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 optimisé, 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 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 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 *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
```

4.2 Fonctions

4.2.1 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 : Besoin initial.

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 chaque fonction qui seront ensuite exécutés 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 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'appliqué 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 :

Priorité : 3

Nom : 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 utilisé que pour les variables des prédicats du premier ordre (de type Presburger par exemple).

Justification :

Priorité : 3

5 Besoins non fonctionnels

5.1 Fonctions

5.1.1 Machine visée

Description : Windows, MacOS, Linux

Justification :

Priorité : 1

5.1.2 Optimisation

Description :

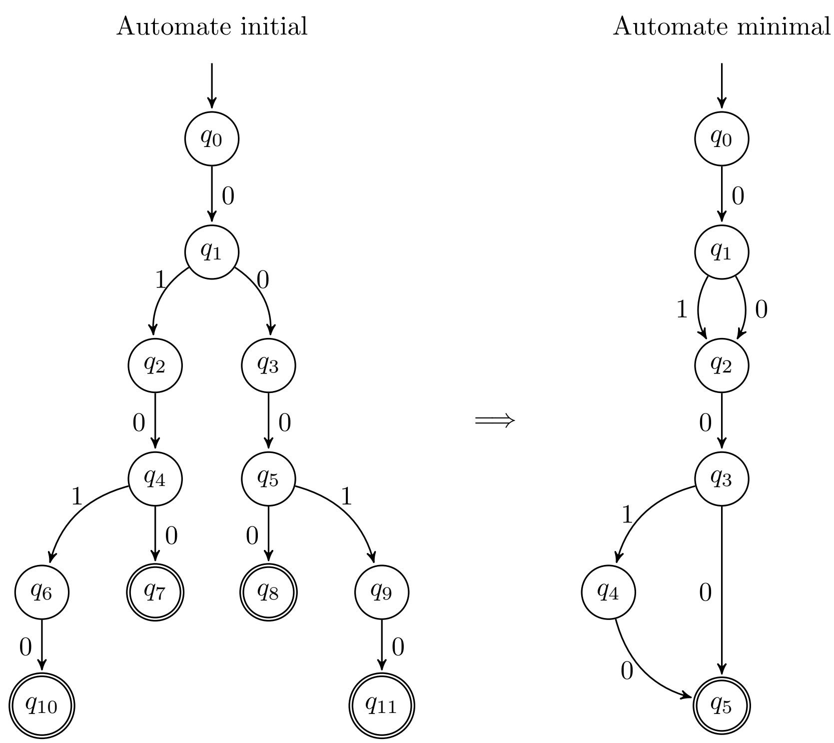
Justification :

Priorité : 1

6 Prototypes et tests préparatoires

6.1 Blum

Implémentation de l'algorithme de Blum en Python.
Voici un exemple que nous avons testé avec l'automate initial et l'objectif.



6.2 LuaJava

7 Planning

1. Implémation de l'algorithme de Blum.
2. Révision du code PresTaf.
3. Iterfaçage en Lua.
4. Rédaction du mémoire intermédiaire

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.

Glossaire

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

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

3

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 forumules LTL (Linear Temporal Logic) 3