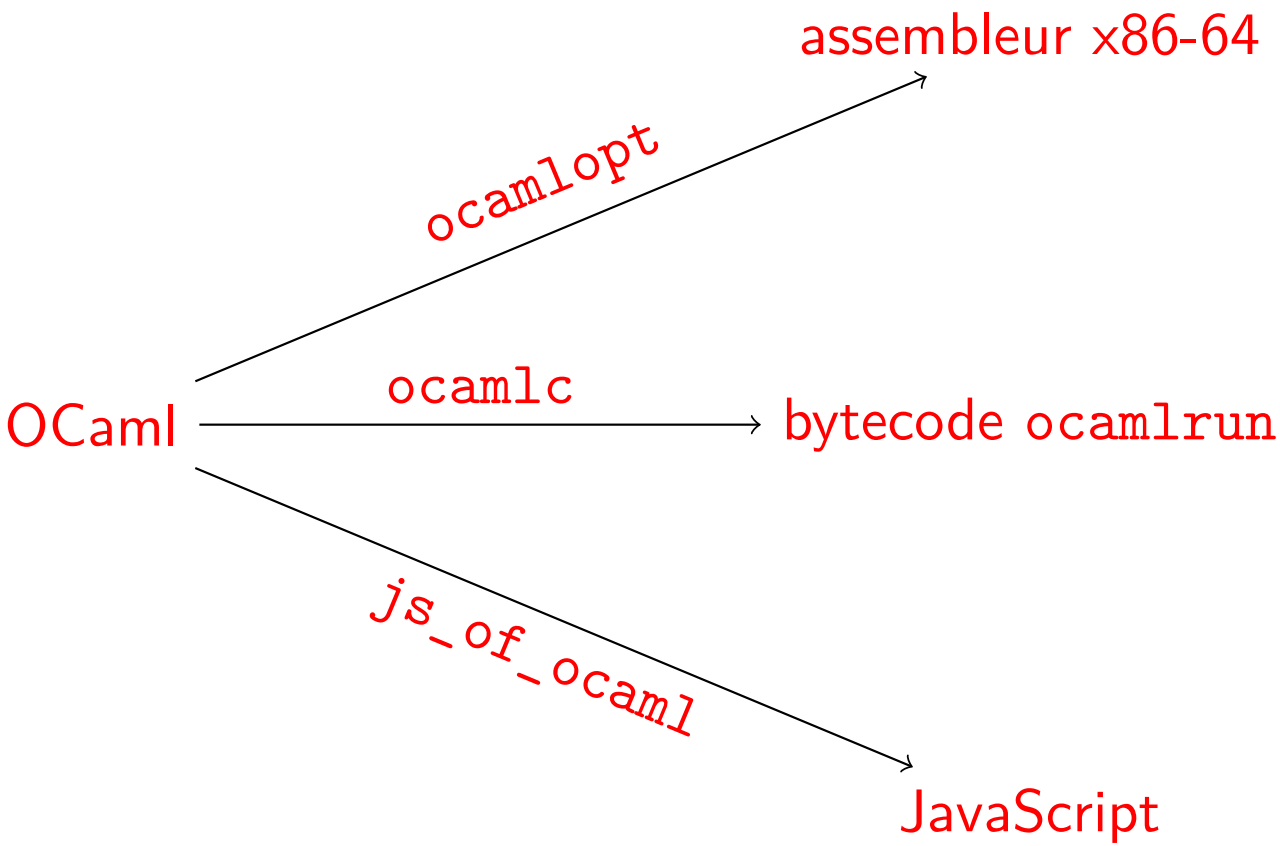
- Beaucoup plus simple qu'OCaml, Java, Python, ou le λ-calcul.
- Exhibe pourtant certaines difficultés caractéristiques.
- Peut être progressivement étendu en un langage plus riche.

Cours 1

Syntaxe et sémantique de Marthe

Contexte : les compilateurs

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 abstraits de Marthe, décrits sous la forme d'une grammaire BNF au premier cours.

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

- 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.

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

- 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 $Eval \subseteq |\mathcal{M}| \times Env \times \mathbb{N}$?

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

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

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

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

- Le terme $M \underline{+} N$ 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 $\underline{*}$.

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

- **[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.

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

2. Deux cas : évaluer P itération par itération.

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

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

Les deux approches sont *mathématiquement* équivalentes.

Fabriquer la sémantique (1/2)

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

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

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

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

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

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

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

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 $k \in \mathbb{N}$.

$$\mathit{Eval}_0 = \{(\underline{n}, \sigma, n) \mid n \in \mathbb{N}, \sigma \in \mathit{Env}\} \cup \{(x, \sigma, \sigma(x)) \mid x \in \mathit{Var}, \sigma \in \mathit{Env}\}$$

$$\begin{aligned}\mathit{Eval}_1 = \mathit{Eval}_0 &\cup \{(M \underline{+} N, \sigma, m+n) \mid (M,\sigma,m),(N,\sigma,n) \in \mathit{Eval}_0\} \\ &\cup \{(M \underline{*} N, \sigma, mn) \mid (M,\sigma,m),(N,\sigma,n) \in \mathit{Eval}_0\} \\ &\cup \left\{ \left(\underline{\sum}_{x=M}^{N} P, \sigma, 0\right) \middle| (M,\sigma,m),(N,\sigma,n) \in \mathit{Eval}_0, m > n \right\} \\ &\cup \left\{ \begin{pmatrix} \underline{\sum}_{x=M}^{N} P, \\ \sigma, p+r \end{pmatrix} \middle| \begin{matrix} (M,\sigma,m),(N,\sigma,n),(P,\sigma[x \mapsto m],p), \\ (\underline{\sum}_{x=M \underline{+} \underline{1}}^{N} P, \sigma, r) \in \mathit{Eval}_0, m \leq n \end{matrix} \right\}\end{aligned}$$

$$\mathit{Eval}_{k+1} = \mathit{Eval}_k \cup \{(M \underline{+} N, \sigma, m+n) \mid (M,\sigma,m),(N,\sigma,n) \in \mathit{Eval}_k\} \cup \ldots$$

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

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

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.

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

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

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

[Exercice] Construire un arbre de racine $\underline{\sum}_{x=\underline{1}}^{\underline{2}} x \underline{*} \underline{3};\emptyset \Downarrow n$ (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 $\sum_{x=M}^{N}(-)$ (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.

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

- Un terme M est dit *clos* si $FV(M) = \emptyset$ et *ouvert* sinon.
- **[Exercice]** $\underline{\Sigma}_{\mathrm{x}=\underline{1}}^{\underline{4}}\,\mathrm{x}$, $\underline{\Sigma}_{\mathrm{x}=\underline{1}}^{\underline{4}}\,\mathrm{y}$ et $(\underline{\Sigma}_{\mathrm{y}=\underline{1}}^{\underline{4}}\,\mathrm{y}) \underline{+} \mathrm{y}$ 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.

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

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

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.

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

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

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

Les graphes de liaison :

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

$$\mathsf{Gr}(\underline{\Sigma}^{9}_{\mathtt{x}=\underline{0}}\,(\mathtt{x}\,\underline{+}\,\mathtt{y})) = \underline{\Sigma}^{9}_{\square=\underline{0}}\,(\square\,\underline{+}\,\mathtt{y})$$

- 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.

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

$$\underline{n}[y/x] = \underline{n}$$

$$(M \underline{+} P)[y/x] = M[y/x] \underline{+} P[y/x]$$

$$(M \underline{*} P)[y/x] = M[y/x] \underline{*} P[y/x]$$

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

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

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

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

L'α-conversion : définition

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

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

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.

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

$$\frac{M \equiv_\alpha M' \qquad N \equiv_\alpha N'}{M \underline{*} N \equiv_\alpha M' \underline{*} N'} \qquad \frac{M \equiv_\alpha M' \qquad N \equiv_\alpha N' \qquad P[z/x] \equiv_\alpha P'[z/y] \qquad z \notin FV(P) \cup FV(P')}{\sum_{x=M}^{N} P \equiv_\alpha \sum_{y=M'}^{N'} P'}$$

[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 ?

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

Comment séparer la partie commune des deux termes équivalents ?

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

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.

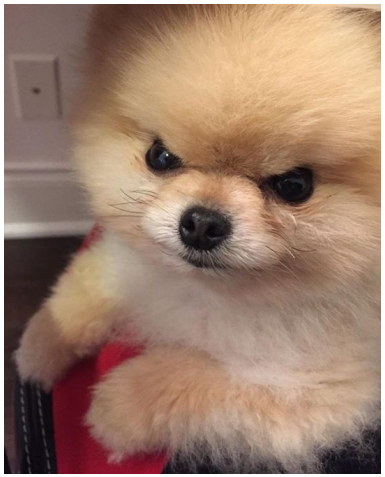
Bilan de cette séance

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.)

Cours 2

Sémantique de Marthe^{++}

Au delà de Marthe

La semaine dernière, on a étudié la syntaxe, la sémantique et l'équivalence observationnelle de Marthe.

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

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

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

Que manque-t-il à Marthe pour être un vrai langage de programmation ?

- Des *types de données* plus riches : booléens, paires, listes, etc.
- Des constructions de contrôle, par exemple les *conditionnelles*.
- Une capacité d'abstraction, par exemple des *fonctions*.
- ...

Construire Marthe^{++}

Durant cette séance, on va progressivement construire Marthe^{++} :

1. ajout d'une construction à la syntaxe, puis
2. description de la sémantique de cette construction.

On verra chemin faisant que certaines constructions justifient des modifications de notre panoplie mathématique, par exemple une généralisation de la définition de l'équivalence observationnelle.

Partir sur des bases simples

Pour généraliser Marthe, on va commencer par :

- ne pas se limiter aux entiers comme seul type de données,
- supprimer la construction de somme formelle, trop spécifique.

Cela nous mène à la syntaxe ci-dessous, qui inclue une catégorie de *valeurs*.

$$M, N, P ::= \underline{n}_{\mathtt{i}} \mid \underline{b}_{\mathtt{b}} \mid M \mathit{bop} N \mid \mathtt{let}\ x = M\ \mathtt{in}\ N \qquad (n \in \mathbb{N}, b \in \mathbb{B})$$

$$V, W ::= \underline{n}_{\mathtt{i}} \mid \underline{b}_{\mathtt{b}} \qquad \mathit{bop} ::= \underline{+} \mid \underline{*} \mid \underline{\wedge} \mid \underline{\vee} \mid \dots$$

La sémantique devient $M; \sigma \Downarrow V$, où σ associe des valeurs aux variables.

$$\frac{}{\underline{n}_{\mathtt{i}}; \sigma \Downarrow \underline{n}_{\mathtt{i}}} \qquad \frac{}{\underline{b}_{\mathtt{b}}; \sigma \Downarrow \underline{b}_{\mathtt{b}}} \qquad \frac{M; \sigma \Downarrow \underline{n_1}_{\mathtt{i}} \qquad N; \sigma \Downarrow \underline{n_2}_{\mathtt{i}}}{M \mathbin{\underline{+}} N; \sigma \Downarrow \underline{n_1 + n_2}_{\mathtt{i}}}$$

$$\frac{M; \sigma \Downarrow \underline{b_1}_{\mathtt{b}} \qquad N; \sigma \Downarrow \underline{b_2}_{\mathtt{b}}}{M \mathbin{\underline{\wedge}} N; \sigma \Downarrow \underline{b_1 \wedge b_2}_{\mathtt{b}}} \qquad \dots \qquad \frac{M; \sigma \Downarrow V \qquad N; \sigma[x \mapsto V] \Downarrow V'}{\mathtt{let}\ x = M\ \mathtt{in}\ N; \sigma \Downarrow V'}$$

[Exercice] Ajouter négation booléenne $\underline{\neg}$ et comparaison d'entiers $\underline{\leq}$.

Les paires

On voudrait manipuler des paires de valeurs. Qu'ajouter à la syntaxe ?

$$M, N, P ::= \cdots \mid (M, N) \mid \texttt{fst}\ M \mid \texttt{snd}\ M \qquad V, W ::= \underline{n}_{\texttt{i}} \mid \underline{b}_{\texttt{b}} \mid (V, W)$$

Quelle sémantique ?

$$\frac{M; \sigma \Downarrow V \qquad N; \sigma \Downarrow W}{(M, N); \sigma \Downarrow (V, W)} \qquad \frac{M; \sigma \Downarrow (V, W)}{\texttt{fst}\ M; \sigma \Downarrow V} \qquad \frac{M; \sigma \Downarrow (V, W)}{\texttt{snd}\ M; \sigma \Downarrow W}$$

[Exercice] Donner la valeur V telle que le jugement

$$\texttt{let x} = (\underline{3}_{\texttt{i}} \underline{+} \underline{5}_{\texttt{i}}, \underline{\mathit{true}}_{\texttt{b}})\ \texttt{in}\ (\texttt{snd x}, \texttt{fst x}); \emptyset \Downarrow V$$

soit dérivable, ainsi que la dérivation correspondante.

Les conditionnelles

On voudrait ajouter une instruction conditionnelle à la OCaml au langage.

$$M ::= \cdots \mid \texttt{if}\ M\ \texttt{then}\ N\ \texttt{else}\ P$$

Quelle serait sa sémantique ? “Si l'évaluation de M renvoie vrai, le résultat est celui de N, sinon c'est celui de P”.

$$\frac{M;\sigma \Downarrow \underline{true}_{\mathsf{b}} \qquad N;\sigma \Downarrow V}{\texttt{if}\ M\ \texttt{then}\ N\ \texttt{else}\ P;\sigma \Downarrow V} \qquad \frac{M;\sigma \Downarrow \underline{false}_{\mathsf{b}} \qquad P;\sigma \Downarrow V}{\texttt{if}\ M\ \texttt{then}\ N\ \texttt{else}\ P;\sigma \Downarrow V}$$

[Exercice] L'équation donnée ci-dessous vous semble-t-elle valide dans le langage décrit jusqu'ici, pour M, N, P clos et s'évaluant sans erreurs ?

$\texttt{if}\ M\ \texttt{then}\ N\ \texttt{else}\ P \equiv \texttt{let x} = (N, P)\ \texttt{in if}\ M\ \texttt{then fst x else snd x}.$

Et son analogue en OCaml ?

De nouvelles erreurs ?

Lors du cours précédent, on a vu que Marthe vérifiait la propriété suivante.

Propriété (Marthe uniquement)

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

Est-ce encore le cas pour le langage du transparent précédent ?

- Est-ce que si $M; \sigma \Downarrow V$ alors $FV(M) \subseteq \sigma^{-1}(\mathbb{N})$? Non :

$$\texttt{if } \underline{\mathit{true}}_{\texttt{b}} \texttt{ then } \underline{42}_{\texttt{i}} \texttt{ else x}; \emptyset \Downarrow \underline{\mathit{true}}_{\texttt{b}}.$$

- Est-ce que si $FV(M) \subseteq \sigma^{-1}(\mathbb{N})$, il existe V tel que $M; \sigma \Downarrow V$? Non, à cause de potentielles erreurs, comme dans le terme

$$\texttt{if } (\underline{1}_{\texttt{i}}, \underline{2}_{\texttt{i}}) \texttt{ then } \underline{\mathit{true}}_{\texttt{b}} \texttt{ else } \underline{\mathit{false}}_{\texttt{b}}; \emptyset \Downarrow ??.$$

  On verra à la prochaine séance comment s'en prémunir.

Les fonctions de seconde classe (1/2)

On ajoute des constructions permettant d'abstraire un sous-terme pour pouvoir le réutiliser plusieurs fois : définition de fonction et application.

$$M, N, P ::= \cdots \mid \texttt{def}\ f\ x = M\ \texttt{in}\ N \mid f\ M$$

On désigne par f, g, h des variables distinctes de x, y, z. On dit que ces deux familles de variables constituent des espaces de noms différents. Par conséquent, on ne peut pas "mélanger" variables ordinaires et noms de fonction, et donc, pour des raisons purement syntaxiques :

- un terme ne peut pas s'évaluer vers une fonction,

  $\texttt{def f x = x in f}$ n'étant pas un terme,

- une fonction ne peut pas prendre en argument une autre fonction,

  $\texttt{def f g = g}\ \underline{1}_{\texttt{i}}\ \texttt{in}\ M$ non plus.

Les objets soumis à de telles restrictions sont dits de seconde classe.

Les fonctions de seconde classe (2/2)

La sémantique doit maintenant être paramétrée par deux environnements, un par espace de nom :

- l'environnement σ usuel, associant aux variables $\mathtt{x}, \mathtt{y}, \mathtt{z}$ des valeurs,
- un nouvel environnement ϕ, associant aux variables $\mathtt{f}, \mathtt{g}, \mathtt{h}$ des… ?

Essayons d'écrire les règles de notre jugement $M; \sigma; \phi \Downarrow V$.

$$\frac{}{x; \sigma; \phi \Downarrow \sigma(x)} \qquad \cdots \qquad \frac{M; \sigma; \phi \Downarrow \underline{m}_{\mathtt{i}} \qquad N; \sigma; \phi \Downarrow \underline{n}_{\mathtt{i}}}{M \underline{+} N; \sigma; \phi \Downarrow \underline{m+n}_{\mathtt{i}}} \qquad \cdots$$

$$\frac{N; \sigma; \phi[f \mapsto ?] \Downarrow V}{\mathtt{def}\ f\ x = M\ \mathtt{in}\ N; \sigma; \phi \Downarrow V} \qquad \frac{N; \sigma; \phi \Downarrow V_a \qquad ?; ?[? \mapsto V_a]; ? \Downarrow V}{f\ N; \sigma; \phi \Downarrow V}$$

Pour exécuter un appel à la fonction f, on doit :

- connaître au moins son corps et la variable qui lui sert d'argument,
- déterminer quels environnements utiliser (plusieurs choix possibles !).

Choix de portée (1/3)

On peut se restreindre aux fonctions dont le corps est clos.

$$\frac{N;\sigma;\phi[f \mapsto (x, M)] \Downarrow V}{\texttt{def } f\ x = M \texttt{ in } N;\sigma;\phi \Downarrow V} \qquad \frac{\phi(f) = (x, M) \quad N;\sigma;\phi \Downarrow V_a \quad M;\emptyset[x \mapsto V_a];\emptyset \Downarrow V}{f\ N;\sigma;\phi \Downarrow V}$$

[Exercice] Pour chaque M_i ci-dessous, déterminer s'il existe une valeur V_i telle qu'une dérivation $M_i;\emptyset;\emptyset \Downarrow V$ existe, et les donner le cas échéant.

$M_1 \overset{\text{def}}{=} \texttt{let x} = \underline{2}_{\texttt{i}} \texttt{ in def f x} = \texttt{x} \underline{+} \underline{1}_{\texttt{i}} \texttt{ in f x}$ $\quad V_1 = \underline{3}_{\texttt{i}}$

$M_2 \overset{\text{def}}{=} \texttt{let x} = \underline{2}_{\texttt{i}} \texttt{ in def f y} = \texttt{y} \underline{+} \texttt{x in f x}$ $\quad V_2 = \times$

$M_3 \overset{\text{def}}{=} \texttt{def f x} = \texttt{x} \underline{+} \underline{1}_{\texttt{i}} \texttt{ in def g x} = \texttt{f x in g} \underline{1}_{\texttt{i}}$ $\quad V_3 = \times$

$M_4 \overset{\text{def}}{=} \texttt{let x} = \underline{2}_{\texttt{i}} \texttt{ in def f y} = \texttt{y} \underline{+} \texttt{x in let x} = \underline{3}_{\texttt{i}} \texttt{ in f x}$ $\quad V_4 = \times$

Choix de portée (2/3)

On peut choisir la portée dynamique.

$$\frac{N;\sigma;\phi[f \mapsto (x, M)] \Downarrow V}{\texttt{def } f\ x = M \texttt{ in } N;\sigma;\phi \Downarrow V} \qquad \frac{\phi(f) = (x, M) \quad N;\sigma;\phi \Downarrow V_a \quad M;\sigma[x \mapsto V_a];\phi \Downarrow V}{f\ N;\sigma;\phi \Downarrow V}$$

[Exercice] Pour chaque M_i ci-dessous, déterminer s'il existe une valeur V_i telle qu'une dérivation $M_i;\emptyset;\emptyset \Downarrow V$ existe, et les donner le cas échéant.

$$M_1 \stackrel{\text{def}}{=} \texttt{let x} = \underline{2}_{\texttt{i}} \texttt{ in def f x} = \texttt{x} \underline{+} \underline{1}_{\texttt{i}} \texttt{ in f x} \qquad V_1 = \underline{3}_{\texttt{i}}$$
$$M_2 \stackrel{\text{def}}{=} \texttt{let x} = \underline{2}_{\texttt{i}} \texttt{ in def f y} = \texttt{y} \underline{+} \texttt{x in f x} \qquad V_2 = \underline{4}_{\texttt{i}}$$
$$M_3 \stackrel{\text{def}}{=} \texttt{def f x} = \texttt{x} \underline{+} \underline{1}_{\texttt{i}} \texttt{ in def g x} = \texttt{f x in g} \underline{1}_{\texttt{i}} \qquad V_3 = \underline{2}_{\texttt{i}}$$
$$M_4 \stackrel{\text{def}}{=} \texttt{let x} = \underline{2}_{\texttt{i}} \texttt{ in def f y} = \texttt{y} \underline{+} \texttt{x in let x} = \underline{3}_{\texttt{i}} \texttt{ in f x} \qquad V_4 = \underline{6}_{\texttt{i}}$$

Choix de portée (3/3)

On peut choisir la portée lexicale grâce à l'usage de fermetures.

$$\frac{N;\sigma;\phi[f \mapsto (x, M, \sigma, \phi)] \Downarrow V}{\texttt{def } f\ x = M \texttt{ in } N;\sigma;\phi \Downarrow V} \qquad \frac{\phi(f) = (x, M, \sigma_f, \phi_f) \quad N;\sigma;\phi \Downarrow V_a \quad M;\sigma_f[x \mapsto V_a];\phi_f \Downarrow V}{f\ N;\sigma;\phi \Downarrow V}$$

[Exercice] Pour chaque M_i ci-dessous, déterminer s'il existe une valeur V_i telle qu'une dérivation $M_i;\emptyset;\emptyset \Downarrow V$ existe, et les donner le cas échéant.

$$M_1 \stackrel{\text{def}}{=} \texttt{let x} = \underline{2}_{\texttt{i}} \texttt{ in def f x} = \texttt{x} \underline{+} \underline{1}_{\texttt{i}} \texttt{ in f x} \qquad V_1 = \underline{3}_{\texttt{i}}$$

$$M_2 \stackrel{\text{def}}{=} \texttt{let x} = \underline{2}_{\texttt{i}} \texttt{ in def f y} = \texttt{y} \underline{+} \texttt{x in f x} \qquad V_2 = \underline{4}_{\texttt{i}}$$

$$M_3 \stackrel{\text{def}}{=} \texttt{def f x} = \texttt{x} \underline{+} \underline{1}_{\texttt{i}} \texttt{ in def g x} = \texttt{f x in g} \underline{1}_{\texttt{i}} \qquad V_3 = \underline{2}_{\texttt{i}}$$

$$M_4 \stackrel{\text{def}}{=} \texttt{let x} = \underline{2}_{\texttt{i}} \texttt{ in def f y} = \texttt{y} \underline{+} \texttt{x in let x} = \underline{3}_{\texttt{i}} \texttt{ in f x} \qquad V_4 = \underline{5}_{\texttt{i}}$$

Pourquoi choisir la portée lexicale ?

- Plus simple (permet le raisonnement local), plus utile.
- Plus naturelle du point de vue de l'équivalence observationnelle.

Les fonctions de première classe

Autoriser les fonctions à passer/renvoyer des fonctions à leurs appelants nous rapproche des langages réels (p. ex. OCaml mais aussi Java, C…).

Il suffit de remplacer la construction `def` par une construction créant une fonction anonyme, et d'ajouter les fermetures aux valeurs.

$$M, N, P ::= \cdots \mid \texttt{fun}\ x.\ M \mid M\ N$$
$$V, W ::= \cdots \mid (x, M, \sigma)_{\mathsf{c}}$$

La sémantique ressemble à celle des fonctions de seconde classe respectant la portée lexicale, mais sans environnement supplémentaire.

$$\frac{}{\texttt{fun}\ x.\ M; \sigma \Downarrow (x, M, \sigma)_{\mathsf{c}}}$$

$$\frac{M; \sigma \Downarrow (x, M_f, \sigma_f)_{\mathsf{c}} \qquad N; \sigma \Downarrow V_a \qquad M_f; \sigma_f[x \mapsto V_a] \Downarrow V}{M\ N; \sigma \Downarrow V}$$

Bilan d'étape

La syntaxe de Marthe^{++} est la suivante.

$$
\begin{aligned}
M, N, P &::= \underline{n}_{\mathtt{i}} \mid \underline{b}_{\mathtt{b}} \mid M \mathit{bop} N \mid \mathit{uop}\, M \mid \mathtt{let}\ x = M\ \mathtt{in}\ N \mid (M, N) \mid \\
&\quad \mid\ \mathtt{fst}\ M \mid \mathtt{snd}\ M \mid \mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ P \mid \mathtt{fun}\ x.\, M \mid M\ N \\
V, W &::= \underline{n}_{\mathtt{i}} \mid \underline{b}_{\mathtt{b}} \mid (V, W) \mid (x, M, \sigma)_{\mathtt{c}} \\
\mathit{bop} &::= \underline{+} \mid \underline{*} \mid \underline{\wedge} \mid \underline{\vee} \mid \underline{\leq} \\
\mathit{uop} &::= \underline{\neg}
\end{aligned}
$$

On pourra rajouter des opérateurs binaires ou unaires au besoin.
[Exercice] Implémenter l'évaluateur correspondant à la sémantique.

Équivalence observationnelle (1/2)

Adaptons naïvement l'équivalence observationnelle définie pour Marthe.

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

Est-ce un choix raisonnable ?
[Exercice] Chercher deux termes M, N qui devraient intuitivement être équivalents mais tels que $M \not\equiv N$ pour la définition ci-dessus.

On peut par exemple choisir $M = \texttt{fun x.}\,(\underline{1}_{\texttt{i}} \mathbin{\underline{+}} \underline{1}_{\texttt{i}})$ et $N = \texttt{fun x.}\,\underline{2}_{\texttt{i}}$.

Pourquoi M et N devraient-ils être équivalents ?

Dans tout terme P s'évaluant vers un booléen ou un entier et où M apparaît comme sous-terme, remplacer M par N n'affecte pas le résultat.

On veut élargir la définition de l'équivalence via une notion de *contexte*.

Équivalence observationnelle (2/2)

Un contexte est un terme K contenant *exactement une occurrence* du symbole spécial $\square$, qu'on appelle souvent le "trou" de K.

$$K ::= \square \mid K \; bop \; M \mid M \; bop \; K \mid uop \; K \mid \texttt{let}\; x = K \;\texttt{in}\; M \\ \mid \quad \texttt{let}\; x = M \;\texttt{in}\; K \mid \texttt{fun}\; x.\, K \mid K \; M \mid M \; K$$

On peut "boucher" le trou d'un contexte K avec un terme M pour obtenir un terme $K[M]$. Cette opération est définie récursivement sur K.

$$\square[M] = M$$
$$(K \; bop \; N)[M] = K[M] \; bop \; N$$
$$(N \; bop \; K)[M] = N \; bop \; K[M]$$
$$\ldots$$

On utilise ces définitions pour raffiner l'équivalence observationnelle :

$$M \equiv N \stackrel{\text{def}}{=} \forall K \in Ctx, \forall b \in \mathbb{B}, K[M] \Downarrow \underline{b}_{\mathsf{b}} \Leftrightarrow K[N] \Downarrow \underline{b}_{\mathsf{b}}$$

où $M \Downarrow V$ est un raccourci pour $M; \emptyset \Downarrow V$.

Bilan de cette séance

La méthodologie vue dans cette séquence de cours s'étend :

- à un langage un peu plus réaliste, comme on l'a vu aujourd'hui,
  - Des types de données, des instructions de contrôle, des fonctions,
  - plus de comportements (*erreurs*...),
  - mais les mêmes concepts de base de la syntaxe.
- mais aussi, bien au delà, à des langages beaucoup plus riches !
  - L'équivalence observationnelle sera toujours *contextuelle*,
  - on implémentera très souvent la portée lexicale via les *fermetures*.

La prochaine séance traitera des systèmes de types.

Cours 3

Typage de Marthe^{++}

Programmer en Marthe^{++}

Durant la séance précédente, nous avons augmenté Marthe avec des booléens, entiers, paires, conditionnelles, et fonctions de première classe.

Le résultat, Marthe^{++}, peut sembler inexpressif, mais l'est-il vraiment ?

- **[Exercice]** Chercher V t.q. $\Omega \Downarrow V$, où $\Delta \stackrel{\text{def}}{=} \texttt{fun x. x x}$ et $\Omega \stackrel{\text{def}}{=} \Delta\ \Delta$.
- **[Exercice]** Chercher V tel que $F\ \underline{4}_{\texttt{i}} \Downarrow V$, avec :

$$F \stackrel{\text{def}}{=} Y\ M,$$

$$Y \stackrel{\text{def}}{=} \texttt{fun f. (fun x. f (fun y. x x y)) (fun x. f (fun y. x x y))},$$

$$M \stackrel{\text{def}}{=} \texttt{fun F. fun x. if x} \leq \underline{1}_{\texttt{i}} \texttt{ then } \underline{1}_{\texttt{i}} \texttt{ else x} * \texttt{(F (x} - \underline{1}_{\texttt{i}}\texttt{))}.$$

- **[Exercice]** Proposer une traduction de Marthe dans Marthe^{++}.

Expressivité du point de vue de la théorie de la calculabilité

Marthe^{++} peut implémenter toutes les fonctions calculables de $\mathbb{N}$ dans $\mathbb{N}$.

Comportement des programmes Marthe^{++}

Un terme de Marthe^{++}, dans un environnement fixé, peut soit s'évaluer vers une valeur, soit planter à cause d'une erreur de type, soit diverger.

[Exercice] Comment notre sémantique traite-t-elle ces deux derniers cas ?

$$\{V \mid \Omega \Downarrow V\} \qquad \text{vs.} \qquad \{V \mid \underline{15}_{\mathtt{i}}\ \underline{15}_{\mathtt{i}} \Downarrow V\} \qquad ?$$

Les deux ensembles sont vides !

- Notre sémantique ne distingue pas l'erreur de la divergence.
- Cela mène à une version de l'équivalence observationnelle douteuse.
- Révisons un peu la définition de cette dernière.

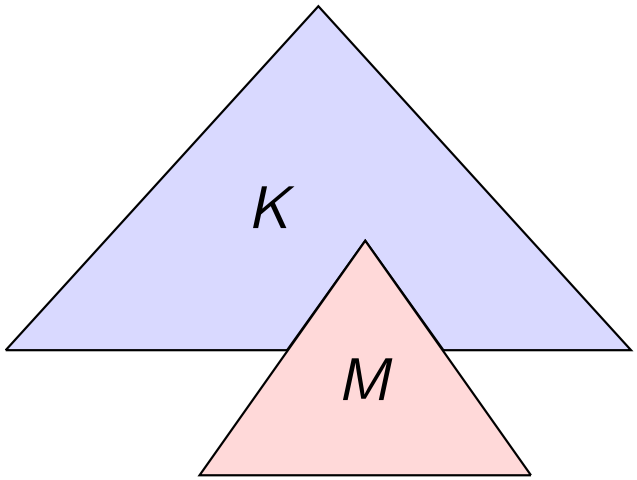
Retour sur la notion de contexte

Pour mieux définir l'équivalence, on avait introduit la notion de contexte.

- Un contexte représente un terme avec un unique emplacement distingué, marqué par le symbole spécial $\square$: c'est un "terme à trou".

$$K ::= \square \mid K \mathit{bop} M \mid M \mathit{bop} K \mid \mathit{uop}\, K \mid \texttt{let}\ x = K\ \texttt{in}\ M \mid \texttt{let}\ x = M\ \texttt{in}\ K \mid \texttt{fun}\ x.\, K \mid K\ M \mid M\ K$$

- Le terme $K[M]$ est obtenu en "bouchant" le trou de K avec M.



$$K \stackrel{\text{def}}{=} \texttt{let x} = \underline{1}_{\texttt{i}}\ \texttt{in}\ \square \mathbin{\underline{*}} \underline{2}_{\texttt{i}}$$

$$M \stackrel{\text{def}}{=} \texttt{x} \mathbin{\underline{+}} \underline{1}_{\texttt{i}}$$

$$K[M] = \texttt{let x} = \underline{1}_{\texttt{i}}\ \texttt{in}\ (\texttt{x} \mathbin{\underline{+}} \underline{1}_{\texttt{i}}) \mathbin{\underline{*}} \underline{2}_{\texttt{i}}$$

Équivalence observationnelle, erreurs et divergence

Durant la séance précédente, nous avions donné la définition suivante :

$$M \equiv N \stackrel{\text{def}}{=} \forall K \in \mathit{Ctx}, \forall V \in \mathit{Val}, K[M] \Downarrow V \Leftrightarrow K[N] \Downarrow V.$$

[Exercice] Montrer qu'elle identifie au moins autant de termes que la précédente : $M \equiv N \Rightarrow \forall \sigma \in \mathit{Env}, \forall V \in \mathit{Val},\ M;\sigma \Downarrow V \Leftrightarrow N;\sigma \Downarrow V$.
[Exercice] Discutez des optimisations que cette définition autorise un compilateur Marthe^{++} à appliquer aux programmes qui divergent.

Deux corrections possibles

- Un prédicat d'erreur $M\lightning$, et la définition suivante pour $M \equiv N$:

  $$\forall K \in \mathit{Ctx}, (K[M]\lightning \wedge K[N]\lightning) \vee (\forall V \in \mathit{Val},\ K[M] \Downarrow V \Leftrightarrow K[N] \Downarrow V).$$

- Un prédicat de divergence $M\uparrow$, et la définition suivante pour $M \equiv N$:

  $$\forall K \in \mathit{Ctx}, (K[M]\uparrow \wedge K[N]\uparrow) \vee (\forall V \in \mathit{Val},\ K[M] \Downarrow V \Leftrightarrow K[N] \Downarrow V).$$

Définir un prédicat d'erreur

On peut essayer de définir un prédicat d'erreur :

$$M; \sigma \lightning \Leftrightarrow \text{“l'évaluation de } M \text{ dans } \sigma \text{ rencontre une erreur”.}$$

À quoi ressemblent ses règles ? (On écrit $V \not\sim \mathtt{i}$ quand V n'est pas une valeur entière, $V \not\sim \mathtt{c}$ quand V n'est pas une valeur fermeture, etc.)

$$\frac{M; \sigma \lightning}{M \underline{+} N; \sigma \lightning} \qquad \frac{M; \sigma \Downarrow V \quad V \not\sim \mathtt{i}}{M \underline{+} N; \sigma \lightning} \qquad \frac{N; \sigma \lightning}{M \underline{+} N; \sigma \lightning} \qquad \frac{N; \sigma \Downarrow V \quad V \not\sim \mathtt{i}}{M \underline{+} N; \sigma \lightning} \qquad \dots$$

$$\frac{M; \sigma \lightning}{M\ N; \sigma \lightning} \qquad \frac{M; \sigma \Downarrow V \quad V \not\sim \mathtt{c}}{M\ N; \sigma \lightning} \qquad \frac{N; \sigma \lightning}{M\ N; \sigma \lightning} \qquad \frac{M; \sigma \Downarrow (x, M_f, \sigma_f)_{\mathtt{c}} \quad N; \sigma \Downarrow V_a \quad M_f; \sigma_f[x \mapsto V_a] \lightning}{M\ N; \sigma \lightning}$$

Peu satisfaisant : beaucoup de redondance avec les règles de $M; \sigma \Downarrow V$!

Définir un prédicat de divergence

[Exercice] Essayer de définir un jugement inductif $M; \sigma \uparrow$.

- Les dérivations d'un jugement inductif sont par définitions des arbres *finis*, or la divergence est un comportement *infinitaire*.
- Définir un jugement de divergence *coinductif*, dont les dérivations sont des arbres *infinis*, dépasse le cadre de cette introduction.
- À la place, on va adopter un style de sémantique dit *à petits pas*.

Sémantiques à petits pas

- Une relation de réduction "atomique" entre termes.

$$M \rightsquigarrow M' \Leftrightarrow \text{"le terme } M \text{ se réduit en } M' \text{ en un pas de calcul"}$$

- Un terme M diverge s'il existe une séquence $(M_i)_{i \geq 1}$ telle que :

$$M \rightsquigarrow M_1 \rightsquigarrow M_2 \rightsquigarrow \cdots \rightsquigarrow M_n \rightsquigarrow M_{n+1} \rightsquigarrow \ldots$$

Sémantique à petits pas de Marthe^{++} (1/4)

Environnements ou substitutions ?

- Le choix d'utiliser un environnement σ est indépendant de la question de la dichotomie entre "grand pas" et "petits pas".

$$\frac{M;\sigma \Downarrow V \qquad N;\sigma[x \mapsto V] \Downarrow V'}{\texttt{let}\ x = M\ \texttt{in}\ N;\sigma \Downarrow V'} \quad \text{v.s.} \quad \frac{M \Downarrow V \qquad N[V/x] \Downarrow V'}{\texttt{let}\ x = M\ \texttt{in}\ N \Downarrow V'}$$

- Pour varier les plaisirs, on va définir la sémantique à petits en utilisant la substitution plutôt qu'un environnement.

Valeurs

- En l'absence d'environnements, plus besoin de fermetures.

$$V, W ::= \underline{n}_{\mathtt{i}} \mid \underline{b}_{\mathtt{b}} \mid (V, W) \mid \texttt{fun}\ x.\, M$$

- Les valeurs forment maintenant un strict sous-ensemble des termes.

Sémantique à petits pas de Marthe^{++} (2/4)

On définit tout d'abord une relation dite de β-*réduction*, notée $\rightsquigarrow_\beta$.

$$\underline{m}_{\mathtt{i}} \pm \underline{n}_{\mathtt{i}} \rightsquigarrow_\beta \underline{m \pm n}_{\mathtt{i}}$$

$$\ldots$$

$$\mathtt{let}\ x = V\ \mathtt{in}\ M \rightsquigarrow_\beta M[V/x]$$

$$\mathtt{fst}\ (V, W) \rightsquigarrow_\beta V$$

$$\mathtt{snd}\ (V, W) \rightsquigarrow_\beta W$$

$$\mathtt{if}\ \underline{\mathit{true}}_{\mathtt{b}}\ \mathtt{then}\ M\ \mathtt{else}\ N \rightsquigarrow_\beta M$$

$$\mathtt{if}\ \underline{\mathit{false}}_{\mathtt{b}}\ \mathtt{then}\ M\ \mathtt{else}\ N \rightsquigarrow_\beta N$$

$$(\mathtt{fun}\ x.\ M)\ V \rightsquigarrow_\beta M[V/x]$$

[Exercice] Chercher M tel que $\Omega \rightsquigarrow_\beta M$.

[Exercice] Chercher M, N t.q. $\mathtt{let\ x} = \underline{3}_{\mathtt{i}}\ \mathtt{in}\ \underline{1}_{\mathtt{i}} \pm (\underline{2}_{\mathtt{i}} \pm \mathtt{x}) \rightsquigarrow_\beta M \rightsquigarrow_\beta N$.

Sémantique à petits pas de Marthe^{++} (3/4)

La β-réduction s'applique uniquement à la racine d'un terme. Pour traiter p. ex. $\underline{1}_{\mathtt{i}} \pm (\underline{2}_{\mathtt{i}} \pm \underline{3}_{\mathtt{i}})$, il faut pouvoir réduire en profondeur.
Pour définir $\rightsquigarrow$, on autorise l'application de $\rightsquigarrow_\beta$ ailleurs qu'à la racine des termes. Mais où, précisément ?
On peut essayer d'autoriser l'application à un sous-terme arbitraire. C'est très facile à formuler à l'aide de notre notion de contexte.

$$\frac{M \rightsquigarrow_\beta N}{K[M] \rightsquigarrow K[N]}$$

[Exercice] Critiquer la relation ainsi définie.

- Elle permet d'appliquer $\rightsquigarrow_\beta$ à trop d'emplacements : dans les branches des conditionnelles, dans le corps des fonctions, etc.
- On peut avoir $M \rightsquigarrow M'$, $M \rightsquigarrow M''$ et $M' \neq M''$ (non-déterminisme).
- La relation est incompatible avec notre définition de $M\uparrow$.

[Exercice] Montrer que la définition entraîne $\mathtt{if}\ \underline{\mathit{true}}_{\mathtt{b}}\ \mathtt{then}\ \underline{1}_{\mathtt{i}}\ \mathtt{else}\ \Omega\uparrow$.

Sémantique à petits pas de Marthe^{++} (4/4)

Imaginions qu'on veuille définir $\rightsquigarrow$ par un jugement inductif.

$$\frac{M \rightsquigarrow_\beta N}{M \rightsquigarrow N} \qquad \frac{M \rightsquigarrow M'}{M \mathit{\ bop\ } N \rightsquigarrow M' \mathit{\ bop\ } N} \qquad \frac{N \rightsquigarrow N'}{V \mathit{\ bop\ } N \rightsquigarrow V \mathit{\ bop\ } N'}$$

$$\frac{M \rightsquigarrow M'}{\mathit{uop}\ M \rightsquigarrow \mathit{uop}\ M'} \qquad \frac{M \rightsquigarrow M'}{\texttt{let}\ x = M\ \texttt{in}\ N \rightsquigarrow \texttt{let}\ x = M'\ \texttt{in}\ N} \qquad \ldots$$

On peut la reformuler de façon plus concise via les contextes d'évaluation.

$$\begin{aligned} E ::= {} & \square \mid E \mathit{\ bop\ } M \mid V \mathit{\ bop\ } E \mid \mathit{uop}\ E \\ \mid {} & \texttt{let}\ x = E\ \texttt{in}\ M \mid (E, M) \mid (V, E) \mid \texttt{fst}\ E \\ \mid {} & \texttt{snd}\ E \mid \texttt{if}\ E\ \texttt{then}\ M\ \texttt{else}\ N \mid E\ M \mid V\ E \end{aligned} \qquad \frac{M \rightsquigarrow_\beta N}{E[M] \rightsquigarrow E[N]}$$

Avec cette définition, notre sémantique à petits pas :

- ne réduit pas les branches conditionnelles, le corps des fonctions, etc.
- réduit de gauche à droite les opérateurs binaires, paires, etc.

Correspondance entre petits pas et grands pas

Quel est le lien entre sémantique à grands pas et sémantique à petits pas ?

Définition

La *clôture réflexive transitive* d'une relation $R \subseteq A \times A$ est définie comme la plus petite relation réflexive et transitive $R^* \subseteq A \times A$ telle que $R \subseteq R^*$.

[Exercice] Formuler R^* sous la forme d'un jugement inductif $(x, y) \in R^*$.

Théorème

Pour tout terme M et valeur V, on a $M \Downarrow V \Leftrightarrow M \rightsquigarrow^* V$.

Quelques remarques :

- l'énoncé du théorème est légèrement informel (pas les mêmes valeurs),
- la preuve nécessite de généraliser l'énoncé aux environnements σ,
- Le cas crucial de l'application ne fonctionne qu'avec la portée lexicale !

$$(\texttt{fun}\, x.\, M)\ V \rightsquigarrow_\beta M[V/x]$$

Se débarrasser des erreurs

Comment se manifestent les erreurs dans la sémantique à petits pas ? Par des termes qui se réduisent vers un terme “coincé”.

$$\texttt{if}\ (\underline{1}_{\texttt{i}} \underline{+} \underline{3}_{\texttt{i}}, \underline{\mathit{true}}_{\texttt{b}})\ \texttt{then}\ \underline{1}_{\texttt{i}}\ \texttt{else}\ \underline{2}_{\texttt{i}} \rightsquigarrow \texttt{if}\ (\underline{4}_{\texttt{i}}, \underline{\mathit{true}}_{\texttt{b}})\ \texttt{then}\ \underline{1}_{\texttt{i}}\ \texttt{else}\ \underline{2}_{\texttt{i}} \not\rightsquigarrow$$

D’où vient le problème ? La condition doit s’évaluer vers un booléen !

On va abstraire les programmes par des formules logiques très simples, les types, classifiant les résultats des programmes.

$$M : A \Leftrightarrow \text{“le terme } M \text{ calcule une valeur de type } A\text{”}$$

Quelles peuvent être les règles du jugement inductif $M : A$?

$$\frac{}{\underline{b}_{\texttt{b}} : \textbf{bool}} \qquad \frac{}{\underline{i}_{\texttt{i}} : \textbf{int}} \qquad \frac{M : \textbf{bool} \qquad N : A \qquad P : A}{\texttt{if}\ M\ \texttt{then}\ N\ \texttt{else}\ P : A} \qquad \ldots$$

[Exercice] Quid des variables ? Définitions locales ? Fonctions anonymes ?

Marthe^{++} simplement typé (1/4) : contextes

Pour traiter du cas des variables, on a besoin de contextes de typage.

$$A, B ::= \textbf{int} \mid \textbf{bool} \mid A \times B \mid A \Rightarrow B \qquad \Gamma ::= \cdot \mid \Gamma, x : A$$

La notion de contexte permet de généraliser le typage aux termes ouverts.

$\Gamma \vdash M : A \Leftrightarrow$ "Le terme M calcule une valeur de type A si ses variables libres sont remplacées par des valeurs respectant les types prescrits par Γ."

Remarques sur les contextes

- On note $dom(\Gamma)$ pour l'ensemble des variables apparaîssant Γ.
- On supposera qu'une variable apparaît *au plus* une fois dans un contexte (quitte à α-convertir nos termes).
- Si $x \in dom(\Gamma)$, on écrira $\Gamma(x)$ pour l'unique A associé à x dans Γ.

Marthe^{++} simplement typé (2/4) : les règles

$$\frac{\Gamma(x) = A}{\Gamma \vdash x : A} \qquad \frac{}{\Gamma \vdash \underline{b}_{\mathtt{b}} : \textbf{bool}} \qquad \frac{}{\Gamma \vdash \underline{i}_{\mathtt{i}} : \textbf{int}} \qquad \frac{\Gamma \vdash M : \textbf{int} \qquad \Gamma \vdash N : \textbf{int}}{\Gamma \vdash M \underline{+} N : \textbf{int}}$$

$$\ldots \qquad \frac{\Gamma \vdash M : A \qquad \Gamma, x : A \vdash N : B}{\Gamma \vdash \mathtt{let}\ x = M\ \mathtt{in}\ N : B} \qquad \frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A \times B}$$

$$\frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \mathtt{fst}\ M : A} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \mathtt{snd}\ M : B} \qquad \frac{\Gamma \vdash M : \textbf{bool} \qquad (\Gamma \vdash N_i : A)_{i \in \{1\,2\}}}{\Gamma \vdash \mathtt{if}\ M\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2 : A}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \mathtt{fun}\ x.\, M : A \Rightarrow B} \qquad \frac{\Gamma \vdash M : A \Rightarrow B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}$$

Marthe^{++} simplement typé (3/4) : sûreté

Lemme : progrès

Si $\vdash M : A$ alors soit M est une valeur, soit il existe M' avec $M \rightsquigarrow M'$.

Lemme : réduction du sujet

Si $\vdash M : A$ et $M \rightsquigarrow M'$ alors $\vdash M' : A$.

Théorème : sûreté du typage

Si $\vdash M : A$ alors soit $M\uparrow$ soit il existe $\vdash V : A$ tel que $M \rightsquigarrow^* V$.

Des paraphrases informelles mais frappantes de ce théorème :

Well-typed programs can't go wrong – Milner (1978)

Well-typed programs don't get stuck – Wright & Felleisen (1994)

Marthe^{++} simplement typé (4/4) : en pratique

En pratique, on préfère les systèmes de types décidables :

- Étant donné Γ, M et A, décider $\Gamma \vdash M : A$.
- Étant donné Γ et M, trouver A tel que $\Gamma \vdash M : A$ (souvent unique).

Dans notre cas, la décidabilité n'est pas évidente à cause des fonctions.

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \texttt{fun}\ x^{A}.\ M : A \Rightarrow B}$$

Comment y remédier ?

- On peut adopter une syntaxe où les lieurs sont annotés avec des types.
- On peut utiliser un moteur global d'inférence de types (cf. jalon 4).

[Exercice] Implémenter un typeur pour la version annotée de Marthe^{++}.

Les systèmes de types : discussion

Les systèmes de types constitue des *analyses statiques* relativement simples par rapport à d'autres techniques, ce qui n'est pas sans inconvénients.

⊖ Ils rejettent beaucoup de programmes corrects (sans erreurs).

$$\texttt{if } \underline{\mathit{true}}_{\texttt{b}} \texttt{ then } \underline{42}_{\texttt{i}} \texttt{ else } \underline{\mathit{true}}_{\texttt{b}} \times$$

⊕ Ils sont modulaires, c'est à dire respectent le *lemme de substitution*.

$$\Gamma, x : A \vdash M : B \quad \wedge \quad \Gamma \vdash N : A \quad \Rightarrow \quad \Gamma \vdash M[N/x] : B$$

⊕ Ils obéissent à des règles systématiques qui peuvent être internalisées par les programmeurs, à l'inverse de méthodes plus heuristiques.

⊕/⊖ Les types sont des invariants de la réduction, et ont donc du mal à traiter de certaines propriétés (p.ex., quantitatives).

Bilan de cette séance

On a vu aujourd'hui une méthodologie devenue standard :

- donner une sémantique à petits pas entre termes non-typés,
- formuler un système de types adéquat (substitution, préservation...),
- en déduire des propriétés sur l'exécution des programmes typés.

Le triptyque sémantique - système de types - équivalence observationelle est intéressant en théorie et en pratique.

- En théorie : liens forts avec la logique, les mathématiques en général.
- En pratique : certification de compilateur (CompCert de X. Leroy).