INTRODUCTION

- Nom J.M. (Mike) Robson

- mail robson@labri.u-bordeaux.fr

- Titre Traduction de Programmes

- Philosophie Comprendre comment nos programmes en langages évolués peuvent être exécutés par des machines avec jeu d'instructions et architecture simples

- Calendrier Lundi: cours
  mercredi TD approfondi
  Projet: date? coefficient?

- Transparents:
  `http://dept-info.labri.u-bordeaux.fr/`
  `~robson/transparents/compilation/`

- Livres: Le Dragon et autres

- Langage et langue: Algol-6X, Pascal, CPL, BCPL, C, java

APERCU

- En finir avec l'analyse syntaxique
- Grammaires attribuées comme outil pour analyse sémantique et traduction
- Génération de code intermédiaire
  structures performantes du langage cible qui correspondent aux structures utiles du langage source
- Code pour de vraies machines
- Méthodes d'optimisation
- Langages Fonctionnels

En finir avec l'analyse syntaxique Récapitulation des algorithmes <X>LR(1)

Toujours le même principe

- Pile (virtuelle) de terminaux et non-terminaux
- 2 actions possibles: réduction et décalage
- décision selon l'état d'un automate après lecture de la pile et peut dépendre aussi du prochain symbôle
  reconnaissance de "manches"
- empile l'état de l'automate
- <X> ne dépend que de la méthode de construction de l'automate

L'algorithme LALR(1)

Construction de l'automate LALR(1)

- Automate non-déterministe LR(0)

- Déterminisation

- Si une ϵ transition produit une transition d'un état avec l'item $X : \alpha.Y\beta\{F\}$ à un état avec l'item $Y : .\gamma\{E\}$, rajouter à E tout symbôle qui pourrait être le premier symbôle de β (et rajouter aussi F si β peut être vide)

- propager

Algorithme utile et utilisé (yacc, gag,...)

Comment le LALR(1) peut-il être différent du SLR(1)

Les items pareils de plusieurs états de l'automate LALR peuvent avoir des ensembles de prévision différents; dans l'automate SLR, le même item a partout le même ensemble de prévision; la différence peut servir à résoudre une ambigité.

- $P : Yq$; $P : ya$; $Y : y$; $X : Ya$
- si on cherche un X, y suivi de a doit être réduit à Y
- si on cherche un P, non
- $S : xX$; $S : pP$

- Donc, il existe des langages qui sont LALR(1) mais ne sont pas SLR(1)
  en outre si une grammaire est SLR(1), un analyseur LALR(1) peut mieux identifier l'erreur dans un programme incorrect; (parce que l'analyseur SLR aurait pu faire des réductions de plus avant de remarquer l'erreur)

Un très petit exemple concret du Dragon

$S \to Aa \mid bAc \mid dc \mid bda$
$A \to d$
Le problème pour SLR(1) vient d'un "programme" comme dc

LALR(1) est moins puissant que LR(1)
un exemple

$S \to Aa \mid bAc \mid Bc \mid bBa$
$A \to d$
$B \to d$
est LR(1) mais pas LALR(1).

Mais LALR(1) suffit pour permettre des grammaires raisonnables pour les langages de programmation.

Complexité des algorithmes XLR(1)

- Temps d'analyse: est toujours linéaire (fonction de la longueur du programme)

- Espace mémoire: pour SLR et LALR exactement le même (à part astuces de compactification); le nombre d'états est le nombre d'états de l'automate SLR(0) (déterministe) dans le pire cas 2^n où n est le nombre d'états de l'automate nondéterministe LR(1): peut être le nombre de chemins de l'automate nondéterministe? ($\approx n!$)

- Temps de calcul des tableaux: au pire la taille du tableau déterministe multiplié par la taille de l'alphabet.

Exemple d'un langage où la taille du tableau LR est exponentielle (par rapport à celle de la grammaire)

$S \to A_i b_i \quad 1 \leq i \leq n$

$A_i \to a_j A_i \mid a_j \quad j \neq i$

Complexité Théorique de l'analyse pour une grammaire non contextuelle

- Programmation dynamique en temps $O(n^3)$
- Réduction (en principe) par multiplication de matrices
- Mais $O(n^2)$ si la grammaire est non ambigüe
- Et ce ne sont que des bornes supérieures pour des grammaires irréalistes
- Mais ce genre d'algorithme d'analyse est trop gourmand d'espace