Automates à compteur avec limite d'inversion
Le test du vide

Nicolas Feron

Université Libre de Bruxelles, Bruxelles, Belgique
nicolas.feron@ulb.ac.be

Abstract

Les automates à compteur avec limite d'inversion sont déjà connus et introduits dans un article de Oscar H. Ibarra. Le but de ce mini-mémoire était de chercher et implémenter un test du vide pour ces derniers. Est-ce un test réalisable ? En quelle complexité ? Quels éléments rendent le problème décidable ? Ce sont toutes des questions auxquelles on va tenter de répondre dans cet article.

TO BE COMPLETED

Introduction

Les automates sont souvent décrits comme des machines donnant une réponse suivant une entrée à fournir. De manière plus précise et scientifique, ce sont des 5-tuples

$$M = (Q, \Sigma, q_0, \delta, F)$$

avec respectivement un ensemble d'états, un alphabet des symboles d'entrée, un état initial, une fonction de transition et un ensemble d'états acceptants. On commence toujours par l'état initial, et on voyage dans les états de Σ en suivant la fonction δ pour les valeurs en entrée.

Cependant cet article se concentre sur le test du vide des automates à compteur avec limite d'inversion. Ce sont des automates finis [1] déterministe [2] à compteurs [3] avec limite d'inversion [4], ils peuvent être décrits par un 6-tuple

$$M = (k, Q, \Sigma, \delta, q_0, F)$$

respectivement la limite sur les compteurs, les états, les entrées, la fonction de transition, l'état initial et les états acceptants. Ces contraintes simplifient le problème étudié (test du vide) et notamment la limite d'inversion qui le rend décidable en temps polynomiale pour les automates à compteur en ajoutant une limite à la **taille** des mots d'entrée (détaillé dans un article d'Oscar H. Ibarra, Journal 22).

1. un nombre fini d'états
2. pour un état et une entrée, la fonction δ donnera toujours le même résultat
3. des compteurs sont maintenus et incrémenté/décrémenté lors de changement d'états, ils peuvent être comparés à 0 dans la fonction δ
4. les compteurs peuvent changer de sens incrémentation <–> décrémentation un nombre fini de fois

Environnement automate

Avant de commencer à travailler sur un test du vide, il était nécessaire de développer un environnement de travail. Après une étude des automates, le travail a donc commencé. En résultat, nous avons maintenant un environnement permettant de créer des graphes représentant des automates. Ces derniers sont actuellement uniquement importés via des fichiers texte. Le parcours du graphe est possible suivant un mot et correspond au parcours d'un automate avec comme point de départ son état initial et en entrée toujours ce même mot.

Il est possible d'utiliser plusieurs types d'automate. En effet, l'environnement ne s'est pas développé d'un coup, il a fallu commencer par travailler avec de simples automates. Chaque ajout n'empêche pas l'utilisateur d'utiliser les automates acceptés précédemment, il y a donc un travail de maintenance sur les versions précédentes. Dans l'ordre, les ajouts furent : l'import depuis un fichier, le parcours de l'automate, les compteurs avec opérateur de comparaison =, le fait de pouvoir ignorer des compteurs, les opérateurs < et > et enfin les limites d'inversion.

Voici quelques commandes basiques pour l'utilisation de l'environnement :

Création d'un graphe automate :

Attention, le format du contenu du fichier d'import est strict : il faut, dans l'ordre, en première ligne, la taille du header (le nombre de ligne(s) contenant des caractéristiques de l'automate), les dites caractéristiques, les états et enfin les règles de transition (liens).

```
Graph newGraph("automaton3.txt");
```

Affichage des nombres de noeuds et liens :

```
newGraph.uglyPrint();
```

Entrée d'un mot :

La fonction renvoie un booléen d'où la condition ternaire

```
std::string word;
std::cout << "enter your word: ";
std::cin >> word;
(newGraph.wordEntryWithCounters(word)) ?
std::cout<<"accepted"<<std::endl :
std::cout<<"refused"<<std::endl;
```

Lancement d'un test du vide :

La fonction renvoie un booléen d'où la condition ternaire

```
std::string accepted = "accepts at least
one word";
std::string refused = "no word accepted";
(newGraph.voidTest(Graph::DEPTH_FIRST)) ?
std::cout << accepted << std::endl :
std::cout << refused << std::endl;
```

Le test du vide

Au niveau du test du vide, le plus simple à mettre en oeuvre est un algorithme qui va tester chaque mot jusqu'à en trouver un qui est accepté. Ce fut donc la première étape. Cependant un algorithme pareil peut tourner indéfiniment si nous ne lui donnons pas de condition d'arrêt. Au niveau de condition d'arrêt, nous pouvons lui fournir plusieurs choses, une limite d'itération de l'algorithme, une limite en taille de mot, une limite en nombre de noeuds parcourus, une limite sur le nombre de fois qu'on parcourt un même noeud,... Il y a beaucoup de possibilités à ce niveau.

Taille des mots

La première limite introduit fut sur la taille des mots, en effet, comme précédemment cité, Oscar H. Ibarra a introduit une limite sur la taille des mots pour ce type de compteur (avant d'être certain d'avoir un état acceptant ou un boucle dans l'automate). Cette limite prend en compte plusieurs caractéristiques de l'automate étudié : le nombre de compteur(s) 'm', le nombre de lien(s) 's' et une constance 'c' :

$$(m\ s)^{sc}$$

Cette version du test du vide a une complexité dans le pire des cas en $O(a^n)$ avec n, la taille maximale d'un mot, et a, le nombre de lettres dans l'alphabet de l'automate.

Parcours en longueur

C'est la première version du test du vide développée. Un parcours en longueur signifie qu'on travaille avec un stack (tas). On commence par remplir le mot avec la première lettre de l'alphabet, puis on met la deuxième lettre à la fin, etc.. Pour un alphabet A, B et des mots de 4 lettres, cela ressemble à : A, AA, AAA, AAAA, AAAB, AABA, AABB, ABAA, etc..

Parcours en largeur

Dans cette version, nous utilisons un heap (pile). Le premier rentré sera le premier servi, un parcours va ressembler à : A, B, AA, AB, BA, BB, AAA, AAB, ABA, ABB, etc..

Références

Oscar H. Ibarra (1973) Journal of Computer and System Sciences 7, pages 28 - 36

Oscar H. Ibarra (1981) Journal of Computer and System Sciences 22, 220 - 229

Michael Sipser (2006) Introduction to the Theory of Computation (2nd edition) chapitres 0 et 1