Compilation et langages

TD sémantique et types, 17 novembre 2017

Cocktail : un langage de programmation avec des glaçons

On s'intéresse à un petit langage arithmétique avec des « glaçons », c'est-à-dire une construction qui permet de retarder l'évaluation d'une expression. Ce langage, Cocktail, est une variante de Mini-ML, qui n'utilise pas de fonctions mais qui contient deux nouvelles formes d'expressions : `freeze` transforme une expression *expr* en un glaçon, qui sera évalué plus tard ; `eval` « dégèle » un glaçon, c'est-à-dire déclenche l'évaluation de l'expression *expr* qu'il contient.

Par exemple, l'évaluation de l'expression suivante affiche, dans l'ordre, `1`, `2`, `3`, `4` :

```
let x = 2 in
let y = print(1); freeze( print(x+1); x*x ) in
print(x); print(eval(y))
```

Un programme Cocktail est une expression *expr* construite avec la grammaire suivante :

expr	::=	*n*	constante entière
	|	*x*	variable
	|	`(` *expr* `)`	parenthèses
	|	*expr binop expr*	opérations arithmétiques
	|	`let` *x* `=` *expr* `in` *expr*	définition locale
	|	`print(` *expr* `)`	affichage
	|	*expr* `;` *expr*	séquence
	|	`freeze(` *expr* `)`	création d'un glaçon
	|	`eval(` *expr* `)`	évaluation d'un glaçon
binop	::=	`+` | `-` | `*` | `/`	

Question 1. Donner l'affichage et la valeur retournée par l'expression suivante :

```
let y = let x = 2 in
        print(x); freeze( let z = x+1 in print(z); z )
in
2 * eval(y)
```

Typage

Pour typer les instruction de Cocktail, on définit un système de types similaire à Mini-ML monomorphe, qui introduit les types `int`, `unit`, `int frozen` et `unit frozen`. L'objectif du système de types est de garantir par typage qu'une expression a exactement l'un de ces quatre types.

Le type `int` désigne les valeurs entières. Le type `unit` désigne les expressions ne retournant pas de valeur (comme `print`). Le type τ `frozen` désigne les glaçons renfermant une expression de type τ. Le fait de n'avoir que les types `int frozen` et `unit frozen` permet de garantir qu'une expression bien typée ne peut contenir des glaçons qui contiennent d'autres glaçons.

Le jugement de typage $\Gamma \vdash e : \tau$ signifie que l'expression e est de type τ dans l'environnement Γ. Remarque : le code de la question précédente est bien typée, de type `int`, dans un environnement vide.

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{}{\Gamma \vdash n : \texttt{int}} \qquad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash (e) : \tau} \qquad \frac{\Gamma \vdash e : \texttt{int}}{\Gamma \vdash \texttt{print}(e) : \texttt{unit}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 \ \mathit{binop} \ e_2 : \texttt{int}} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let } x \texttt{ = } e_1 \texttt{ in } e_2 : \tau_2} \qquad \frac{\Gamma \vdash e : \tau \ \texttt{frozen}}{\Gamma \vdash \texttt{eval}(e) : \tau}$$

Question 2. Définir les règles de typage pour les constructions $e_1 \,;\, e_2$ et `freeze(`e`)`.

Question 3. Les expressions suivantes sont-elles bien typées ? Si oui, donner leur type de retour.

```
let x = let y=2 in freeze(y+1) in eval(x)
```

```
let x = freeze( let y=2 in freeze(y+1) ) in eval(x)
```

```
let x = let y=2 in freeze( print(y+1) ) in eval(x)
```

Sémantique à grands pas

On va définir une sémantique à grands pas pour les programmes Cocktail. On se donne un symbole `()` qui est l'unique valeur de type `unit`, qui servira notamment dans la règle d'évaluation de l'instruction `print`.

Question 4. Donner une grammaire pour représenter l'ensemble des valeurs que peut produire un programme Cocktail.

Le jugement $e \xrightarrow{o} v$ signifie que l'expressions Cocktail close e s'évalue en la valeur v et que cette évaluation affiche, dans l'ordre, les entiers de la séquence o.

$$\frac{}{n \xrightarrow{\emptyset} n} \qquad \frac{e \xrightarrow{o} v}{(e) \xrightarrow{o} v} \qquad \frac{e \xrightarrow{o} v}{\texttt{print(}e\texttt{)} \xrightarrow{o,v} ()}$$

$$\frac{e_1 \xrightarrow{o_1} v_1 \qquad e_2[x \leftarrow v_1] \xrightarrow{o_2} v_2}{\texttt{let } x \texttt{ = } e_1 \texttt{ in } e_2 \xrightarrow{o_1,o_2} v_2} \qquad \frac{e_1 \xrightarrow{o_1} v_1 \qquad e_2 \xrightarrow{o_2} v_2}{e_1 \texttt{ ; } e_2 \xrightarrow{o_1,o_2} v_2}$$

Question 5. Donner les règles pour `+`, `freeze` et `eval`.

On veut démontrer que cette sémantique associe une valeur à tout programme Cocktail clos et bien typé, ce qu'on énonce par ce théorème :

Théorème 1. Pour toute expression e et tout type τ, si $\emptyset \vdash e : \tau$ alors il existe une valeur v et une séquence d'entiers o telles que $e \xrightarrow{o} v$.

Évidemment [1], la preuve se fait par récurrence sur la dérivation de typage justifiant le jugement $\emptyset \vdash e : \tau$. Pour s'aider dans cette preuve, on suppose acquis le lemme suivant

Lemme 1. Toute valeur de type τ `frozen` a la forme `freeze(`e`)` avec e une expression de type τ.

Question 6. Détailler les cas de la preuve par récurrence concernant les règles de typage des constructions `;`, `freeze` et `eval`.

Dans le cas de la preuve concernant la règle de typage de `let`, nous avons besoin d'un lemme liant le typage et la substitution.

Question 7. Énoncer ce lemme.

1. Vous étiez au cours ce matin, n'est-ce pas ?