IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

TP16 – Compilation

Le typage et l'exécution

Ingénierie du développement logiciel – F2B304

Correction

1 Introduction

Dans les séances précédentes, nous avons découvert les phases dites d'*analyse lexicale* et d'*analyse syntaxique* qui transforme un flux de caractère en un arbre syntaxique abstrait. Le but de ce TP est de découvrir une partie des étapes suivantes d'un compilateur. Plus précisément, nous allons réaliser un *typeur* qui va parcourir l'arbre syntaxique abstrait pour : (1) éliminer les programmes incorrects et (2) enrichir l'AST avec les informations de type de tous ses nœuds. Nous allons également réaliser un *évaluateur* qui évaluera cet AST typé pour calculer les valeurs du programme.

La figure 1 ci-dessous représente cet enchainement et l'illustre avec un exemple en rouge.

Contrairement aux phases précédentes qui utilisaient des langages spécialisés, le typage et l'exécution sera réalisée en OCaml. En effet, il s'agira de programmer des fonctions qui vont réaliser ces opérations.

2 Contexte de la séance

Pour illustrer simplement les notions de la séance, nous allons travailler sur un langage d'expression mélangeant du calcul arithmétique sur des entiers, des comparaisons entre entiers et des calculs sur les booléens.

Soit le langage suivant :

⟨*start*⟩ : := (⟨*expr*⟩ `EOL`)+ `EOF`

⟨*expr*⟩ : := '(' ⟨*expr*⟩ ')' | `IDENT` | `INT` | 'true' | 'false' | ⟨*unop*⟩ ⟨*expr*⟩ | ⟨*expr*⟩ ⟨*binop*⟩ ⟨*expr*⟩

⟨*unop*⟩ : := '-' | '!'

⟨*binop*⟩ : := '<' | '<=' | '>' | '>=' | '!=' | '==' | '+' | '-' | '*' | '/' | '%' | '&&' | '||'

Le code de son arbre syntaxique abstrait, de ses analyseurs lexicale et syntaxique ainsi qu'un petit programme de test vous sont fournis.

Vous pouvez passer un peu de temps à le découvrir et l'exécuter !

3 L'exécution de programme

Nous allons commencer par l'évaluateur, même si c'est inhabituel, cela permettra de découvrir les différentes erreurs d'exécution et ainsi d'en déduire les types qui seront nécessaires.

La sémantique des différentes opérations est normalement connue.

Exercice 1 (*Exécution des expressions*)

▷ **Proposer et implanter un système d'execution pour notre langage d'expression.**

Voir dans le repertoire exo1.

4 La vérification de type

Comme présenté dans le cours langages, les types sont des ensembles de valeurs munis d'opérations. Ils vont avoir, entre autres, deux utilités : (a) s'assurer que le programme respecte des règles de bonne construction et (b) préparer les phases suivantes [1].

Un système de type consiste donc en :

- des types
  - prédéfinis
  - définis par l'utilisateur dans son programme
- des règles que doivent respecter les programmes concernant
  - l'usage des noms (validité, portée, accessibilité ...)
  - la bonne formation des expressions et instructions
  - l'usage des variables (affectation ou usage correct)
  - la correction globale (par exemple, l'héritage simple, la non surcharge, ...)

Exercice 2 (*Typage des expressions*)

Dans l'exercice précédent, un certain nombres de cas d'erreur peuvent se produire.

▷ **Proposer et implanter un système de type pour notre langage d'expression qui empêche ses erreurs d'exécution.**

Voir dans le repertoire exo2.

1. Dans cette séance nous nous limiterons à une exécution simple pour laquelle les types ne sont pas indispensables mais le cours d'optimisation du 21 janvier vous montrera des phases qui ont besoin de cette information.

5 Pour aller plus loin

Exercice 3 (*Des définitions*)

▷ **Proposer une extension du langage pour permettre la définition des variables dans l'expression [2]. Implanter les différents éléments nécessaires à exécuter ces nouveaux programmes.**

Voir dans le repertoire exo3.

2. Par exemple, on peut imaginer ajouter une construction `let` v `=` e_1 `in` e_2.

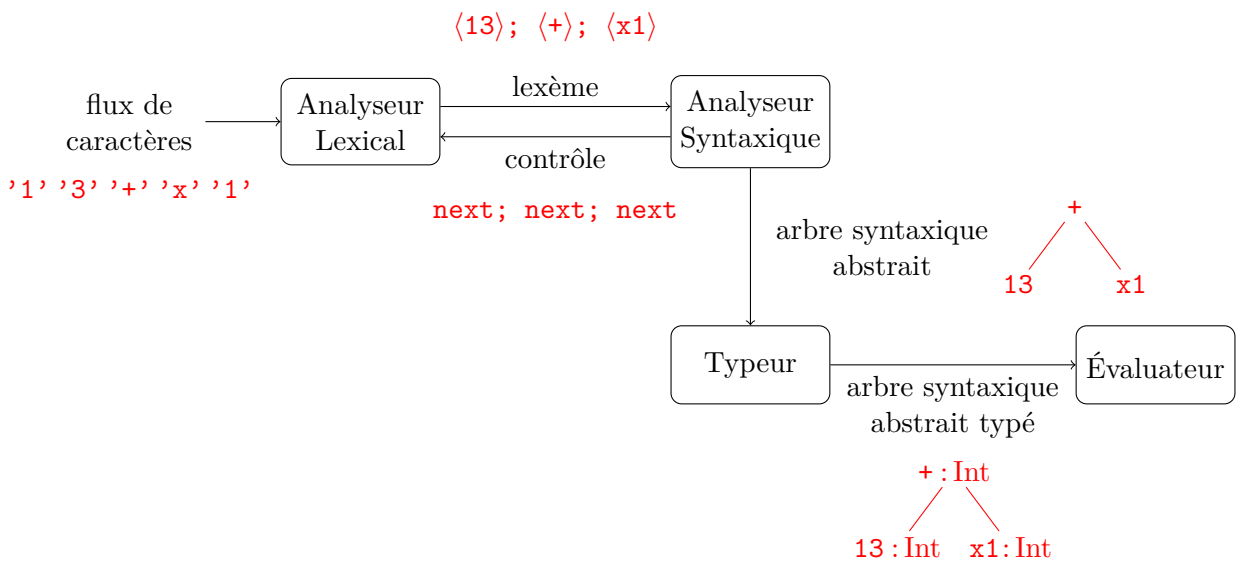

FIGURE 1 – Le détail d'une chaîne de compilation simple.