Compilation — Contrôle continu

M1 informatique

28 juin 2023

Documents : interdits

Durée : 1h

Barème : /20 donné à titre indicatif

Enseignant : Jules Chouquet

Préambule : Définition du langage

On considère le langage `programming with numbers and letters` (`pnl`) dont les seules valeurs sont les nombres et les lettres de l'alphabet. Les conditions pour les boucles et conditionnelles seront implémentées avec des entiers (considérés comme vrais si différents de 0).

Voici sa syntaxe :

```
exp : Letter
    | Int
    | 'toLetter' '(' exp ')'
    | 'toInt' '(' exp ')'
    | exp '+' exp
    | 'lower' '(' exp ')'
    | 'upper' '(' exp ')'
    | 'read' '(' ')'

Letter :  \'[a-zA-Z]\'
Id :  [a-zA-Z][a-zA-Z0-9_-]
```

```
inst : 'print' '(' exp ')'
     | 'if' '(' exp ')' inst
     | 'while' '(' exp ')' inst
     | Type Id
     | Id '=' exp
     | '{' (inst ';')* '}'

Type :  'int' | 'letter'
Int :  [0-9]+
prog :  inst
```

Exercice 1 : Questions

Question 1 (/4)

1. L'opérateur `'+'` ne doit être utilisé qu'avec des expressions de type `int`. Sachant cela, peut-on remplacer la ligne `exp '+' exp` par `Int '+' Int` ?

   Pourquoi ? Qu'est-ce que cela change ?

2. Donnez la représentation du programme suivant sous forme d'AST :
   `while(lower('a')){ if (x+1) print('A'+2); int x; z=toInt(x);}`

Exercice 2 : Typage formel

Pour rappel, les règles de dérivation de typage prennent la forme suivante, où Γ est un environnement de typage quelconque, c'est-à-dire une association des variables aux types. Voici quelques exemples pour `pnl` (dont les types nécessaires sont `int,letter,inst`) :

$$\frac{}{\Gamma, x : T \vdash x : T}\ \text{var} \qquad \frac{\Gamma, x : T \vdash p : \texttt{inst}}{\Gamma \vdash \{T x; p;\} : \texttt{inst}}\ \text{decl}$$

$$\frac{\Gamma \vdash m : \texttt{int} \qquad \Gamma \vdash n : \texttt{int}}{\Gamma \vdash m + n : \texttt{int}}\ \texttt{plus} \qquad \frac{\Gamma \vdash m : \texttt{int}}{\Gamma \vdash \texttt{print}(m) : \texttt{inst}}\ \text{print}$$

Question 1 (/3) Écrivez la dérivation de typage[1]du programme suivant : `{int x; int y; print(x+y);}`

Question 2 (/7) Écrivez des règles de typage cohérentes pour les constructions syntaxiques suivantes, avec les contraintes associées. (On considèrera comme en cours qu'il n'y a aucune structure de bloc dans les programmes et que l'on a toujours un seul environnement) :

1. `toInt` et `toLetter`, qui sont des opérateurs de conversion entre entiers et lettres.
2. `read`, qui lit une lettre sur l'entrée standard du programme.
3. `if`, sachant que les conditions sont représentées par des entiers
4. `affectation`
5. `upper` qui renvoie une lettre en majuscule à partir d'une lettre en minuscule.

Exercice 3 : Vérification du typage, visiteur et extension

On suppose que les classes de l'AST de `pnl` sont données : `Node`,`Exp`,`ExpLetter`, `ExpInt`,`ExpToLetter`,..., `Inst`,`InstPrint`,`InstIf`,...,`InstList` et conçues comme en cours et dans les TP. De même, on suppose que l'interface visiteur `Visitor` est donnée, et que toutes les classes de l'AST disposent d'une méthode `accept`.

[1] Attention, dans une dérivation de typage, il n'y a plus Γ, mais uniquement l'environnement nécessaire à la dérivation. Et si le programme ne contient aucune variable non déclarée, l'environnement doit être vide en conclusion de la dérivation.

Chaînes de caractères (/6) On souhaite rajouter des chaînes de caractère au langage.

1. Indiquez les modifications à faire dans la grammaire, sachant qu'on souhaite également un opérateur `add` qui permette d'ajouter une lettre à la fin d'une chaîne de caractères. (par exemple avec la syntaxe `add("baba",'r')`).

2. Supposons qu'on dispose d'une classe `TypeChecker` implémentant l'interface `Visitor<Type>`, dont les méthodes renvoient le type de chaque nœud et renvoient une erreur en cas de mauvais typage.

   Écrivez en Java la méthode de visite correspondant à l'opérateur `add` qu'il faudrait rajouter dans cette classe. (On suppose que vous avez accès à des méthodes permettant de signaler les erreurs, c'est-à-dire, comme en TP, une instance de classe `Error` avec une méthode add(Node n, String s).