Projet du cours « Compilation »
Jalon 2 : Interprétation de HOPIX

version numéro Mon Jan 23 15 :33 :37 CET 2017

1 Syntaxe abstraite

Dans les sections suivantes, lorsque l'on utilisera de la syntaxe concrète dans les spécifications, on fera référence implicitement aux arbres de syntaxe abstraites sous-jacents définis par la grammaire suivante :

p	::=	$\overline{\text{d}}$	*Programme*
d	::=	`type` T $\overline{\alpha}$:= d_t	*Définition de type*
	|	`extern` x : τ	*Déclaration d'une valeur externe*
	|	d_v	*Définition de valeur(s)*
d_t	::=	$\Sigma\overline{K : \overline{\tau}}$	*Type somme*
	|	•	*Type abstrait*
d_v	::=	`val` x $\overline{\text{m}}$: $\tau^?$ `=` e	*Valeur simple*
	|	`rec` $\overline{x\ \overline{\text{m}} : \tau^? \texttt{=}\ \text{e}}$	*Valeurs mutuellement récursives*
τ	::=	T`[`$\overline{\tau}$`]`	*Type construit*
	|	α	*Variable de type*
e	::=	n	*Entier*
	|	b	*Booléen*
	|	c	*Caractère*
	|	s	*Chaîne de caractères*
	|	x	*Variable*
	|	K`[`$\overline{\tau}$`]``(`$\overline{\text{e}}$`)`	*Construction d'une donnée étiquetée*
	|	`(`e : τ`)`	*Annotation de type*
	|	d_v `;` e	*Définition locale*
	|	e`[`$\overline{\tau}$`]``(`$\overline{\text{e}}$`)`	*Application*
	|	`\`[$\overline{\alpha}$`]``(`$\overline{\text{m}}$`)` `=>` e	*Fonction anonyme*
	|	e `?` $\overline{\text{b}}$	*Analyse de motifs*
	|	`if` e `then` e `else` e	*Conditionnelle*
	|	`ref` e	*Création d'une référence*
	|	e `:=` e	*Modification d'une référence*
	|	!e	*Lecture d'une référence*
	|	`while` `(`e`)` e	*Boucle*
b	::=	m `=>` e	*Cas d'analyse*
m	::=	x	*Motif universel liant*
	|	_	*Motif universel non liant*
	|	`(`m : τ`)`	*Annotation de type*
	|	n	*Entier*
	|	b	*Booléen*
	|	c	*Caractère*
	|	s	*Chaîne de caractères*
	|	K`(`$\overline{\text{m}}$`)`	*Valeurs étiquetées*
	|	m `|` m	*Disjonction*
	|	m `&` m	*Conjonction*

Dans cette grammaire, on a utilisé les *métavariables* suivantes :
— T pour représenter un constructeur de type.
— x pour représenter un identificateur de valeur.
— K pour représenter le nom d'un constructeur de données.
— τ pour représenter un type.
— e pour représenter une expression.
— m pour représenter un motif.

— n pour représenter un litéral de type entier.
— b pour représenter un litéral de type booléen.
— c pour représenter un litéral de type caractère.
— s pour représenter un litéral de type chaîne de caractères.
— α pour représenter une variable de type.
— b pour représenter une branche d'analyse de motifs.
Par ailleurs, on a aussi écrit $\overline{X}$ pour représenter une liste potentiellement vide de X.

2 Interprétation

2.1 Valeurs

Les valeurs du langage sont définies par la grammaire suivante :

v	::=	n	*Entier*
	|	b	*Booléen*
	|	c	*Caractère*
	|	s	*Chaîne de caractères*
	|	()	*Unité*
	|	$K(\overline{v})$	*Valeur étiquetée*
	|	a	*Adresse d'une référence*
	|	$(\overline{\text{m}}$ => e$)[E]$	*Fermeture*
	|	prim	*Primitive*

Environnement d'évaluation Les identificateurs de programme sont associés à des valeurs à l'aide d'un environnement d'évaluation E. On écrit « $E[x]$ » pour parler de la valeur de x dans l'environnement E. On écrit « $E + x \mapsto v$ » pour parler de l'environnement E étendu par l'association entre l'identificateur x et la valeur v.

Références Les adresses des références sont allouées dynamiquement dans une mémoire M. On écrit « $M + a \mapsto v$ » pour représenter la mémoire qui étend la mémoire M avec une nouvelle adresse a où se trouve stockée la valeur v. On écrit « $M[a \leftarrow v]$ » pour représenter la mémoire M modifiée seulement à l'adresse a en y stockant la valeur v. Enfin, on écrit « $(a \mapsto v) \in M$ » pour indiquer que la valeur v est stockée à l'adresse a de la mémoire M.

Primitive Les primitives du langage sont définies dans le module `HopixInterpreter`. Ce sont des fonctions OCaml que l'on a rendues accessibles depuis Hopix ou bien des valeurs primitives comme **true** ou **false**.

2.2 Évaluation des programmes

Un programme p s'évalue à partir d'une mémoire vide et d'un environnement contenant les primitives du langage en évaluant successivement les définitions de valeurs dans leur ordre d'apparition dans le programme.

Pour spécifier précisément ce processus, il faut donc décrire la façon dont les définitions de valeurs s'évaluent : c'est le rôle de la section **??**. Les expressions sont les termes qui s'évaluent en des valeurs. Leur évaluation est spécifiée dans la section **??**. Elle s'appuie sur l'évaluation de l'analyse par cas (section **??**) et l'analyse de motifs (section **??**).

2.3 Évaluation des définitions

L'évaluation des définitions s'appuie sur le jugement :

$$E, M \vdash \text{d}_v \Rightarrow E', M'$$

qui se lit « Dans l'environnement E et la mémoire M, la définition d_v étend E et E' et modifie M en M'.

Cette partie de la spécification est omise volontairement. Vous devez réfléchir à une spécification raisonnable.

2.4 Évaluation des expressions

Le jugement d'évaluation des expressions s'écrit :

$$E, M \vdash \text{e} \Downarrow v, M'$$

et se lit « Dans l'environnement d'évaluation E, l'expression e s'évalue en v et change la mémoire M en M'. »

Dans la suite, les termes sont allégés de leurs types car ces derniers n'influent pas sur l'évaluation. Le jugement d'évaluation est défini par les règles suivantes :

$$\frac{}{E, M \vdash n \Downarrow n, M} \qquad \frac{}{E, M \vdash c \Downarrow c, M} \qquad \frac{}{E, M \vdash b \Downarrow b, M} \qquad \frac{}{E, M \vdash s \Downarrow s, M} \qquad \frac{E(x) = v}{E, M \vdash x \Downarrow v, M}$$

$$\frac{M_0 = M \qquad \forall i \in [1 \dots n], E, M_{i-1} \vdash \mathrm{e}_i \Downarrow v_i, M_i}{E, M \vdash K(\mathrm{e}_1, \dots, \mathrm{e}_n) \Downarrow K(v_1, \dots, v_n), M_n} \qquad \frac{E, M \vdash \mathrm{d}_v \Rightarrow E', M' \qquad E', M' \vdash \mathrm{e} \Downarrow v, M''}{E, M \vdash \mathrm{d}_v; \mathrm{e} \Downarrow v, M''}$$

$$\frac{\begin{array}{c} E, M \vdash \mathrm{e}_f \Downarrow (\mathrm{m}_1 \cdots \mathrm{m}_n \texttt{ => } \mathrm{e})[E'], M' \qquad M_0 = M' \\ \forall i \in [1 \dots n], E, M_{i-1} \vdash \mathrm{e}_i \Downarrow v_i, M_i \qquad E_0 = E' \qquad \forall i \in [1 \dots n], E_{i-1}, M_n \vdash \mathrm{m}_i \sim v_i \Uparrow E_i \qquad E_n, M_n \vdash \mathrm{e} \Downarrow v, M' \end{array}}{E, M \vdash \mathrm{e}_f\,(\mathrm{e}_1, \cdots, \mathrm{e}_n) \Downarrow v, M'}$$

$$\frac{E, M \vdash \mathrm{e} \Downarrow a, M' \qquad (a \mapsto v) \in M'}{E, M \vdash !\mathrm{e} \Downarrow v, M'} \qquad \frac{E, M \vdash \mathrm{e} \Downarrow a, M' \qquad E, M' \vdash \mathrm{e}' \Downarrow v, M''}{E, M \vdash \mathrm{e} \texttt{ := } \mathrm{e}' \Downarrow (), M''[a \mapsto v]]}$$

$$\frac{E, M \vdash \mathrm{e} \Downarrow v_s, M' \qquad E, M' \vdash v_s \sim \overline{\mathrm{b}} \Downarrow v, M''}{E, M \vdash \mathrm{e} \texttt{ ? } \overline{\mathrm{b}} \Downarrow v, M''}$$

$$\frac{E, M \vdash \mathrm{e}_b \Downarrow \mathbf{true}, M' \qquad E, M' \vdash \mathrm{e} \Downarrow (), M'' \qquad E, M'' \vdash \texttt{while } (\mathrm{e}_b)\ \mathrm{e} \Downarrow (), M'''}{E, M \vdash \texttt{while } (\mathrm{e}_b)\ \mathrm{e} \Downarrow (), M'''} \qquad \frac{E, M \vdash \mathrm{e}_b \Downarrow \mathbf{false}, M'}{E, M \vdash \texttt{while } (\mathrm{e}_b)\ \mathrm{e} \Downarrow (), M'}$$

2.5 Analyse par cas

L'analyse par cas d'une valeur v consiste à traiter une liste de cas représentés par des branches $\overline{\mathrm{b}}$ de la forme « m `=>` e » en évaluant la première de ces branches dont le motif capture la valeur v. Ce dernier mécanisme est représenté dans la section suivante.

$$\frac{E, M \vdash v \sim \mathrm{m} \Uparrow E' \qquad E', M \vdash \mathrm{e} \Downarrow v', M'}{E, M \vdash v \sim (\mathrm{m} \texttt{ => } \mathrm{e})\overline{\mathrm{b}} \Downarrow v', M'} \qquad \frac{E, M \vdash v \not\sim \mathrm{m} \qquad E, M \vdash v \sim \overline{\mathrm{b}} \Downarrow v', M'}{E, M \vdash v \sim (\mathrm{m} \texttt{ => } \mathrm{e})\overline{\mathrm{b}} \Downarrow v', M'}$$

2.6 Analyse de motifs

L'évaluation des motifs s'appuie sur le jugement :

$$E, M \vdash v \sim \mathrm{m} \Uparrow E'$$

qui se lit « Dans l'environnement E et dans la mémoire M, la valeur v est capturée par le motif m en étendant l'environnement E en E'. »

Cette partie de la spécification est omise volontairement. Vous devez réfléchir à une spécification raisonnable.

3 Travail à effectuer

La seconde partie du projet est l'écriture de l'interprète de la sémantique opérationnelle décrite plus haut.
La compilation s'effectue par la commande« `make` ».
Le projet est à rendre **avant le** :

21 janvier 2016 à 23h59

Pour finir, vous devez vous assurer des points suivants :

- Le projet contenu dans cette archive **doit compiler**.
- Vous devez **être les auteurs** de ce projet.
- Il doit être rendu **à temps**.

Si l'un de ces points n'est pas respecté, la note de 0 vous sera affectée.

4 Log