Mémoire intermédiaire PresTaf

Bourgeois Adrien, Marbois Bryce, Roque Maxime, Turnherr Jérémy
Université UFR Collégium des Sciences et Technique d'Orléans-la-Source

15 mars 2017

Table des matières

1 Résumé du projet

Notre projet s'inscrit dans la mis-à-jour de PresTaf, logiciel d'analyse de formules logique et de leur transformation en automate minimal. Nous intervenons ici dans la création d'un interfaçage Lua, ainsi qu'une reprise du code source PresTaf pour donner une version plus optimisée, plus claire et propre.

Dans un second temps nous chercherons à intégrer de nouvelles logiques telle que la logique monadique ou encore l'interprétation des formules de Presburger en base -2.

2 Domaine

Le logiciel sur lequel on s'appuie pour notre travail de départ est PresTaf implémentant en Java la logique Arithmétique de Presburger [3]. Il existe des logiciels concurrents travaillant avec d'autres logiques telle que la Logique monadique du second ordre [4] avec des logiciel comme Mona [5], ou la logique arithmétique de Presburger et d'autres logiques sur les mots infinis avec Lash [1]

3 Analyse de l'existant

PresTaf est un programme codé par M. Jean-Michel Couvreur, qui prend des formules de Presburger en entrées et les résout à l'aide d'automates minimaux. Tout d'abord il génère des automates déterministes et finis mais non minimaux. Il faut donc les minimiser, et pour se faire PresTaf utilise un algorithme d'Hopcroft modifié. L'ensemble des transitions menant de l'état initial vers un état final est solution de la formule. En outre si l'état final est l'état *zero* alors il n'y a aucune solution et si l'état final est l'état *one* alors la formule est une tautologie.

Mona, est une bibliothèque C qui résout des formules monadiques. La où PresTaf n'implémente à ce jour que la logique arithmétique de Presburger, la logique monadique pourrait être implémentée dans le futur.

Lash [1] est une bibliothèque C qui résout des formules de Presburger, mais la différence avec PresTaf est qu'elle fonctionne sur des automates infinis. Cette différence induit une importante baisse de performance. En effet PresTaf pour les mêmes formules était bien plus rapides à s'executer que Lash. [2]

4 Besoins fonctionnels

4.1 Prototype papier

Le prototype qui suit serait un fichier Lua qui se servirait de la logique *MaLogique*.

```
logique = require("MaLogique") // Choix de la logique
x = logique.var('X') // Declaration d'une variable
y = logique.var('Y')
f = (x ∨ y) ∧ x ∧ ¬ y
// Pour exporter l'automate
todot(f, "f.dot")
// Soient des formules a et b
// Intersections et unions d'automates.
c = a ∩ f ∪ b
```

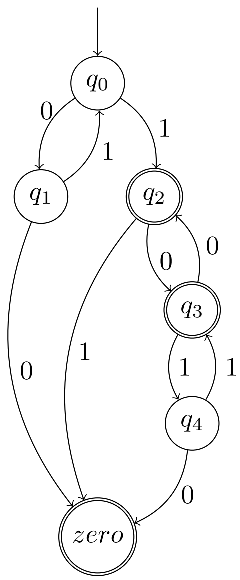
Pour lancer ce fichier Lua il faudrait passer par un fichier jar que l'on appellera *prestaf.jar*. Pour lancer le jar et le fichier lua il faudrait faire la commande suivante :

```
java -jar prestaf.jar fichier.lua
```

En sortie l'utilisateur aurait un fichier dot, par exemple, si l'utilisateur se sert comme logique de l'arithmétique de Presburger et que dans son fichier lua il écrit le code suivant :

```
pres = require("Presburger")
x = pres.var('X')
y = pres.var('Y')
f = x == y + 1
todot(f, "f.dot")
```

Le fichier f.dot ressemblerait à :



4.2 Fonctions

La priorité des fonctions varie de 1 à 5, du plus important au moins important, sachant que la priorité 5 correspond à une fonctionnalité optionnelle.

4.2.1 Bilbiothèque d'automate générique

Description : La bibliothèque PresTaf est générique et doit accepter toutes sortes d'automates.

Justification : L'utilisateur aura la possibilité d'implémenter ses propres logiques, la bibliothèque doit donc accepter toutes logiques. En effet PresTaf sera une bibliothèque d'automates, permettant de minimiser un automate, de faire des intersections, des unions, etc. Il ne faut donc pas que PresTaf soit ciblé sur Presburger, mona ou une quelconque autre logique.

Priorité : 1

4.2.2 Minimisation d'automate

Description : Ensemble de fonctions qui prennent un automate (fini, complet) et déterministe en entrée et retourne l'automate minimal équivalent.

Justification : Besoin initial.

Priorité : 1

4.2.3 Interfaçage Lua

Description : Permet le codage des automates en Lua, ainsi que l'utilisation de chaques fonctions qui seront ensuite exécutées en Java.

Justification : Le lua est un langage de script simple à prendre en main et qui permet facilement d'écrire des automates et d'utiliser des fonctions.

Priorité : 2

4.2.4 Portabilité du code

Description : Windows, MacOS, Linux

Justification : Comme java est un langage portable executé via la Java Virutal Machine (JVM), et que LuaJava est executé via java il embarque sa propre machine virtuelle, en théorie le code sera donc portable sur tous les systèmes d'exploitation. En dehors de la portabilité du code, Windows MacOs et Linux sont les principaux systèmes d'exploitation, il est donc important d'avoir un code portable pour chaque machine pour faciliter l'accès.

Priorité : 1

4.2.5 Optimisation

Description : La bilbiothèque d'automates PresTaf doit être rapide à s'exécuter.

Justification : L'utilisateur n'aura pas le temps d'attendre quelques dizaines de minutes que son automate soit généré. Il voudra obtenir son résultat rapidement.

Priorité : 5

5 Besoins non fonctionnels

5.1 Fonctions

5.1.1 Transformation de formules arithmétiques de Presburger

Description : Ensemble de fonctions qui prennent en entrée une formule arithmétique et retourne l'automate acceptant cette formule.

Justification : La bilbiothèque PresTaf n'implémentera pas d'elle-même une logique. Ainsi l'implémentation de la logique de Presburger en script Lua, permettra de fournir une démo à l'utilisateur. Il aurait un aperçu des bonnes pratiques à avoir, les méthodes qu'il se doit d'implémenter, et des fonctionnalités présentes.

Priorité : 1

5.1.2 Logique monadique du second ordre

Description : Il s'agit d'une logique du second ordre, c'est-à-dire qu'un prédicat peut avoir en argument un autre prédicat, mais celui-ci ne peut pas avoir un troisième prédicat en argument (arité un). De plus dans le cadre de la logique monadique du second ordre les quantificateurs ne peuvent être utilisés que pour les variables des prédicats du premier ordre (de type Presburger par exemple).

Justification : La logique de Presburger est moins complète que la logique Monadique, puisque la logique Monadique propose une notion de successeur, donc en implémentant la logique Monadique dans une autre bibliothèque Lua, on fournirait d'avantage de démo à l'utilisateur.

Priorité : 3

5.1.3 Acceptation de formules en base -2

Description : La base -2 est défini par : $\sum_{i=0}^{n}(-2)^i * k_i$. Par exemple $2_{decimal} = 0*(-2)^0 + 1*(-2)^1 \Rightarrow 2_{decimal} = -2_{base-2}$. Pour déterminer un nombre en base -2, il suffit de determiner son écriture binaire et ensuite d'appliquer le calcule base -2. Si l'on a le nombre $10110010_{binaire}$ alors on aura en base -2 : $0*(-2)^0 + 1*(-2)^1 + 0*(-2)^2 + 0*(-2)^3 + 1*(-2)^4 + 1*(-2)^5 + 0*(-2)^6 + 1*(-2)^7 = -146_{base-2}$

Justification : Il serait intéressant de permettre à l'utilisateur d'utiliser cette bilbiothèque avec diverses bases, surtout la base -2.

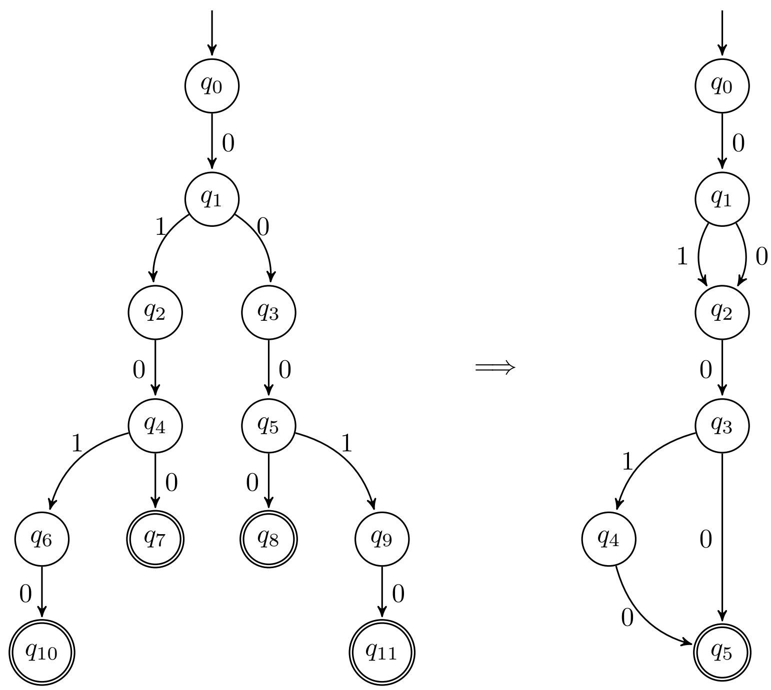
Priorité : 5

6 Prototypes et tests préparatoires

6.1 Blum

Voici un exemple que nous avons testé avec l'automate initial et l'objectif.

Automate initial Automate minimal



Nous avons analysé et tenté une implémentation en python de l'algorithme de Norbert Blum, basé sur celui de John Hopcroft, qui minimise un automate en temps n log n. Initialement l'algorithme divise les états en deux classes : les finaux et les non-finaux. Cet algorithme a besoin de plusieurs structures de données pour s'exécuter :

— Des listes doublement chaînées $L(i, a, j)$ avec i et j deux classes d'automates et a une lettre de l'alphabet, Tels qu'il existe un automate de la classe j atteint par un automate de la classe i à l'aide de la lettre a. $L(i, a, j)$ contient aussi sa taille pour pouvoir y accéder en temps constant. Les maillons de chaque liste contiennent d'une part, un automate contenu dans la classe i et vérifiant le tuple (i, a, j) et d'autre part un pointeur vers la liste $L(i, a, j)$.

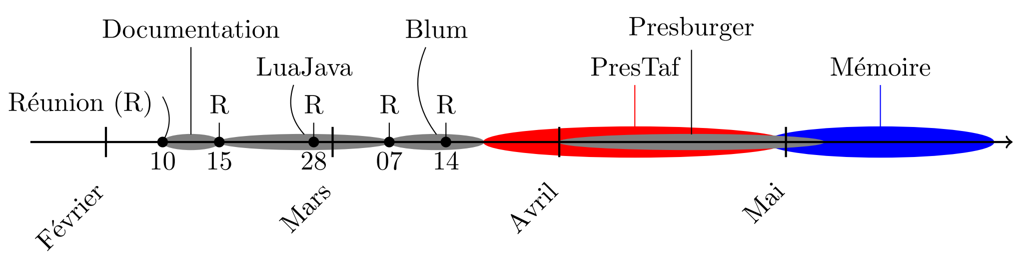
— Un tableau Δ à deux dimensions qui contient en fonction d'un automate p et d'une étiquette a, le pointeur vers le maillon, dans la structure $L(i, a, j)$ correspondante tel que p appartient à la classe i, et que l'automate d'arrivée par la lettre a appartienne à la classe j.Comme tous les automates traités sont déterministes, lorsque l'on applique a à p, on obtient au plus un seul automate, or cet automate d'arrivée n'appartient qu'à une seule classe d'équivalence, ainsi la case [p][a], ne pointe que vers un élément d'une liste unique.

— Un tableau Δ^{-1} à deux dimensions qui contient en fonction d'un état et d'une étiquette de l'alphabet, la liste des prédécesseurs de cet automate par cette étiquette. Ce tableau permet d'accéder plus rapidement aux prédécesseurs d'un état.

— Une liste $\Delta'(i, a)$, contient toutes les listes $L(i, a, j)$ quelque soit j.

— Un ensemble K contenant tous les $\Delta'(i, a)$ ayant au moins deux éléments. Ainsi tous les $\Delta'(i, a)$ dans K permettent une subdivision de leur classe de départ i, car il existe deux j différents qui succèdent à i par a.

7 Planning



Glossary

Arithmétique de Presburger L'arithmétique de Presburger à été introduite par Mojżesz Presburger en 1929. Cette arithmétique du premier ordre dispose de deux constantes 0 et 1 ainsi qu'un symbole binaire +. Ce langage est limité aux entiers naturels et est défini par les lois suivantes :

1. $\forall x, \neg(0 = x + 1)$
2. $\forall x, \forall y, x + 1 = y + 1 \rightarrow x = y$
3. $\forall x, x + 0 = x$
4. $\forall x, \forall y, x + (y + 1) = (x + y) + 1$

5. $\forall P(x, y_1, \ldots, y_n) \in$ Formule du premier ordre, $\forall y_1 \ldots \forall y_n[(P(0, y_1, \ldots, y_n) \vee \forall x(P(x, y_1, \ldots, y_n) \to P(x+1, y_1, \ldots, y_n))) \to \forall y P(y, y_1, \ldots, y_n)]$

Logique monadique du second ordre aussi connu sous le nom de *Monadic Second Order* ou *MSO*, est notamment utilisé dans un autre programme de M.Couvreur : VeriTaf. VeriTaf permet de vérifier des formules CTL (Computation Tree Logic) et des formules LTL (Linear Temporal Logic)

Références

[1] Bernard Boigelot. Lash toolset.

[2] Jean-Michel Couvreur. A bdd-like implementation of an automata package. In *Implementation and Application of Automata, 9th International Conference, CIAA 2004, Kingston, Canada, July 22-24, 2004, Revised Selected Papers*, pages 310–311, 2004.

[3] Seymour Ginsburg and Edwin Spanier. Semigroups, presburger formulas, and languages. *Pacific journal of Mathematics*, 16(2) :285–296, 1966.

[4] J.G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona : Monadic second-order logic in practice. In *Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS '95, LNCS 1019*, 1995.

[5] Nils Klarlund and Anders Møller. *MONA Version 1.4 User Manual*. BRICS, Department of Computer Science, Aarhus University, January 2001. Notes Series NS-01-1. Available from `http://www.brics.dk/mona/`. Revision of BRICS NS-98-3.