Mémoire PresTaf

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

19 mai 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 optimisé, plus claire et propre. Qui plus est, le Lua étant un langage basé sur le C++ cela offre la possibilité à l'utilisateur de surcharger des opérateurs. Ainsi il pourra utiliser les opérateurs pour écrire simplement ses formules logiques de façon simple.

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 implementant en Java la logique Arithmétique de Presburger [3]. Il existe des logiciels concurrent 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 les d'autre logiques sur les mots infini 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 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 monadique. La où PresTaf n'implémente à ce jour que la logique arithmétique de Presburger, la logique monadique pourrait être implémenter dans le futur.

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

4 Besoins fonctionnels

4.1 Prototype papier

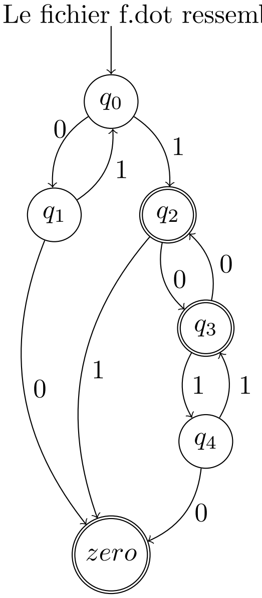
Le prototype qui suit serait un fichier Lua qui se servirait de la logique *Presburger* (cette logique étant elle-même codée en Lua).

```
pres = require('Presburger') // Choix de la logique

local x = variable('x') // Declaration d'une variable
local y = variable('y')
local f = equals(y + integer(1), x)

// Pour exporter l'automate
f:todot("f.dot")
```

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
```

Le fichier f.dot ressemblerai 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 : $0*(-2)^0 + 1*(-2)^1 + 0*(-2)^2 + 0*(-2)^3 + 1*(-2)^4 + 1*(-2)^5 + 0*(-2)^6 + 1*(-2)^7 = -146_{base-2}$

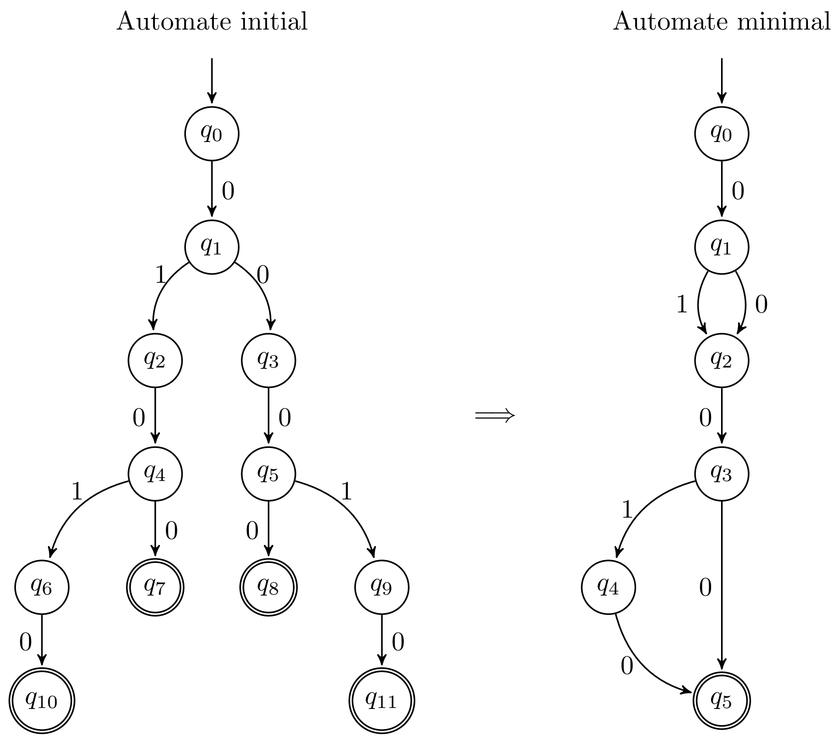
Justification : Il serait intéressant de permettre à l'utilisateur d'utiliser cette bilbiothèque avec diverses bases, surtout la base -2.

Priorité : 5

6 Prototypes et tests préparatoires

6.1 Blum

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



7 Exemple de fonctionnement

8 Architecture

L'architecture de notre logiciel est composée de 4 couches :

1. La bibliothèque PresTaf qui permet de manipuler des automates et d'éffectuer des opérations dessus. Nous avons créé une interface en java qui permet d'accéder plus simplement aux fonctions de la bibliothèques.
2. La première couche Lua est composée des différentes bibliothèques de logique utilisant la bibliothèque Prestaf. Nous avons développé la bibliothèque Presburger.
3. La deuxième couche Lua contient les différents fichiers des utilisateurs des bibliothèques de logique.
4. Le Java. C'est le launcher qui permet de démarrer l'application avec une unique commande tout en permettant d'utiliser du java et du Lua.

Ainsi, la première et quatrième couches de l'application font parties de la bibliothèque Prestaf. Il y a deux sortes d'utilisateurs. En effet, une partie des utilisateurs utiliseront directement notre bibliothèque pour développer leur propre bibliothèque de logique, tandis que l'autre partie des utilisateurs utiliseront les bibliothèques de logique. Un utilisateur peut également être utilisateur de sa propre bibliothèque. Le premier groupe d'utilisateur travaillera essentiellement sur la 2 eme couche de l'application, tandis que le second groupe travaillera sur la 3 eme couche.

Pour pouvoir utiliser le code Lua en Java nous avons utilisé la bibliothèque LuaJava.

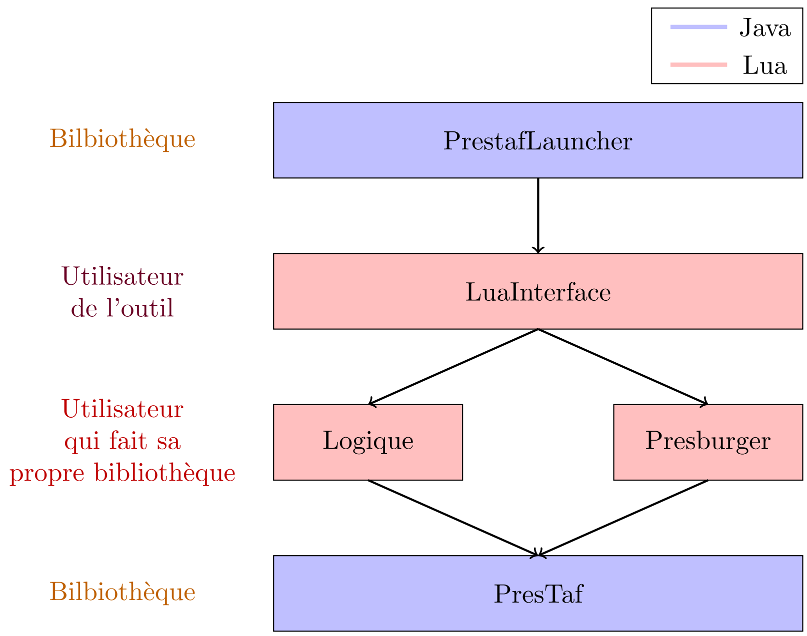


Figure 1 – Architecture type de PresTaf et son utilisation

9 Description et justification du code

9.1 Surcharge d'opérateurs

L'un des principaux avantages de Lua est d'être un langage de script qui permet la surcharge d'opérateur. On peut en sucharger 18. Nous avons donc surchargé les opérateurs $+ - *$ afin de permettre à l'utilisateur d'utiliser des opérateurs à la place de fonctions pour écrire ses formules Presburger.

Nous voulions également surcharger l'opérateur ==, cependant nous nous sommes rendus comptes que quelque soit la valeur de retour que nous mettions, elle est était toujours transformée en booléen. Nous avons donc choisi que cet opérateur ne retournerait pas de valeur mais placerait la valeur dans une variable, il faut donc une deuxieme fonction pour l'appeler.

En Lua, on peut également créé un opérateur de la forme $'operateur'$. Nous avons choisi de créer l'opérateur $' ='$ qui retourne la bonne valeur. L'utilisateur peut donc choisir entre l'opérateur == et l'opérateur $' ='$.

Pour l'égalité, l'addition, la soustraction et la multiplication de Presburger, l'utilisateur peut donc utiliser la fonction associée ou utiliser l'opérateur surchargé.

10 Analyser la compléxité

10.1 PresTaf

10.1.1 Minimisation d'automate

Presburger est une surcouche de Prestaf, il est donc directement lié à la complexité de Prestaf. L'une des complexités essentielles de Prestaf est celle de la minimisation d'automate. Un automate peut en effet avoir plusieurs milliers d'états. Nous avons commencé par étudier les algorithmes d'Hopecroft et Blum qui sont en $O(nlog(n))$. On sait que la complexité de Prestaf est en $O(n^2)$.

10.2 Presburger

D'après Derek C. Oppen [6] la complexité de Presburger est une tour d'exponentielle défini par la borne supérieure suivante : $O(2^{2^{2^{pn}}})$, où p est une constante $p > 1$ et n est la taille de l'entrée.

11 Tests de validations et fonctionnement

11.1 Test unitaires

Afin de vérifier que la validié de chacune des fonctions que nous avons écrites, nous avons comparé les versions Lua et Java de ces fonctions sur différents exemples, en affichant l'état des variables contenus dans ces fonctions.

11.2 Test globaux

Nous avons testé la bibliothèque Presburger sur différentes formules. Pour ce faire, nous avons comparé les graphes obtenus par la nouvelle version de Presburger et l'ancienne.

12 Description des extensions possibles

12.1 Complexité

Il y a toujours deux compléxités : celle de la bibliothèque logique implémenté, et celle de PresTaf.

12.2 Diversité des bibliothèques

PresTaf peut être amélioré en développant des bibliothèques de différentes logiques. Les possibilités à partir du même fichier LuaInterface seront donc augmentées.

Glossaire

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, $\forall y_1 \ldots \forall y_n [(P(0, y_1, \ldots, y_n) \vee \forall x(P(x, y_1, \ldots, y_n) \rightarrow P(x + 1, y_1, \ldots, y_n))) \rightarrow \forall y P(y, y_1, \ldots, y_n)]$

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.

[6] Derek C Oppen. A 222pn upper bound on the complexity of presburger arithmetic. *Journal of Computer and System Sciences*, 16(3) :323–332, 1978.