Compilateur PtiCaml

deuxième partie : analyse sémantique

à rendre avant le 10 novembre 10 heures

1 Analyse sémantique

Cette deuxième partie consiste à vérifier la conformité du fichier source, une fois l'analyse syntaxique effectuée avec succès.

1.1 Types et environnements de typage

Dans tout ce qui suit, les expressions de types sont de la forme suivante :

$$\tau ::= \texttt{unit} \mid \texttt{string} \mid \texttt{bool} \mid \texttt{int} \mid \texttt{float} \mid \alpha \mid \tau \to \tau \mid \tau \times \cdots \times \tau \mid \tau\ \texttt{list}$$

Un environnement de typage Γ est une suite de déclarations de la forme $x : \forall\alpha_1, \ldots, \alpha_n.\tau$ où $\forall\alpha_1 \ldots \alpha_n.\tau$ désigne un *schéma* de type, *i.e.* un type τ où les variables α_i sont généralisées. Un schéma de type peut être réduit à un type si $n = 0$. Par soucis de concision, on utilisera la notation $\bar{\alpha}$ pour désigner un ensemble de variables $\{\alpha_1, \ldots, \alpha_n\}$ et de même, on pourra écrire $\forall\bar{\alpha}.\tau$ pour le schéma $\forall\alpha_1, \ldots, \alpha_n.\tau$. L'environnement Γ peut être vu comme une fonction partielle et on notera $\Gamma(x)$ le schéma de type associé à x, s'il existe. On note $\texttt{inst}(\forall\alpha_1, \ldots, \alpha_n.\tau)$ l'ensemble des types obtenus en substituant dans τ les variables α_i par des types quelconques.

Dans la suite, on utilisera la notation $\Gamma \oplus p : \forall\bar{\alpha}.\tau$ définie récursivement de la manière suivante :

$$\begin{aligned} \Gamma \oplus \_ : \forall\bar{\alpha}.\tau &= \Gamma \\ \Gamma \oplus x : \forall\bar{\alpha}.\tau &= \Gamma, x : \forall\bar{\alpha}.\tau \\ \Gamma \oplus (p_1, \ldots, p_n) : \forall\bar{\alpha}.\tau_1 \times \cdots \times \tau_n &= \Gamma \oplus p_1 : \forall\bar{\alpha}.\tau_1 \oplus \cdots \oplus p_n : \forall\bar{\alpha}.\tau_n \end{aligned}$$

Enfin, si $\texttt{vars}(\tau)$ représente l'ensemble des variables apparaissant dans le type τ, on note $\texttt{fv}(\forall\alpha_1 \ldots \alpha_n.\tau)$ l'ensemble de variables défini par $\texttt{vars}(\tau) \setminus \{\alpha_1, \ldots, \alpha_n\}$

1.2 Typage des expressions

On introduit le jugement $\Gamma \vdash e : \tau$ signifiant « dans l'environnement Γ, l'expression e est bien typée de type τ » et on le définit par les règles d'inférence suivantes :

$$\frac{c \text{ constante de type } \tau}{\Gamma \vdash c : \tau} \qquad \frac{\tau \in \texttt{inst}(\forall\alpha.\alpha\ \texttt{list})}{\Gamma \vdash \texttt{[]} : \tau} \qquad \frac{\tau \in \texttt{inst}(\Gamma(x))}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma \vdash e : \texttt{int}}{\Gamma \vdash \texttt{-}\ e : \tau} \qquad \frac{\Gamma \vdash e : \texttt{float}}{\Gamma \vdash \texttt{-.}\ e : \tau} \qquad \frac{\Gamma \vdash e : \texttt{bool}}{\Gamma \vdash \texttt{not}\ e : \texttt{bool}}$$

$$\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau \quad op \in \{\texttt{=},\ \texttt{<>}, <=, >=, <, >\}}{\Gamma \vdash e_1\ op\ e_2 : \texttt{bool}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int} \quad op \in \{\texttt{+},\ \texttt{-},\ \texttt{*}, /\}}{\Gamma \vdash e_1\ op\ e_2 : \texttt{int}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{float} \quad \Gamma \vdash e_2 : \texttt{float} \quad op \in \{\texttt{+.},\ \texttt{-.},\ \texttt{*.}, /.\}}{\Gamma \vdash e_1\ op\ e_2 : \texttt{float}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \texttt{bool} \quad op \in \{\texttt{\&\&},\ \texttt{||}\}}{\Gamma \vdash e_1\ op\ e_2 : \texttt{bool}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau\ \texttt{list}}{\Gamma \vdash e_1 :: e_2 : \tau} \qquad \frac{\Gamma \vdash e_i : \tau_i}{\Gamma \vdash (e_1, \ldots, e_n) : \tau_1 \times \cdots \times \tau_n}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}$$

$$\frac{\Gamma \vdash e_1 : \tau'\ \texttt{list} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \oplus p_1 : \tau' \oplus p_2 : \tau'\ \texttt{list} \vdash e_3 : \tau}{\Gamma \vdash \texttt{match}\ e_1\ \texttt{with}\ [] \to e_2 \mid p_1 :: p_2 \to e_3 : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \oplus p : \forall \bar{\alpha}.\tau_1 \vdash e_2 : \tau_2 \quad \bar{\alpha} = \texttt{fv}(\tau_1) \setminus \texttt{fv}(\Gamma)}{\Gamma \vdash \texttt{let}\ p{=}e_1\ \texttt{in}\ e_2 : \tau_2}$$

$$\frac{\Gamma, x : \tau \vdash e_1 : \tau \quad \Gamma, x : \forall \bar{\alpha}.\tau \vdash e_2 : \tau_3 \quad \bar{\alpha} = \texttt{fv}(\tau) \setminus \texttt{fv}(\Gamma) \quad \tau = \tau_1 \to \tau_2}{\Gamma \vdash \texttt{let rec}\ x{=}e_1\ \texttt{in}\ e_2 : \tau_3}$$

$$\frac{\Gamma \oplus p : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \texttt{function}\ p \to e : \tau_1 \to \tau_2}$$

En plus de ces règles de typage, vous devez vérifier que pour chaque construction impliquant un motif aucune variable n'apparaît deux fois dans ce motif. Plus précisément, vous devez vérifier l'unicité des variables dans

- le motif p dans `function` $p \to \ldots$
- le motif p dans `let` p `=` $\ldots$
- la paire de motifs p_1,p_2 dans `match` e_1 `with` $[] \to \ldots \mid p_1 :: p_2 \to \ldots$

Enfin, dans la règle de typage de l'expression `let rec` $x{=}e_1$ `in` e_2, on vérifiera que l'expression e_1 est de la forme `function` $p \to e$.

1.3 Typage des déclarations

On introduit le jugement $\Gamma_1 \vdash_d d \Rightarrow \Gamma_2$ pour le typage des déclarations. On le définit par les règles d'inférence suivantes :

$$\frac{\Gamma \vdash e : \tau \quad \bar{\alpha} = \texttt{fv}(\tau) \setminus \texttt{fv}(\Gamma)}{\Gamma \vdash_d \texttt{let}\ p{=}e \Rightarrow \Gamma \oplus p : \forall \bar{\alpha}.\tau}$$

$$\frac{\Gamma, x : \tau \vdash e : \tau \quad \bar{\alpha} = \texttt{fv}(\tau) \setminus \texttt{fv}(\Gamma) \quad \tau = \tau_1 \to \tau_2}{\Gamma \vdash_d \texttt{let rec}\ x{=}e \Rightarrow \Gamma, x : \forall \bar{\alpha}.\tau}$$

De même que pour le typage des expressions, on vérifiera dans la déclaration `let rec` $x{=}e$ que e est de la forme `function` $p \to e'$.

1.4 Typage des fichiers

On introduit finalement le jugement $\Gamma \vdash_f d_1 \cdots d_n$ signifiant « dans l'environnement Γ le fichier constitué par la suite de déclarations d_1, ..., d_n est bien formé ». Le typage d'un fichier consiste à typer successivement les déclarations dans le contexte étendu par chaque nouvelle déclaration, d'où les règles :

$$\frac{}{\Gamma \vdash_f \emptyset} \qquad \frac{\Gamma \vdash_d d_1 \Rightarrow \Gamma_1 \quad \Gamma_1 \vdash_f d_2 \cdots d_n}{\Gamma \vdash_f d_1\ d_2 \cdots d_n}$$

1.5 Fonctions prédéfinies

Le langage PtiCaml contient un ensemble de primitives qui devront être connues lors de la phases d'analyse sémantique. La liste de ces primitives avec leur type est la suivante :

`print_int` : `int` → `unit`
`print_float` : `float` → `unit`
`print_string` : `string` → `unit`
`print_newline` : `unit` → `unit`
`read_int` : `unit` → `int`
`read_float` : `unit` → `float`
`read_line` : `unit` → `string`
`sqrt` : `float` → `float`
`int_of_float` : `float` → `int`
`float_of_int` : `int` → `float`
`open_screen` : `int` × `int` → `unit`
`clear_screen` : `unit` → `unit`
`draw_line` : `int` × `int` × `int` × `int` → `unit`
`set_color` : `int` × `int` × `int` → `unit`

Toutes ces primitives sont dans la bibliothèque standard d'Ocaml sauf les 4 dernières. Afin de vérifier vos programmes de test avec le compilateur `ocamlc`, vous devez utiliser la bibliothèque graphique `graphics.cma` et inclure à vos fichiers le prélude suivant :

```
open Graphics
let open_screen (x, y) = open_graph (Printf.sprintf " %dx%d" x y)
let clear_screen = clear_graph
let draw_line (x1, y1, x2, y2) =  moveto x1 y1; lineto x2 y2
let set_color (r, g, b) = set_color (rgb r g b)
```

2 Travail demandé

Adapter votre compilateur pour qu'il effectue la passe d'analyse sémantique, à l'issue de l'analyse syntaxique. En plus du fichier PtiCaml en argument, votre compilateur devra reconnaître désormais deux options, `-parse-only` et `-type-only`. La première spécifie de s'arrêter juste après l'analyse syntaxique. La seconde est sans effet dans cette deuxième partie et son utilisation sera justifiée par la suite.

En plus des erreurs lexicales et syntaxiques, votre compilateur devra maintenant afficher les erreurs sémantiques et, le cas échéant, terminer avec le code de sortie 1 (`exit 1`). Les erreurs sémantiques seront signalées de la même manière que les erreurs lexicales et syntaxiques ; par exemple :

```
File "test.ml", line 21, characters 13-14:
this expression has type int but is expected to have type string
```

Le comportement en cas d'absence d'erreur ou d'erreur du compilateur ne change pas.

Modalités de remise de votre projet. Votre projet doit se présenter sous forme d'une archive tar compressée (option "z" de tar), appelée *vos_noms*`.tgz` qui doit contenir un répertoire appelé *vos_noms* (exemple : `dupont-durand.tgz`). Dans ce répertoire doivent se trouver les *sources* de votre programme (ne donnez pas les fichiers compilés). Quand on se place dans ce répertoire, la commande `make` doit créer votre compilateur, qui sera appelé `compilo`. La commande `make clean` doit effacer tous les fichiers que `make` a engendré et ne laisser dans le répertoire que les fichiers sources.

Votre projet est à rendre à votre chargé de TP par e-mail[1] au plus tard le 10 novembre à 10h00.

[1] `Sylvain.Conchon@lri.fr`, `Louis.Mandel@lri.fr`, `Romain.Bardou@lri.fr`