Réalisation d'un compilateur

Corinne Ancourt Fabien Coelho

29 septembre 2014

1 Outils disponibles

L'objectif de cette séance de travaux pratiques est de réaliser un petit compilateur pour un pseudo **pascal**, ou tout autre syntaxe de votre choix, avec `jflex` et `cup`, pour une mini machine. Vous disposez des outils suivants :

1.1 Analyseur lexical `jflex`

Il génère un analyseur lexical *(lexer)* à partir d'un fichier de configuration spécifique. Il permet de découper un flux de caractères (`Reader`) en *tokens* (unités lexicales élémentaires).

1.2 Analyseur syntaxique `cup`

Il génère un analyseur syntaxique *(parser)* à partir d'un fichier de configuration spécifique. Une classe contient une méthode `parse` qui analyse la syntaxe d'un flux de token, une autre contient les numéros des tokens pour la synchronisation avec l'analyseur lexical.

1.3 Machine virtuelle minimale `run`

La machine cible est une machine à pile (il n'y a pas de registre) avec des mémoires de programme et de données, dont les opérations fonctionnent sur des entiers exclusivement. Elle est exécutée avec la commande `run`. Elle comprend 17 instructions élémentaires, dont voici une description rapide :

PUSH x met l'entier x sur la pile

LOAD accès mémoire à l'adresse en tête de pile

STORE stocke l'élément en tête de pile (adresse, valeur)

SWAP inverse les deux éléments en tête

ADD SUB MUL DIV AND OR opérations arithmétiques et logiques sur deux entiers en tête de pile, résultat sur la pile.

NOT opération logique unaire

BEZ x, BGZ x branchements conditionnels selon l'élément en tête de pile.

GOTO branchement vers l'adresse en tête

STOP arrêt de la machine

IN OUT entrées et sorties entre la pile et l'écran ou le clavier

1.4 Assembleur minimum `asm`

Nous vous fournissons un assembleur qui permet de convertir du code symbolique avec des labels vers du code machine. Il permet également de réserver de la mémoire.

2 Développement progressif

Quelques idées pour développer **progressivement** votre compilateur, dans le bon sens.

Il est très important de tester au fur et à mesure de vos travaux. Le compilateur doit produire le texte assembleur du programme compilé, qui peut apparaître sur la sortie standard par exemple.

Choisir tout d'abord une syntaxe pour votre langage (expressions, variables, fonctions, arguments), par exemple en vous inspirant du langage `pascal`.

2.1 Langage de base

1. définir une expression comme une contante entière
2. afficher une expression
3. un programme est une liste d'instructions
4. ajouter les opérations arithmétiques + et - : associativité ?
5. ajouter les parenthèses
6. ajouter les opérations * et / : associativité, précédence ?
7. permettre des commentaires dans le programme (ligne, zones)
8. déclarer/définir une variable scalaire
9. affectation d'une expression à une variable scalaire
10. utilisation d'une variable dans une expression
11. entrée standard vers un scalaire
12. condition *if-then-else*, condition vrai si entier non nul
13. opérations : moins unaire, modulo
14. boucle *while* : *algorithme d'Euclide pour le calcul du PGCD*
15. définition de tableaux d'entier de taille fixée
    algorithme du crible d'Ératosthène pour trouver les nombres premiers
16. déclaration d'une liste de variables
17. initialisation d'une variable scalaire avec sa déclaration
18. constantes booléennes *true false*
19. comparaisons entières $< \leq > \geq = \neq$
20. calcul des expression booléennes (et, ou, not)
    attention, l'opérateur NOT de la machine n'est pas *logique* mais *binaire* !
21. fonction système *exit(int)* qui arrête le programme en laissant la valeur sur la pile

2.2 Fonctions

1. simples : pas de retour, pas de paramètres
2. avec retour et commande de retour d'une valeur
3. avec arguments passés par valeur
4. variables dans la fonction globales par défaut
5. changement : variables de fonction locales par défaut
6. déclaration de variable globales explicites...
7. fonctions récursives : (par exemple factorielle)
   sauvegarde des variables **locales** de la fonction...

2.3 Développements avancés

1. construction *case* : choix multiples entiers
2. construction *elsif* intermédiaire
3. définitions de constantes symboliques (table symbole ?)
4. ajouter un type pointeur sur entier
5. passage d'arguments scalaire par référence (par pointeur, modifiables)
6. passage d'arguments tableaux par référence
7. ajouter un opérateur factoriel
8. typage : conversion entier booléen quand nécessaire seulement
   ajouter la déclaration de variables booléennes dont les valeurs sont toujours limitées à 0 ou 1.
9. si des évaluations d'expression sont utilisées plusieurs fois (modulo ? autre ?), utiliser des variables temporaires
10. faire une évaluation des expressions logiques rapide (Vrai OU x est Vrai)
11. initialisation d'un tableau avec sa déclaration...
12. ajouter une allocation dynamique de tableaux dans la zone de données (on ne s'occupera pas de libérer la mémoire...)
13. implémenter une évaluation partielle des expressions entières
14. vérifier que les accès de tableaux sont dans les bornes déclarées

composé avec LaTeX, document révision 478