UNIVERSITÉ DE NANTES

LABORATOIRE D'INFORMATIQUE DE NANTES ATLANTIQUE

Rapport de stage de fin de licence

Création d'un répertoire de logiciels d'inférence grammaticale

Auteur :
Hugo Mougard

Superviseur :
Colin De La Higuera

25 juin 2012

Table des matières

1 Introduction

Ce stage prend place dans le Laboratoire d'Informatique de Nantes Atlantique.

Son objet est de concevoir et réaliser un dépôt de logiciels d'inférence grammaticale, la discipline qui lui sert de cadre. Le but est, dans la mesure du possible, de parvenir à regrouper l'existant en contactant les principaux chercheurs concernés et en exploitant les ressources déjà présentes sur internet.

Pour que le résultat final soit utile au plus de personnes possibles, il faut documenter les programmes récupérés et les modifier quand cela est nécessaire afin qu'ils soient facilement utilisables.

Enfin, la ressource doit être organisée de manière à mettre en valeur les concepts importants de l'inférence grammaticale et à bien trier les algorithmes en fonction de leurs caractéristiques.

Ce sujet de stage peut entrainer des temps morts : s'il y a pénurie de logiciels à intégrer, il est prévu en parallèle un petit travail concernant le sujet de recherche suivant : l'apprentissage à partir de "morceaux" d'automate.

L'objet de ce travail est de s'intéresser à la problématique suivante : comment apprendre un langage en combinant de petits automates (inexacts) plutôt qu'en créant directement un grand automate ?

Ce sujet n'a été introduit dans sa totalité que 3 semaines avant la fin du stage.

Le rapport présentera d'abord le cadre du stage : le laboratoire et l'équipe d'accueil, puis le domaine : l'inférence grammaticale. Enfin le travail sera présenté en trois parties : une rapide sur les lectures d'introduction, et deux plus détaillées sur chacun des sujets du stage.

2 Présentation du LINA

Le LINA (Laboratoire d'Informatique de Nantes Atlantique) regroupe 10 équipes de recherche et emploie environ 170 personnes.

Les principaux thèmes des recherches qui y sont menées sont :

- Architectures logicielles distribuées :
  - Gestion de données ;
  - Génie logiciel.
- Systèmes d'aide à la décision :
  - Contraintes et optimisation ;
  - Extraction de connaissance ;
  - Bio-informatique ;
  - Thème langage naturel.

Les 10 équipes du LINA sont :

- AeLoS - **A**rchitectures **e**t **Lo**giciels **S**ûrs
- ASCOLA - **AS**pect and **CO**mposition **LA**nguages
- AtlanMod - **Atlan**tic **Mod**eling
- COD - **CO**nnaissances et **D**écision
- COMBI - **COM**binatoire et **B**io-**I**nformatique
- GDD - **G**estion de **D**onnées **D**istribuées
- GRIM - **G**estion, **R**ésumé, **I**nterrogation et apprentissage sur les **M**asses de données
- OPTI - **Opti**misation globale, optimisation multi-objectifs
- TALN - **T**raitement **A**utomatique du **L**angage **N**aturel
- TASC - **T**héorie, **A**lgorithmes et **S**ystèmes en **C**ontraintes

3 Présentation de l'équipe TALN

Ce stage prend place au sein de l'équipe TALN. Cette équipe compte actuellement 21 membres et ses principaux projets en cours sont :

- TTC - **T**erminology Extraction, **T**ranslation Tools and **C**omparable Corpora
  Vers l'automatisation de la production de terminologie multilingue à partir de corpus comparable pour la traduction automatique et les outils d'aide à la traduction

- DEPART - **D**ocuments **E**crits et **Pa**roles - **R**econnaisance et **T**raduction
  Ce projet vise la constitution au niveau de la région des Pays de la Loire d'un pôle de compétences unique en France associant analyse du signal audio et manuscrit au traitement automatique des langues. Il s'intéresse plus particulièrement à la résolution de problèmes scientifiques et technologiques difficiles mettant en jeu des données multimodales et multilingues.

- MeTRICC - **Me**moire de **T**raduction, **R**echerche d'**I**nformation et **C**orpus **C**omparables
  Le projet MeTRICC vise à utiliser des corpus comparables en vue de l'extraction de lexiques multilingues dans le cadre des mémoires de traduction et de la recherche d'informations interlingue.

Matthieu Vernier et Fabien Poulard, deux membres de l'équipe TALN, ont co-fondé une entreprise, Dictanova, qui est hébergée au sein du LINA.

4 Présentation de l'inférence grammaticale

L'objet de l'inférence grammaticale est de retrouver la structure d'un langage à partir d'un certain nombre de ses éléments.

Exemple Considérons l'échantillon constitué des trois exemples suivants :
- aab
- aaaab
- aaaaaab

L'automate qui reconnait ce langage est le suivant :

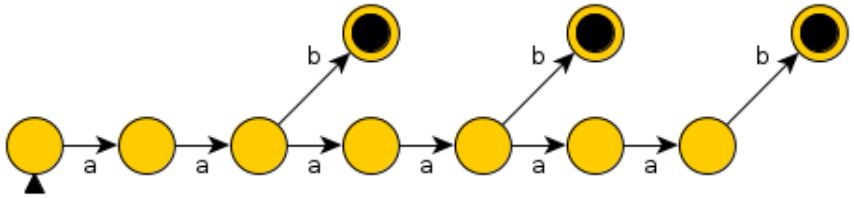

FIGURE 1 – L'automate "non général" - ne permet pas d'extrapoler à des exemples non recontrés jusque là

Toutefois, on se rend assez vite compte que cet automate ne reconnait que les trois échantillons donnés. La structure du langage n'a donc pas été apprise. L'application des techniques d'inférence grammaticale dans ce cas pourrait nous permettre d'apprendre :

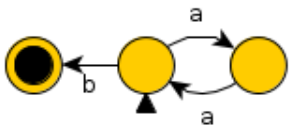

FIGURE 2 – Une des cibles possibles...

Avec ce nouvel automate, les chaînes suivantes seraient aussi reconnues :

- b
- aaaaaaaab

Il y a donc généralisation. Cet exemple illustre bien le but de l'inférence grammaticale : dégager des règles derrière un ensemble donné d'exemples.

5 Lectures d'introduction

Afin de découvrir la discipline, quelques jours ont été accordés à la lecture d'articles sur le thème de l'inférence grammaticale. Voici les principaux :

- les tutoriels [7, 5, 1] introduisent les notions de base de la discipline
- les articles [11, 12] détaillent des notions théoriques fondamentales
- le livre [6] contient quasiment tous les détails nécessaires quand un doute persiste sur une notion.

De même les sites des principaux chercheurs de la discipline ont été consultés afin d'y trouver d'autres articles et des pointeurs vers des ressources potentiellement intéressantes. Afin de ne pas polluer la bibliographie avec de trop nombreuses entrées ils ne sont pas détaillés.

Pour finir la découverte de la discipline, les sites des différentes compétitions [14, 16, 18, 4, 3, 2] ainsi que d'autres projets du même type ont été visités [15].

6 Grammatical Inference Software Repository

6.1 Présentation et état de l'art

La première partie du stage concerne donc la création d'un répertoire de logiciels d'inférence grammaticale. Ce répertoire est intéressant dans le sens où l'inférence grammaticale est un champ abstrait, à la croisée des disciplines. On peut la retrouver dans :

- la sécurité : détection des virus ;
- la génétique ;
- le traitement des langues ;
- des problèmes d'optimisation ;
- la compression de données ;
- etc.

Cela a pour conséquence que les idées développées dans le domaine de l'inférence grammaticale sont toujours assez "éloignées" de leurs implémentations : en plus de l'algorithme d'inférence grammaticale choisi, pour développer une application, il faut toute la logique spécifique au domaine.

De ce fait, peu de logiciels sont développés dans ce domaine de recherche et il est difficile de trouver des exemples d'implémentation de certains algorithmes. L'objet de ce projet est donc de pallier ce manque en créant une ressource réunissant l'existant (dans la mesure du possible).

L'état de l'art est donc simple : il n'existe pas à l'heure actuelle, à notre connaissance, de "forge" dédiée à l'inférence grammaticale. Ce projet constituera donc, nous l'espérons, un bon point de départ à l'échange de programmes dans cette discipline.

6.2 Point de départ

La plupart des programmes réalisés à ce jour en inférence grammaticale le sont pour participer aux principales compétitions organisées pour faire avancer la discipline. Ces compétitions sont :

- Abbadingo [14], 1997 ;
- Gecco [16], 2004 ;
- Omphalos [18], 2004 ;
- Zulu [4], 2010 ;
- Stamina [3], 2010 ;
- Pautomac [2], 2012.

Passés ces programmes, il faut demander aux chercheurs ce qu'ils ont gardé et ce qu'ils veulent partager.

6.3 Architecture de la solution retenue

Pour mettre en valeur les programmes récupérés, il a été décidé d'utiliser Redmine [13] (logiciel de gestion de projets) et de garder le code sur un serveur Subversion [9] (logiciel de gestion de versions).

Ainsi il est possible de renseigner un wiki pour chaque logiciel et d'avoir tous les bénéfices d'un logiciel de gestion de projet si nécessaire : gestion des bugs, des demandes d'évolution, des droits d'administration, etc. De la même manière, pour le code, Subversion étant très utilisé dans le monde du développement, c'était un choix correct pour héberger nos programmes.

6.4 Exemple de traitement d'un programme

Pour éviter trop de répétitions, le traitement d'un seul programme va être détaillé (le répertoire final compte une dizaine de programmes).

Repérage

En utilisant le serveur Gowachin [15], un serveur d'entrainement aux tâches d'inférence grammaticale, il apparaissait que des tests récents avaient été faits sur de grands automates avec une réussite honorable par Alban Batard.

Il s'est avéré après discussion avec Colin De La Higuera qu'il était un de ses anciens stagiaires.

Prise de contact

L'étape suivante a été de contacter Alban Batard afin de savoir s'il avait du code à partager et s'il était prêt à le rendre public. Ce dernier a partagé deux programmes : l'un était une implémentation d'un des algorithmes classiques, RPNI, l'autre était une parallèlisation de cet algorithme, nommée PSMA.

Traitement du programme

Le code en main, il fallait maintenant voir si celui-ci était exploitable. Cela consiste en plusieurs points :

- les méthodes et librairies utilisées sont toujours disponibles et non obsolètes ;

– le déploiement est facile pour un utilisateur lambda ;
– le code est lisible ;
– le code est “correct”.

Note : le fait que le code soit en mauvais état ne veut rien dire quant aux qualités en développement de l'auteur, souvent les programmes récupérés n'avaient pas pour but d'être relus, encore moins publiés.

Reprenons chacun de ces points pour voir le travail effectué :

Méthodes et librairies obsolètes Dans le programme récupéré, la structure de données Java `Hashtable` était utilisée.

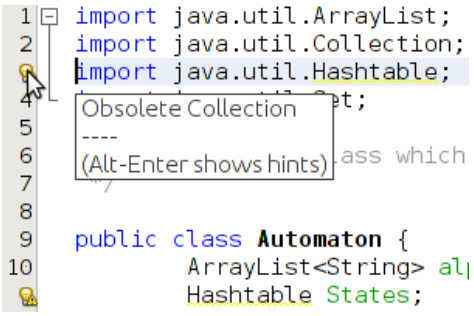

FIGURE 3 – Structure de données obsolète

Cette structure de données étant obsolète, il a fallu la remplacer par la structure `HashMap` correspondante.

Déploiement facile Deux points ont été traités.

Script de déploiement Ce programme n'avait pas de script de déploiement. Ainsi il fallait tout compiler “à la main” et connaître les détails de ce processus en Java pour faire tourner le logiciel.

Un scrit Ant [8] a été proposé pour pallier ce manque. Il est donc maintenant possible de construire le logiciel avec une simple commande :
`ant dist`

Données entrées “en dur” Certains paramètres étaient directement renseignés dans le code, comme le répertoire où se trouvaient les fichiers à utiliser :

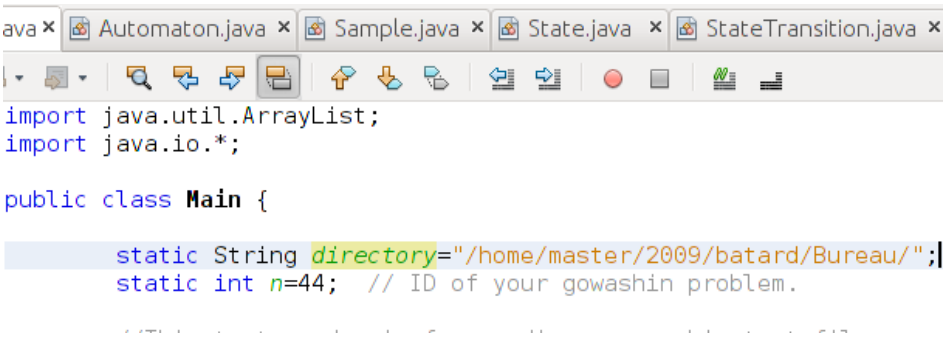

FIGURE 4 – Données hardcodées

Ces données entrées en dur ont été remplacées par une vraie implémentation des paramètres. Ainsi maintenant le code s'utilise de la manière suivante :

```
$ java -jar RpniJava.jar train.txt test.txt output.txt
```

Où les paramètres (`train.txt`, `test.txt` et `output.txt`) sont explicitement donnés au programme.

Code lisible Le code proposé ne respectait pas certains standards :

- une ligne ne fait pas plus de 80 caractères ;
- l'indentation Java est respectée.

Ces points ont été rectifiés.

Code correct De même, quelques standards de programmation n'étaient pas respectés :

- les structures de données n'était pas paramétrées quand elles pouvaient l'être, entraînant des casts partout dans le code ;
- les classes n'étaient pas dans un paquetage.

Ils ont donc fait l'objet de modifications.

Dépôt du code sur le serveur Subversion

Afin de pouvoir récupérer le code facilement depuis le REDMINE, il faut créer un dépôt pour chaque programme. Un dépôt SUBVERSION est très proche d'une arborescence de fichiers classique.

Déterminer la licence du programme

Bien souvent (90% des cas), le code fourni n'a pas de licence. C'est un problème car n'étant pas les auteurs de ces programmes, le copyright ne

nous appartient pas.

Une fois le code prêt à être mis en ligne, il est donc demandé aux auteurs de choisir parmi les 3 grandes solutions suivantes :

- GNU Public License [10], solution militante pour le logiciel libre, le code distribué ne pourra être utilisé que dans des projets libres ;
- MIT License [17], solution plus permissive que la GPL, permettant d'inclure le programme dans des projets commerciaux ou non libres ;
- Domaine public [20], solution qui renonce au copyright.

Dans le cas d'Alban Batard, la GPL a été choisie, il a donc fallu modifier tous les fichiers pour y inclure la mention de licence.

Création de la documentation

Une fois le programme rendu exploitable et la licence connue, il convient de produire la documentation qui sera mise sur le wiki du Redmine. Cette documentation regroupe plusieurs informations importantes :

- auteur
- langage utilisé
- licence
- manuel d'utilisation

Greffe du sous-projet au projet central

Afin de donner de la cohérence au répertoire, il faut trier et relier de manière cohérente tous les projets entre eux, c'est ce qui est fait dans le projet central du Redmine.

6.5 Publication

Le dépôt est prêt à être publié mais attend actuellement une URL publique (qui s'obtient auprès des services compétents de l'Université de Nantes). Le but est qu'il soit disponible pour l'ICGI 2012, grande réunion de la discipline.

7 Stubs automata

La dernière partie du stage, menée en parallèle pendant les temps morts de GISR - c'est à dire quand il n'y avait pas de logiciel à intégrer au dépôt - au cours des 3 dernières semaines, concerne, de manière intuitive, l'apprentissage d'un langage par "petits bouts" plutôt que de manière frontale.

Le principal article étudié [19] nécéssitait des prérequis qui ont rendu la lecture difficile et peu effective.

Le but serait, au lieu d'apprendre une règle compliquée telle que :

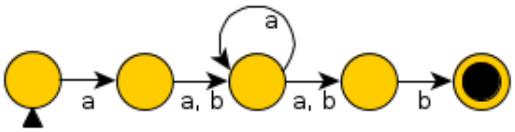

FIGURE 5 – Règle compliquée : commence par un `a`, a une séquence de `a` et se termine par un `b`

d'apprendre 3 règles simples à la place :

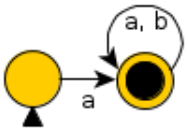

FIGURE 6 – Règle simple 1 : commence par un `a`

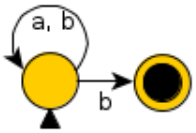

FIGURE 7 – Règle simple 2 : se termine par un `b`

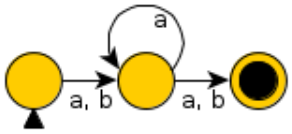

FIGURE 8 – Règle simple 3 : contient une séquence de `a`

Cette étude n'ayant pas beaucoup progressé durant le peu de temps qui lui a été consacré, les recherches ne seront pas détaillées outre mesure. Les principaux axes investigués ont été les suivants :

7.1 Construction des "petits automates"

- générer à partir d'exemples positifs du langage les automates généraux minimaux et les spécialiser avec les exemples négatifs ;
- construire tous les automates possibles en partant des plus généraux et en allant vers les plus spécialisés grâce à un treillis d'automates ;
- comme ci-dessus mais avec des arbres d'expressions rationnelles.

Pour les deux derniers cas, il est intéressant de voir comment réduire l'arbre des possibles pour cibler directement les automates intéressants. C'est ce besoin qui a motivé l'utilisation d'arbres plutôt que d'automates, ceux-ci sont plus facilement étudiés les uns vis à vis des autres.

7.2 Combiner les "petits automates"

La question porte ici sur la nature des combinaisons de petits automates qui seraient intéressantes pour apprendre des langages évolués : l'exemple donné au début utilise le produit d'automates, mais l'union, et des systèmes de vote plus évolués sont envisageables. Ce domaine emprunte beaucoup à l'apprentissage automatique et ses "weak learners".

8 Conclusion

Ce stage avait pour but de faire découvrir le monde de la recherche. C'est une réussite :

- la principale partie du stage, GISR, a permis d'entrer en contact avec de nombreux chercheurs aussi bien directement pour récupérer des programmes qu'indirectement au cours des recherches préliminaires ;
- la seconde partie du stage, quant à elle, a mis en valeur le type de travaux auquel on peut être confronté dans une activité de recherche, avec ses difficultés et ses attraits ;
- enfin la vie en laboratoire de recherche a elle aussi été explorée au sein du LINA.

De plus, GISR, s'il est diffusé de manière suffisante, permettra de faire avancer à sa manière l'inférence grammaticale, peut-être ne serait-ce qu'en donnant aux chercheurs de la discipline l'envie de partager leurs codes, même si au final d'autres moyens sont utilisés.

Le petit travail de recherche quant à lui a bien permis de soulever les problèmes rencontrés durant une activité de recherche : difficulté à tester ses idées, difficulté à cerner exactement le résultat que l'on recherche, etc. Cela aura donc été formateur, même si le temps consacré à cette partie du stage n'a pas vraiment permis de dépasser la phase de prise de contact avec le problème.

Pour ce qui est des perspectives liées à ce stage, elles sont doubles (de part la nature du stage) :

- une occasion a été manquée de publier un papier parlant de GISR, mais ce type d'initiative peut potentiellement attirer de nouvelles contributions et serait donc la continuation logique de la démarche du projet ;
- concernant la recherche, elles sont vastes : travailler la méthodologie, rechercher des problèmes intéressants, tout n'est encore qu'ouverture...

Bibliographie

[1] *Tutorial Notes—Formal and Empirical Grammatical Inference*. Association for Computational Linguistics, 2011.

[2] Hasan Ibne Akram, Rémi Eyraud, Jeffrey Heinz, Colin de la Higuera, James Scicluna, and Sicco Verwer. Pautomac. http://ai.cs.umbc.edu/icgi2012/challenge/Pautomac/, 2012. [En ligne ; visité le 20 Juin 2012].

[3] Kirill Bogdanov, Christophe Damas, Pierre Dupont, and Neil Lambeau, Bernardand Walkinshaw. Stamina. http://stamina.chefbe.net/, 2010. [En ligne ; visité le 20 Juin 2012].

[4] David Combe, Colin de la Higuera, Jean-Christophe Janodet, and Myrtille Ponge. Zulu. http://labh-curien.univ-st-etienne.fr/zulu/index.php, 2010. [En ligne ; visité le 20 Juin 2012].

[5] Colin de la Higuera. Learning finite state machines. Technical report, FSMNLP, 2009.

[6] Colin de la Higuera. *Grammatical inference : learning automata and grammars*. Cambridge University Press, 2010.

[7] Pierre Dupont and Laurent Miclet. Inférence grammaticale régulière : fondements théoriques et principaux algorithmes. Technical Report RR-3449, INRIA, 1998.

[8] Apache Foundation. Ant. http://ant.apache.org/, 2000. [En ligne ; visité le 20 Juin 2012].

[9] Apache Foundation. Subversion. http://subversion.apache.org/, 2000. [En ligne ; visité le 20 Juin 2012].

[10] Free Software Foundation. Gnu public license. http://www.gnu.org/licenses/gpl.html, 2007. [En ligne ; visité le 20 Juin 2012].

[11] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5) :447–474, 1967.

[12] E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3) :302–320, 1978.

[13] Jean-Philippe Lang. Redmine. http://www.redmine.org/, 2006. [En ligne ; visité le 20 Juin 2012].

[14] Kevin J. Lang and Barak A. Pearlmutter. Abbadingo. http://www-bcl.cs.may.ie/, 1997. [En ligne ; visité le 20 Juin 2012].

[15] Kevin J. Lang, Barak A. Pearlmutter, and François Coste. Gowachin. http://www.irisa.fr/Gowachin/, 1998. [En ligne ; visité le 20 Juin 2012].

[16] Simon M. Lucas. Gecco. http://cswww.essex.ac.uk/staff/sml/gecco/NoisyDFA.html, 2004. [En ligne ; visité le 20 Juin 2012].

[17] MIT. Mit license. http://www.opensource.org/licenses/MIT, 1988. [En ligne ; visité le 20 Juin 2012].

[18] Brad Starkie, François Coste, and Menno van Zaanen. Omphalos. http://www.irisa.fr/Omphalos/, 2004. [En ligne ; visité le 20 Juin 2012].

[19] Frédéric Tantini, Alain Terlutte, and Fabien Torre. Combinaisons d'automates et de boules de mots pour la classification de séquences. *Revue d'Intelligence Artificielle*, 2011. preprint.

[20] Unknown. Unlicense. http://unlicense.org/, 2010. [En ligne ; visité le 20 Juin 2012].