DLP : Développement d'un langage de programmation

Master STL, Université Pierre et Marie Curie

Année 2016–2017

# TME 10 – Ajout de mots clés (révisions)

Christian Queinnec & Antoine Miné

## 1 Introduction

Le but de ce TME est de réviser ce qui a été vu en cours et lors des TME précédents, en vue de l'examen. Nous allons ajouter plusieurs instructions au langage, ce qui nécessitera de travailler tous les aspects vus précédemment : les grammaires ANTLR 4, l'analyse syntaxique, l'AST, les visiteurs, l'interprète, le compilateur, la notion d'environnement. Le TME permettra également d'approfondir la distinction entre aspects statiques et aspects dynamiques du langage.

Dans ce TME, nous allons ajouter un mot-clé `exists`, permettant de déterminer si une variable de nom donné existe et est accessible au point où l'instruction est exécutée. Nous ajouterons également un mot-clé `defined`, permettant de déterminer si la variable indiquée a bien une valeur au points donné, ou bien si elle n'a pas encore été initialisée. Nous travaillerons sur ILP4, dans le *package* `com.paracamplus.ilp4.ilp4tme10`.

## 2 Travail à réaliser

### 2.1 Test d'existence de variable (mot-clé `exists`)

L'instruction `exists(a)` retourne la valeur vraie si et seulement si une variable locale ou une variable globale de nom `a` existe. Plus précisément :

- Une variable *locale* `a` existe si l'instruction `exists(a)` est dans le portée de la déclaration de `a`.
- Une variable *globale* `a` existe si elle apparaît à un point quelconque de l'arbre syntaxique du programme, même si le premier accès à la variable est exécuté après le mot-clé `exists`, et même si elle n'apparaît que dans une branche ou une fonction jamais exécutée.
- Nous supposerons également que les *constantes prédéfinies* (`pi`) et les *primitives* (`print`, `newline`) existent toujours.

Voici quelques exemples :

```
function f(a) (
  // a, g0, g1, g2, g3, f, pi et print existent
  // g4 n'existe pas
  g3 = a
)
in (
  // g0, g1, g2, g3, f, pi et print existent
  // a et g4 n'existent pas
  g0 = 1;
  f(g0);
  g1 = g2
)
// g0, g1, g2, g3, pi et print existent
// f, a et g4 n'existent pas
```

Le fait que, dans notre langage, les variables globales ne soient pas déclarées explicitement compliquera notre tâche.

### 2.1.1 Grammaire

Le mot-clé `exists` aura la syntaxe simple suivante :

```
exists(variable)
```

où `variable` est un identificateur.

**Travail à réaliser :** ajoutez `exists` à la grammaire ANTLR du langage ; expliquez pourquoi `exists` ne peut pas être implanté par un ajout de primitive.

### 2.1.2 AST et analyse syntaxique

**Travail à réaliser :** ajoutez un nouveau type de nœud `ASTexists` pour représenter ce mot-clé ; modifiez en conséquence toutes les classes manipulant les nœuds de l'AST (fabriques, visiteurs, etc.) ; ajoutez le mot-clé au *Listener* ANTLR.

### 2.1.3 Interprète

**Travail à réaliser :** ajoutez le support pour `ASTexists` dans l'interprète ; construisez une base de tests couvrant différent cas (variable globale ou locale, position du mot-clé par rapport aux affectations, etc.) ; testez en local ; ajoutez les tests à l'intégration continue et vérifiez que, après un *push*, les tests fonctionnent sur le serveur GitLab du cours.

*Conseil.* Déterminer, en un point de l'interprétation, l'existence d'une variable locale est assez simple puisque l'environnement lexical, contenant ces variables, est passé en argument au visiteur de l'interprète. Déterminer l'existence d'une variable globale est plus complexe, puisque celle-ci peut-être rencontrée par le visiteur de l'interprète *après* l'évaluation du mot-clé `exists`. Il est donc nécessaire de visiter l'AST complet avant l'interprétation pour collecter les variables globales. Vous pourrez éventuellement vous inspirer ou réutiliser le visiteur du compilateur qui effectue un travail similaire.

### 2.1.4 Compilateur

**Travail à réaliser :** ajoutez le support pour `ASTexists` dans le compilateur ; testez en local et dans l'intégration continue.

Dans votre rendu, dans le champ *Release notes* de votre tag, vous répondrez aux questions suivantes :

1. L'information calculée par `exists` est-elle statique ou dynamique ?
2. Quelle partie du travail doit être effectuée par le compilateur (statiquement), et quelle partie par le code C généré ou la bibliothèque d'exécution (dynamiquement) ?

## 2.2 Test de définition de variable (mot-clé `defined`)

Pour le mot-clé `defined`, une variable est définie s'il lui a été affecté une valeur avant l'exécution du mot-clé. Les variables locales de bloc sont toujours définies, puisque chaque déclaration `let var = exp1 in expr2` précise une valeur d'initialisation `exp1`. De même pour les arguments formels des fonctions (eux aussi locaux), puisqu'un appel de fonction précise leur valeur initiale. Les seules variables qui sont potentiellement non définies sont les variables globales, avant leur première affectation. Voici quelques exemples :

```
function f(a) (
  // f, g0 et a sont définies
  // g1, g2 et g3 ne sont pas définies
  g2 = a
  // f, g0, g2 et a sont définies
  // g1 et g3 ne sont pas définies
)
in (
  // f est définie
  // g0, g1, g2 et g3 ne sont pas définies
  g0 = 1;
  // f et g0 sont définies
  // g1, g2 et g3 ne sont pas définies
  f(12);
  // f, g0 et g2 sont définies
```

```
16    // g1 et g3 ne sont pas définies
17    g1 = 12;
18    // f, g0, g1 et g2 sont définies
19    // g3 n'est pas définie
20  )
```

**Travail à réaliser :** Comme pour le mot-clé `exists`, vous ajouterez le support pour le mot-clé `defined` en plusieurs étapes :

1. ajoutez dans la grammaire ANTLR une règle pour `exists(variable)` ;
2. ajoutez dans l'AST et l'analyseur syntaxique une classe `ASTdefined` ;
3. construisez une base de tests, et testez (au fur et à mesure) votre implantation ;
4. ajoutez le support de `ASTdefined` dans l'interprète ;
5. ajoutez le support de `ASTdefined` dans le compilateur ;
6. ajoutez les tests à l'intégration continue et vérifiez le bon fonctionnement sur le serveur GitLab du cours.

Comme précédemment, répondez aux questions suivantes dans les *Release notes* de votre tag GitLab :

1. Quelle partie du travail doit être faite statiquement, et quelle partie doit être faite dynamiquement ?
2. Que se passe-t-il si `defined` est appelée sur une variable locale en dehors de sa portée ?

## 2.3 Rendu

Vous effectuerez un rendu en vous assurant que tout le code développé a été envoyé sur le serveur GitLab (*push*) dans votre *fork* d'ILP4. Vous vous assurerez que les tests d'intégration continue développés ont bien été configurés et fonctionnent sur le serveur. Vous ajouterez un tag « rendu-initial-tme10 » en fin de séance, puis un tag « rendu-final-tme10 » quand le TME est finalisé. N'oubliez pas de répondre aux questions demandées dans les *Release notes* du tag final.