M1IF
Projet Compilation

Paul Feautrier

21 janvier 2009

1 Introduction

L'objectif du projet est la réalisation d'un compilateur pour le langage Pascal- (voir `www.cs.uleth.ca/~hossain/cs4600a/cs4600W2Outline.pdf`). Ce langage est un sous-ensemble de Pascal. Les différences essentielles sont :

- Les seuls types de base sont les entiers et les booléens. Il n'y a ni réels ni caractères.
- Les types construits doivent être nommés à l'aide d'une construction `type`.
- Dans une déclaration, on doit toujours utiliser un nom de type et non pas le type lui-même.
- les seules structures de données sont les tableaux (`array`) et les structures (`record`).
- Il n'y a pas de fonction, mais seulement des procédures. Ceci n'est pas en fait une véritable restriction.
- L'opérateur `new` n'existe pas. Toutes les données sont logées dans la pile d'exécution.

Les aspects essentiels de Pascal- sur lesquels votre compilateur devra se concentrer sont :

- Le contrôle des types.
- La gestion de la pile d'exécution, en particulier en présence de procédures imbriquées.

La cible de la compilation est la machine RISC en FPGA objet du TD9 du cours d'architecture (voir `perso.ens-lyon.fr/jeremie.detrey/05-archi`). Comme cette machine est très simple, il est possible de générer du code binaire sans utiliser d'assembleur.

Le langage d'implémentation du projet est laissé libre. On notera cependant que l'utilisation de générateurs d'analyseurs syntaxiques du type de Lex

et Yacc est presque obligatoire. Ceci milite soit en faveur de C/C++, soit en faveur d'OCaml [LDG+00]. Il existe probablement des versions de Lex et Yacc en Java, mais elles n'ont pas l'autorité des versions C et OCaml. Enfin, un compilateur manipule nécessairement des structures de données dynamiques ; l'existence d'un ramasse-miette (*garbage collector*) est une aide appréciable, ce qui favorise Java et OCaml. Au total, le langage conseillé est OCaml, mais vous êtes libres de ne pas suivre cette recommandation (à votre grand dam).

Le projet sera matérialisé sous la forme d'un compilateur (source et binaire) et de programmes tests dont on vérifiera la bonne compilation et la bonne exécution[1]. Le tout sera accompagné d'un rapport d'implémentation et d'un guide d'utilisation. Chaque projet donnera lieu à une soutenance.

Le projet doit être réalisé par groupes de deux à trois personnes au maximum. Il n'est pas recommandé de travailler seul.

Certaines parties de ce travail se réfèrent aux TD de Compilation de Florent de Dinechin [dD02] ; d'autres se réfèrent aux TD d'architecture de Jérémie Detrey.

Ce document est accessible à travers la page Web `perso.ens-lyon.fr/Paul.Feautrier` et sera régulièrement mis à jour. Vous y trouverez également la grammaire de Pascal-. D'autres informations (par exemple des compléments de cours), seront ajoutées en temps réél.

2 Le langage

Le langage Pascal- pour lequel il faut réaliser un compilateur est un sous-ensemble de Pascal. En particulier, en cas de doute sur la correction d'un programme, vous pouvez toujours vous référer à une définition canonique de Pascal (par exemple [Gro92]) ou même avoir recours au compilateur standard (par exemple `gpc`).

La grammaire de Pascal- a été extraite du site `www.cs.uleth.ca/~hossain/cs4600a/cs4600W2Outline.pdf`. Vous la trouverez telle quelle dans le fichier `implementation.tar`. Cette grammaire est écrite en BNF avec les conventions suivantes :

- Les non-terminaux sont des identificateurs commençant par une lettre majuscule (attention si vous utilisez Ocamlyacc).
- Les terminaux sont écrits entre apostrophes (par exemple `'program'`).
- Les non-terminaux dont le nom est de la forme `xxxNameDef` et `xxxNameUse` représentent les identificateurs, c'est-à-dire les chaînes de caractères

[1] Il est recommandé que la promotion mette en commun ses programmes tests.

commençant par une lettre suivie de lettres ou de chiffres. La première forme indique qu'il s'agit de la première apparition de l'identificateur dans une portée lexicale (l'identificateur doit être *rangé* dans une table de symboles), et la deuxième forme qu'il s'agit d'une utilisation (l'identificateur doit être *trouvé* dans une table des symboles).
- Le non-terminal `Numeral` représente un nombre entier (suite de chiffres).
- Le non-terminal `Empty` représente la chaîne vide.
- La construction `[ syntagme ]` indique que la présence du syntagme est facultative.
- La construction `{ syntagme }` indique que le syntagme peut être répété un nombre arbitraire (y compris nul) de fois.

Ces règles ont peu de chances de convenir au générateur d'analyseur lexical et sémantique que vous allez choisir. C'est à vous de faire l'adaptation.

2.1 Sémantique

La sémantique est en général celle de Pascal.

2.1.1 Tableaux

Les tableaux peuvent avoir autant de dimensions que l'on veut. Chaque dimension est spécifiée à l'aide d'une borne inférieure et d'une borne supérieure qui peuvent avoir des valeurs quelconques mais constantes. En Pascal, avant tout accès à un tableau, on doit vérifier que les indices sont dans les bornes pour chaque dimension.

Les tableaux sont alloués dynamiquement sur la pile d'exécution par le prélude de la fonction qui les contient. Ils doivent être détruits par le postlude correspondant.

2.1.2 Variables entières

Pascal- traite les variables scalaires entières, qui peuvent figurer dans des indices, des bornes de boucles et des expressions arithmétiques entières. Les constantes entières ont la forme usuelle. Les entiers sont codés sur 16 bits, pour se conformer à la structure de la machine cible.

La création et la destruction des variables se font de la même façon que pour les tableaux.

2.1.3 Variables et expressions booléennes

Il existe des expressions booléennes, des variables et des constantes booléennes. Les expressions booléennes sont construites à partir d'expressions

entières, de variables et de constantes à l'aide des opérateurs de comparaison `<, >, <=, >=, =, <>` et des opérateurs booléens (`and, or, not`).

On remarquera que la grammaire n'interdit pas des expressions invalides, comme `false + 1`. C'est le contrôle de type qui doit détecter ces erreurs.

Un booléen peut être stocké dans un mot (16 bits, plus rapide) ou dans un bit (économie de mémoire). Dans la deuxième solution, l'accès est une opération complexe (masquages et décalages). A vous de choisir.

2.1.4 Instructions

Les instructions sont séparées les unes de autres par un point-virgule.

Instructions d'affectation La syntaxe est usuelle : `lhs := rhs`. Le membre gauche (lhs) doit être une référence à une variable. Le membre droit (rhs) doit être une expression du même type que le membre gauche.

Entrées et sorties Pour pouvoir vous assurer du bon fonctionnement d'un programme compilé, il est indispensable de pouvoir faire des lectures et des écritures. Plutôt qu'utiliser les instructions d'entrée / sortie de Pascal, assez complexes, vous définirez les procédures `readInt, readBool, writeInt` et `writeBool`. Vous devrez porter ces procédures à la connaissance du vérificateur de type, et prévoir leur implémentation directement en langage machine dans la bibliothèque d'exécution (voir plus loin Sect. 4.6).

Instructions conditionnelles Une instruction conditionnelle peut avoir l'une des deux formes :

```
if expression then statement
```

ou :

```
if expression then statement
else statement
```

On doit vérifier que le type de l'expression est bien booléen. On notera que si cette grammaire est soumise à Yacc ou à un générateur de la même famille, on obtient un *shift/reduce conflict* qui est correctement résolu.

Boucle `while`

```
while expression do statement
```

Le nombre d'itération d'une boucle `while` n'est pas fixé d'avance. La boucle se termine quand l'expression testée devient fausse.

Procédures Les procédures de Pascal peuvent utiliser deux modes de transmission des paramètres ([Ris] Sect. 6.3) :

- La transmission par valeur : le paramètre effectif est recopié sur la pile.
- La transmission par référence : c'est un pointeur qui est copié sur la pile.

Il est conseillé de toujours utiliser un mot (16 bits) pour ces recopies, quelque soit le type du paramètre.

Le programme En Pascal-, le programme principal peut contenir du code et des déclarations. Il doit typiquement engendrer le point d'entrée du programme objet.

2.1.5 Déclarations

Par rapport à Pascal, le mécanisme des déclarations a été simplifié (pour le compilateur, mais pas pour l'utilisateur). La déclaration d'une variable ne peut utiliser qu'un type prédéfini. Celui-ci est l'un des types de base `integer` ou `real`, ou a été défini antérieurement dans une déclaration de type. Noter en particulier comment les tableaux à plusieurs dimensions sont déclarés.

```
type row = array [1..10] of integer;
type matrix = array [1..10] of row;

var i : integer;
    M : matrix;
```

Une procédure a également un type et celui-ci doit être vérifié. Ceci revient à vérifier que les paramètres effectifs ont un type conforme.

C'est également à ce niveau que doivent être traitées les constantes : typage et préparation de la génération de code. En général, un compilateur range les constantes dans une zone de mémoire statique, par exemple juste après le code du programme. Il peut être intéressant d'éliminer les redondances.

2.2 Exemples

Vous trouverez de très nombreux exemples de programmes Pascal (à convertir, si possible, en Pascal-) dans le livre de Grogono [Gro92].