Compilation - Devoir à la maison

Optimisation d'automates

Les parties d'implémentation 2 et 3 sont indépendantes, même s'il n'est pas interdit de partager intelligeamment du code entre elles. Vous travaillerez par binôme, en cas de monôme il ne fera qu'une des parties d'implémentation.

Chaque binôme devra avoir envoyé un mail le définissant à `Florent.de.Dinechin`*, avant le lundi 22 octobre à midi. Les retardataires seront binômés de force dans l'arbitraire le plus serein.*

Le devoir est à rendre par mail pour le lundi 12 novembre à minuit, le cachet du postmaster faisant foi. Ce sera un répertoire taré au nom d'un des binômes, contenant un fichier postscript pour le blabla, et différents sources et exécutables propres et abondamment commentés.

On considère un automate défini par une table de transitions. Le but de ce devoir est de définir et d'implanter des techniques de compression de table de transition, sans trop pénaliser le temps de transition de l'automate.

1 Introduction

Vous trouverez sur `http://www.ens-lyon.fr/~fdedinec/enseignement/compil/` des fichiers concernant ce devoir.

L'exemple `toy.flex` est coupé presque directement de la documentation de `flex`. Compilez-les avec l'option `-Cf` de `flex`, qui ne compresse pas les tables.

Question 1-1 Répondez, au vu du fichier `lex.yy.c` obtenu, aux questions suivantes :

- Combien d'états a l'automate ?
- Combien d'états sont acceptants ?
- Que représentent les nombres négatifs dans la table de transition ?
- Quelle est la taille de l'ensemble des tables en octets ?
- Quel est à vue de nez le pourcentage de transitions vers échec ?

Désormais (et dans votre prose à vous), on appellera transition utile une entrée de la table qui indique effectivement un autre état de l'automate (et non pas échec).

Question 1-2 Comparez les tailles d'automates envisageables respectivement en analyse lexicale et analyse syntaxique.

Cahier des charges

Votre implémentation pourra être écrite en C, C++, java ou ocaml. Elle devra impérativement utiliser des tables strictement identiques à celles présentes dans le fichier `lex.yy.c` précédent, éventuellement ocamélisées bien sûr. Votre programme devra également être parfaitement compatible avec ce `lex.yy.c`, c'est-à-dire fournir les mêmes sorties sur les mêmes entrées.

2 Compression de table par recouvrement des lignes

L'idée est ici d'aplatir toutes les lignes de la matrice de transition en un seul maouss tableau de transitions, pour écraser les transitions vers échec par des transitions utiles. Il n'est pas possible en général d'empiler toutes les lignes de la matrice les unes sur les autres : on écraserait des transitions utiles par d'autres. Pour éviter cela, chaque ligne est décalée de manière à ne pas écraser de transition utile.

Question 2-1 Comment se souvenir tout de même quelles étaient les transitions vers échec ?

Question 2-2 Comment se présente à présent la fonction `yy_next(state, char)` ?

Naturellement il y a moult décalages possibles, donnant des tableaux de tailles différentes, et toute la difficulté est à présent de définir un algorithme calculant les décalages de manière à minimiser le tableau résultant. Je vous rassure, le problème est NP-complet. Une heuristique simple est de s'occuper gloutonnement des cas les plus difficiles en premier.

Question 2-3 Détaillez et implémentez.

3 Compression de table par coloriage de graphe

Dans cette approche, on utilise une partition de l'ensemble des lignes telle que dans chaque partie, on puisse aplatir les lignes les unes sur les autres sans décalage.

Question 3-1 Comment se souvenir tout de même quelles étaient les transitions vers échec ?

Question 3-2 Comment se présente à présent la fonction `yy_next(state, char)` ?

Pour déterminer la partition, on construit un graphe dont les nœuds sont les lignes, et dont les arêtes représentent les conflits entre les lignes (il y a conflit lorsqu'on ne peut pas aplatir une ligne sur une autre). Trouver une partition revient alors à trouver un coloriage du graphe.

Question 3-3 Détaillez et implémentez. Pourquoi est-ce à nouveau NP-complet ? Dans la mesure du possible vous irez pêcher sur internet une bibliothèque pour le coloriage de graphes (voir Google en tapant *graph coloring library*).

4 Résultats

Présentez proprement vos résultats (taille finale des tables en octets). Comme vous n'avez pas hésité à essayer de nombreuses variantes pleines de subtilité, analysez leur impact.

5 Question subsidiaire

Il y a dans la distribution source d'Objective Caml un fichier `lex/compact.ml` (une copie se trouve dans le répertoire du devoir), et qui fait exactement ce qu'on vous demande (en 120 lignes).

Question 5-1 Quel est son principe ?