UMONS

Faculté
des Sciences

Automates

Étudiant : Benjamin André
Directrice :Véronique Bruyère
6 juillet 2020

Faculté des Sciences • Université de Mons • Place du Parc 20 • B-7000 Mons

Table des matières

Environnements théoriques

Exemples

Environnements algorithmiques

1 Introduction

Quatrième version du document. ***TODO** : Construire une introduction*

Contenu du mail introductif

Objectif Le but ultime du mémoire serait de comprendre l'approche de l'article "Actively learning to verify safety for FIFO automata" [2] et de reproduire une partie des expérimentations.

Les différentes étapes seraient

— te rappeler la notion d'automate fini (voir cours de compilation et de calculabilité et complexité)
— comprendre la notion d'automate fini minimal
— comprendre l'algorithme L* d'Angluin pour l'apprentissage d'un automate
— implémenter l'algorithme de Vardhan pour tester les résultats

Contexte Un professeur connaît un automate A. Un élève veut l'apprendre en posant deux types de questions, celles de *membership* et d'*equivalence*. Si on a un automate équivalent, on peut utiliser un algorithme pour obtenir l'automate minimal qui calcule le même langage que A.

Cet algorithme A* (ou variantes) a plein d'applications, notamment décrites dans l'article [2]. On y considères des automates FIFO, plus généraux. (*de ce que j'ai compris, ils sont équivalent à des automates avec un nombre d'état infini*). Pour ceux-ci, on voudrait savoir si les configurations calculées à partir de l'état initial évitent certaines mauvaises configurations.

Vu la puissance des automates FIFO, il est algorithmiquement impossible de calculer toutes les configurations atteignables à partir de l'état initial. (*ce qui serait cohérent avec ma compréhension*)

L'idée est alors de calculer (par apprentissage) un automate fini qui est une sur-approximation de cet ensemble de configurations et ensuite de tester si oui ou non on peut éviter les mauvaises configurations.

1.1 Reformulation du but général

Il existe les automates FIFO, qui ont des canaux pouvant contenir une infinité de symboles. Ceux-ci sont équivalents à des automates sans canaux mais avec un nombre infini d'états. Ceux-ci peuvent être utilisés pour faire du model checking, c'est-à-dire s'assurer de ne pas atteindre certains états peut importe la suite d'instructions.

1.2 Utilité des sections

Dans la section 2, les notions de langage sont posées. Elles sont ensuite utilisées dans la section 3 sur les automates. L'algorithme "Table Filling" de la section 4 se base sur ces automates et permet de minimiser et répondre à la requête d'équivalence. Cette requête d'équivalence est une des deux requêtes necéssaire au fonctionnement de l'algorithme d'angluin de la section 5.

2 Langage

Cette section pose les différents concepts et notations pour arriver à la notion de langage. Celles-ci reprennent les notations proposées par Hopcroft et al. [1].

2.1 Définitions

Un *alphabet* Σ est un ensemble fini et non vide de *symboles*. Un *mot* sur cet alphabet Σ est une suite finie de k éléments de Σ notée $w = a_1 a_2 \dots a_k$. L'entier k est la *longueur* de ce mot aussi notée $|w| = k$. Le *mot vide* est un mot de taille $k = 0$ noté $w = \epsilon$.

La *concaténation* de deux mots $w = a_1 a_2 \dots a_k$ et $x = b_1 b_2 \dots b_j$ est l'opération consistant à créer un nouveau mot $wx = a_1 a_2 \dots a_k b_1 b_2 \dots b_j$ de longueur $i = k + j$.

Proposition 2.1 (ϵ et la concaténation) *ϵ est* l'identité pour la concaténation, *à savoir pour tout mot w, $w\epsilon = \epsilon w = w$. (ce qui est trivial par la définition de la concaténation)*

L'exponentiation d'un symbole a à la puissance k, notée a^k, retourne un mot de longueur k obtenu par la concaténation de copies du symbole a. Noter que $a^0 = \epsilon$. Σ^k est l'ensemble des mots sur Σ de longueur k. L'ensemble de tous les mots possibles sur Σ est noté $\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k$.

Un ensemble quelconque de mots sur Σ est un *langage* [1], noté $L \subseteq \Sigma^*$. Étant donné que Σ^* est infini, L peut l'être également.

Exemple 1 (Définition d'un langage)
Voici des exemples utilisant plusieurs modes de définition. Σ y est implicite mais peut être donné explicitement.

- $L = \{12, 35, 42, 7, 0\}$, un ensemble défini explicitement
- $L = \{0^k 1^j | k + j = 7\}$, les mots de 7 symboles sur $\Sigma = \{0, 1\}$ commençant par zéro, un ou plusieurs 0 et finissant par zéro,un ou plusieurs 1. Ici, L est donné par notation ensembliste
- L est "Tous les noms de villes belges". Ici L est défini en français.
- $\emptyset$ est un langage pour tout alphabet.
- $L = \{\epsilon\}$ ne contient que le mot vide, et est un langage sur tout alphabet.

2.2 Opérations sur les langages

Soient L et M deux langages. Le langage $L \cup M = \{w | w \in L \vee w \in M\}$ est l'*union* de ces deux langages. Il est composé des mots venant d'un des deux langages.

Le langage composé de tous les mots produits par la concaténation d'un mot de L avec un mot de M est une *concaténation* de ces deux langages et s'écrit $LM = \{vw | v \in L \wedge w \in M\}$.

La *fermeture* de L est un langage constitué de tous les mots qui peuvent être construits par un concaténation d'un nombre arbitraire de mots de L, noté $L^* = \{w_1 w_2 \dots w_n | n \in \mathbb{N}, \forall i \in \{1, 2, \dots, n\}, w_i \in L\}$.

2.3 Expressions régulières

Certains langages peuvent être exprimés par une *expression régulière*. Un exemple de celles-ci est 01^*0 qui décrit la langage constitué de tous les mots commençant et finissant par 0 avec uniquement des 1 entre les deux.

Les expressions régulières suivent un algèbre avec ses opérations et leur priorités. Le langage décrit par une expression est construit de façon inductive par ces différentes opérations. Pour une expression régulière E, le langage exprimé est noté $L(E)$. Un langage qui peut être exprimé par une expression régulière est dit *langage régulier*.

Cas de base Certains langages peuvent être construits directement sans passer par l'induction :

- ϵ est une expression régulière. Elle exprime le langage $L(\epsilon) = \{\epsilon\}$
- $\emptyset$ est une expression régulière décrivant $L(\emptyset) = \emptyset$
- Si a est un symbole, alors **a** est une expression régulière composée uniquement de a. $L(a) = \{a\}$.
- Une variable, souvent en majuscule et italique, représente un langage quelconque, par exemple L.

Induction Les autres langages réguliers sont construits suivant différentes règles d'induction présentées par ordre décroissant de priorité :

- Si E est une expression régulière, (E) est une expression régulière et $L((E)) = L(E)$.
- Si E est une expression régulière, E^* est une expression régulière représentant la fermeture de $L(E)$, à savoir $L(E^*) = L(E)^*$.
- Si E et F sont des expressions régulières, EF est une expression régulière décrivant la concaténation des deux langages représentés, à savoir $L(EF) = L(E)L(F)$. La concaténation étant commutative, l'ordre de groupement n'est pas important, mais par convention, la priorité est à gauche.
- Si E et F sont des expressions régulières, $E + F$ est une expression régulière donnant l'union des deux langages représentés, à savoir $L(E + F) = L(E) \cup L(F)$. Ici encore, l'opération est commutative et la priorité est à gauche.

Exemple 2 (Expression régulière)

Soit l'expression $E = (b + ab)b^*a(a + b)^*$ qui représente le langage L.

- **ba** fait partie de L. En effet, en développant E avec des choix sur les unions et le degré d'une fermeture, on obtient $E = (b)b^0a(a + b)^0 = b\epsilon a\epsilon = ba$.
- **ababbab** fait partie de L. En développant à nouveau E en posant des choix sur les unions et fermetures, on obtient $E = (ab)b^0a(a + b)^4 = ab\epsilon a(a + b)(a + b)(a + b)(a + b) = ababbab$.
- **aa** ne fait **pas** partie de L. Supposons par l'absurde que $aa \in L$. Alors il existerait une façon de décomposer E en aa. Or, les premiers symboles doivent être soit b, soit ab. Il y a contradiction : E ne peut pas être décomposé. Comme aa ne peut pas être construit par E, $aa \notin L$.

2.4 Lemme de la pompe

***TODO** : rédiger, et fournir une preuve et un exemple d'utilisation*

Références

[1] J. E. HOPCROFT, *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*, Addison Wesley, nov 2000.

[2] A. VARDHAN, K. SEN, M. VISWANATHAN, AND G. AGHA, *Actively learning to verify safety for fifo automata*, in International Conference on Foundations of Software Technology and Theoretical Computer Science, Springer, 2004, pp. 494–505.