Compilateur Pascal avec filtrage
deuxième partie : analyse sémantique

à rendre avant le lundi 14 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 :

$$\tau ::= \texttt{boolean} \mid \texttt{integer} \mid \texttt{real} \mid \texttt{array[}n_1\texttt{..}n_2\texttt{] of } \tau \mid id$$

où id désigne un identificateur de type récursif et n_1 et n_2 deux constantes entières. Si τ_1 et τ_2 désignent deux types arithmétiques, on note $\tau_1 \sqcup \tau_2$ le plus grand des deux, c'est-à-dire `real` si l'un des deux est `real`, et `integer` sinon. D'autre part, on note $\tau_1 \sqsubseteq \tau_2$ si $\tau_1 = \tau_2$ ou $\tau_1 = \texttt{integer}$ et $\tau_2 = \texttt{real}$. Enfin, on note $\tau_1 \equiv \tau_2$ si τ_1 et τ_2 ne sont pas des types de tableaux et sont tels que $\tau_1 \sqsubseteq \tau_2$ ou $\tau_2 \sqsubseteq \tau_1$. Enfin, on note $\tau \neq \texttt{array}$ pour signifier que le type τ n'est pas un type de tableau.

Un environnement de typage Γ est une suite de déclarations de la forme `type` id pour un type, $C(\tau_1, \dots, \tau_n) : id$ pour un constructeur C du type id, $x : \tau$ pour une variable, $f(m_1\ \tau_1, \dots, m_n\ \tau_n) : \tau$ pour une fonction ou $p(m_1\ \tau_1, \dots, m_n\ \tau_n)$ pour une procédure (où les m_i désignent les spécificateurs optionnels `var`).

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 le jugement $\Gamma \vdash_l e : \tau$ signifiant « dans l'environnement Γ, l'expression e est une valeur gauche bien typée de type τ ». Ces jugements sont alors définis par les règles d'inférence suivantes :

$$\frac{c \text{ constante de type } \tau}{\Gamma \vdash c : \tau} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash_l x : \tau} \qquad \frac{\Gamma \vdash_l e : \tau}{\Gamma \vdash e : \tau}$$

$$\frac{\Gamma \vdash_l e_1 : \texttt{array[}n_1\texttt{..}n_2\texttt{] of } \tau \quad \Gamma \vdash e_2 : \texttt{integer}}{\Gamma \vdash_l e_1\texttt{[}e_2\texttt{]} : \tau}$$

$$\frac{\Gamma \vdash e : \tau \quad \tau \in \{\texttt{integer, real}\} \quad op \in \{\texttt{+, -}\}}{\Gamma \vdash op\ e : \tau} \qquad \frac{\Gamma \vdash e : \texttt{boolean}}{\Gamma \vdash \texttt{not } e : \texttt{boolean}}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \quad \tau_1 \equiv \tau_2 \quad op \in \{\texttt{=, <>}\}}{\Gamma \vdash e_1\ op\ e_2 : \texttt{boolean}}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \quad \tau_1, \tau_2 \in \{\texttt{boolean},\ \texttt{integer},\ \texttt{real}\} \quad \tau_1 \equiv \tau_2 \quad op \in \{\texttt{<},\ \texttt{<=},\ \texttt{>},\ \texttt{>=}\}}{\Gamma \vdash e_1\ op\ e_2 : \texttt{boolean}}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \quad \tau_1, \tau_2 \in \{\texttt{integer},\ \texttt{real}\} \quad op \in \{\texttt{+},\ \texttt{-},\ \texttt{*}\}}{\Gamma \vdash e_1\ op\ e_2 : \tau_1 \sqcup \tau_2}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \quad \tau_1, \tau_2 \in \{\texttt{integer},\ \texttt{real}\}}{\Gamma \vdash e_1\ \texttt{/}\ e_2 : \texttt{real}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{integer} \quad \Gamma \vdash e_2 : \texttt{integer} \qquad op \in \{\texttt{div},\ \texttt{mod}\}}{\Gamma \vdash e_1\ op\ e_2 : \texttt{integer}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{boolean} \quad \Gamma \vdash e_2 : \texttt{boolean} \quad op \in \{\texttt{and},\ \texttt{or}\}}{\Gamma \vdash e_1\ op\ e_2 : \texttt{boolean}}$$

$$\frac{f(m_1\ \tau'_1, \dots, m_n\ \tau'_n) : \tau \in \Gamma \quad (\Gamma \vdash_l e_i : \tau_i \text{ si } m_i = \texttt{var},\ \Gamma \vdash e_i : \tau_i \text{ sinon}) \quad \tau_i \sqsubseteq \tau'_i}{\Gamma \vdash f(e_1, \dots, e_n) : \tau}$$

$$\frac{C(\tau'_1, \dots, \tau'_n) : id \in \Gamma \quad \Gamma \vdash e_i : \tau_i \quad \tau_i \sqsubseteq \tau'_i}{\Gamma \vdash C(e_1, \dots, e_n) : id}$$

Note : les deux dernières règles s'appliquent également dans le cas où il n'y a aucun argument ($n = 0$).

1.3 Typage des motifs

On introduit le jugement $\Gamma \vdash p \xrightarrow{\tau} \Gamma'$ signifiant « dans l'environnement Γ, le motif p est bien formé pour le type τ et définit le nouvel environnement Γ' ». Ce jugement est établi par les règles d'inférence suivantes :

$$\frac{c \text{ constante de type } \tau}{\Gamma \vdash c \xrightarrow{\tau} \Gamma} \qquad \frac{}{\Gamma \vdash \_ \xrightarrow{\tau} \Gamma} \qquad \frac{}{\Gamma \vdash x \xrightarrow{\tau} \Gamma, x : \tau}$$

$$\frac{C(\tau_1, \dots, \tau_n) : id \in \Gamma \quad \Gamma \vdash p_1 \xrightarrow{\tau_1} \Gamma_1 \quad \dots \quad \Gamma_{n-1} \vdash p_n \xrightarrow{\tau_n} \Gamma_n}{\Gamma \vdash C(p_1, \dots, p_n) \xrightarrow{id} \Gamma_n}$$

Note : on peut avoir $n = 0$ dans cette dernière règle, ce qui correspond au cas d'un constructeur constant, qu'il convient de ne pas confondre avec le cas d'une variable.

1.4 Typage des instructions

On introduit le jugement $\Gamma \Vdash i$ signifiant « dans l'environnement Γ, l'instruction i est bien typée ». Ce jugement est établi par les règles d'inférence suivantes :

$$\frac{\Gamma \vdash_l l : \tau \quad \tau \neq \texttt{array} \quad \Gamma \vdash e : \tau' \quad \tau' \sqsubseteq \tau}{\Gamma \Vdash l\ \ \texttt{:=}\ e}$$

$$\frac{p(m_1\ \tau'_1, \dots, m_n\ \tau'_n) \in \Gamma \quad (\Gamma \vdash_l e_i : \tau_i \text{ si } m_i = \texttt{var},\ \Gamma \vdash e_i : \tau_i \text{ sinon}) \quad \tau_i \sqsubseteq \tau'_i}{\Gamma \Vdash p(e_1, \dots, e_n)}$$

$$\frac{\Gamma \vdash e_i : \tau_i \quad \tau_i \in \{\texttt{boolean},\ \texttt{integer},\ \texttt{real}\}}{\Gamma \Vdash \texttt{writeln}(e_1, \dots, e_n)}$$

$$\frac{\Gamma \vdash e : \texttt{boolean} \quad \Gamma \Vdash i_1}{\Gamma \Vdash \texttt{if } e \texttt{ then } i_1} \qquad \frac{\Gamma \vdash e : \texttt{boolean} \quad \Gamma \Vdash i_1 \quad \Gamma \Vdash i_2}{\Gamma \Vdash \texttt{if } e \texttt{ then } i_1 \texttt{ else } i_2}$$

$$\frac{\Gamma \vdash e : \texttt{boolean} \quad \Gamma \Vdash i}{\Gamma \Vdash \texttt{while } e \texttt{ do } i}$$

$$\frac{x : \texttt{integer} \in \Gamma \quad \Gamma \vdash e_1 : \texttt{integer} \quad \Gamma \vdash e_2 : \texttt{integer} \quad \Gamma \Vdash i}{\Gamma \Vdash \texttt{for } x \texttt{ := } e_1 \texttt{ to } e_2 \texttt{ do } i}$$

$$\frac{\Gamma \Vdash i_j}{\Gamma \Vdash \texttt{begin } i_1\texttt{; } \dots\texttt{; } i_n \texttt{ end}}$$

$$\frac{\Gamma \vdash e : \tau \quad \tau \neq \texttt{array} \quad \Gamma \vdash p_j \xrightarrow{\tau} \Gamma_j \quad \Gamma_j \Vdash i_j}{\Gamma \Vdash \texttt{case } e \texttt{ of } p_1 \texttt{ : } i_1 \texttt{ | } \dots \texttt{ | } p_n \texttt{ : } i_n \texttt{ end}}$$

Concernant la dernière règle, on demande de vérifier également le caractère *exhaustif* du filtrage, i.e. le fait que toute valeur possible pour e est bien filtrée par *au moins* un motif. L'algorithme sera étudié en TD. En revanche, on ne demande pas de détecter les clauses inutiles (i.e. les motifs déjà couverts par des motifs précédents).

Enfin, si l'instruction à typer est le corps d'une fonction f, on ajoute la règle suivante :

$$\frac{f(m_1\ x_1, \dots, m_n\ x_n) : \tau \in \Gamma \quad \Gamma \vdash e : \tau' \quad \tau' \sqsubseteq \tau}{\Gamma \Vdash f \texttt{ := } e}$$

1.5 Typage des fichiers

On dit qu'un type τ est *bien formé* dans un environnement Γ, et on note $\Gamma \vdash \tau\ \mathsf{bf}$, si tous les identificateurs de types apparaissant dans τ correspondent à des types déclarés dans Γ. Cela correspond aux règles d'inférence suivantes :

$$\frac{}{\Gamma \vdash \texttt{boolean}\ \mathsf{bf}} \qquad \frac{}{\Gamma \vdash \texttt{integer}\ \mathsf{bf}} \qquad \frac{}{\Gamma \vdash \texttt{real}\ \mathsf{bf}}$$

$$\frac{n_1 \leq n_2 \quad \Gamma \vdash \tau\ \mathsf{bf}}{\Gamma \vdash \texttt{array[}n_1\texttt{..}n_2\texttt{] of } \tau\ \mathsf{bf}} \qquad \frac{\texttt{type } id \in \Gamma}{\Gamma \vdash id\ \mathsf{bf}}$$

On introduit ensuite le jugement $\Gamma \vdash d \to \Gamma'$ signifiant « dans l'environnement Γ, la suite de déclarations d est bien formée et définit le nouvel environnement Γ' ».

Déclarations de types. On commence par définir le jugement $\Gamma \vdash d \to \Gamma'$ pour les déclarations de types construits :

$$\frac{\Gamma, \texttt{type } id \vdash \tau_{i,j}\ \mathsf{bf} \quad \tau_{i,j} \neq \texttt{array}}{\begin{array}{llll} \Gamma \vdash \texttt{type } id & \texttt{=} & C_1 \texttt{ of } \tau_{1,1} \texttt{ * } \cdots \texttt{ * } \tau_{1,n_1} & \to \Gamma, \texttt{type } id,\ C_1(\tau_{1,1}, \dots, \tau_{1,n_1}) : id, \\ & & \vdots & \qquad \ddots, \\ & \texttt{|} & C_k \texttt{ of } \tau_{k,1} \texttt{ * } \cdots \texttt{ * } \tau_{k,n_k} & \qquad C_k(\tau_{k,1}, \dots, \tau_{k,n_k}) : id \end{array}}$$

Déclarations de variables. On étend le jugement $\Gamma \vdash d \to \Gamma'$ aux déclarations de variables de la manière suivante :

$$\frac{\Gamma \vdash \tau\ \mathsf{bf}}{\Gamma \vdash x : \tau \to \Gamma, x : \tau}$$

Déclarations de fonctions. On étend le jugement $\Gamma \vdash d \rightarrow \Gamma'$ aux déclarations de fonctions de la manière suivante :

$$\frac{\begin{array}{c}\Gamma \vdash \tau \;\mathsf{bf} \quad \tau \neq \mathtt{array} \quad \Gamma \vdash \tau_i \;\mathsf{bf} \quad \Gamma \vdash d \rightarrow \Gamma' \\ \Gamma', f(m_1\ \tau_1, \dots, m_n\ \tau_n) : \tau, x_1 : \tau_1, \dots, x_n : \tau_n \Vdash b\end{array}}{\Gamma \vdash f(m_1\ x_1 : \tau_1, \dots, m_n\ x_n : \tau_n) : \tau; \mathtt{var}\ d\ b \rightarrow \Gamma, f(m_1\ \tau_1, \dots, m_n\ \tau_n) : \tau}$$

On vérifiera d'autre part que tous les flots de contrôle du corps b de la fonction contiennent au moins une instruction positionnant la valeur de retour (i.e. affectant la variable f).

Le typage d'une procédure est similaire :

$$\frac{\Gamma \vdash \tau_i \;\mathsf{bf} \quad \Gamma \vdash d \rightarrow \Gamma' \quad \Gamma', p(m_1\ \tau_1, \dots, m_n\ \tau_n), x_1 : \tau_1, \dots, x_n : \tau_n \Vdash b}{\Gamma \vdash p(m_1\ x_1 : \tau_1, \dots, m_n\ x_n : \tau_n); \mathtt{var}\ d\ b \rightarrow \Gamma, p(m_1\ \tau_1, \dots, m_n\ \tau_n)}$$

Fichiers. On introduit finalement le jugement $\Gamma \vdash_f d_1 \cdots d_n\ i$ signifiant « dans l'environnement Γ le fichier constitué par la suite de déclarations $d_1, \dots, d_n$ et l'instruction i 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 :

$$\frac{\Gamma \Vdash i}{\Gamma \vdash_f \emptyset\ i} \qquad \frac{\Gamma \vdash d_1 \rightarrow \Gamma' \quad \Gamma' \vdash_f d_2 \cdots d_n\ i}{\Gamma \vdash_f d_1\ d_2 \cdots d_n\ i}$$

Règles d'unicité. Enfin, on vérifiera l'unicité :
- des symboles globaux (types, constructeurs, variables *globales*, fonctions et procédures) sur l'ensemble du fichier ;
- des symboles locaux (paramètres formels, variables locales et variables de motifs) dans chaque flot de contrôle de chaque fonction ou procédure.

On notera qu'un symbole local peut porter le même nom qu'un symbole global, à une exception près : un symbole local à l'intérieur d'une fonction ne peut porter le même nom que cette fonction.

Fonctions prédéfinies. Les fonctions suivantes sont supposées prédéfinies et devront être connues à l'analyse sémantique :

```
trunc(x : real) : integer;
cos(r : real) : real;
sin(r : real) : real;
```

de même que les procédures suivantes :

```
open_screen(w : integer; h : integer);
clear_screen;
set_color(r : integer; g : integer; b : integer);
draw_point(x : integer; y : integer);
draw_line(x1 : integer; y1 : integer; x2 : integer; y2 : integer);
draw_circle(x : integer; y : integer; r : integer);
draw_rect(x : integer; y : integer; w : integer; h : integer);
fill_rect(x : integer; y : integer; w : integer; h : integer);
usleep(n : integer);
```

1.6 Indications

Tests. En cas de doute concernant un point de sémantique, vous pouvez utiliser un compilateur Pascal comme référence (sur le fragment ne comprenant pas l'extension de filtrage). Vous pouvez d'ailleurs vous inspirer de ses messages d'erreur pour votre compilateur (en les traduisant ou non en français).

Anticipation. Dans la troisième partie (production de code), certaines informations provenant du typage seront nécessaires, telles que par exemple la nécessité de transformer une valeur entière en une valeur réelle, etc. Il vous est conseillé d'anticiper ces besoins en programmant cette deuxième partie. Deux méthodes (au moins) sont envisageables :
- décorer votre arbre de syntaxe abstraite par des informations de typage (telle que le type de chaque sous-expression) ;
- construire un arbre de syntaxe abstraite plus précis.

Notez que dans les deux cas, cela signifie que vos fonctions de typage ne se contentent pas de parcourir les arbres de syntaxe abstraite mais en retournent de nouveaux.

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 Pascal 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.pas", line 21, characters 13-14:
this expression has type integer but is expected to have type boolean
```

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

Votre code devra comprendre au moins un fichier `typing.ml` contenant les principales fonctions d'analyse sémantique. L'introduction d'autres fichiers devra être justifiée dans le rapport.

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 doit également s'accompagner d'un rapport (5 pages maximum) expliquant les difficultés rencontrées et les solutions techniques adoptées. Le rapport pourra être rendu soit directement dans l'archive contenant les sources de votre projet (formats pdf, postscript ou openoffice), soit imprimé et remis au secrétariat du master M1.

Votre projet est à rendre à votre chargé de TP par e-mail[1] au plus tard le **lundi 14 novembre à 10h00**.

[1] `sylvain.conchon@lri.fr`, `jean-christophe.filliatre@lri.fr`, `marc.pouzet@lri.fr`, `matthieu.sozeau@lri.fr`, `benjamin.monate@cea.fr`