

Laboratoire LISTIC
Polytech Annecy Chambéry
5, chemin de bellevue
74944 Annecy-Le-Vieux

COLLET Benjamin
IUT d'Annecy le vieux
DUT INFO
Année 2015/2016

Rapport de stage :
Traducteur prothétique-Gallina et
interface de gestion d'expressions logiques

DAPOIGNY Richard

DELOULE Françoise

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidés lors de la rédaction de ce rapport.

Tout d'abord, j'adresse mes remerciements à mon professeur ainsi que tuteur de stage, M. Richard DAPOIGNY qui m'a incité à postuler dans son laboratoire de recherche et m'a permis d'obtenir ce stage.

Je tiens à remercier vivement M. Patrick BARLATIER, qui a été mon interlocuteur principal pour la spécification du projet, le temps passé ensemble et le partage de son expertise. Grâce aussi à sa confiance j'ai pu m'accomplir totalement dans mes missions.

Je remercie également toute l'équipe du LISTIC pour leur accueil, ainsi que les stagiaires présents, pour leur esprit d'entraide.

Enfin, je tiens à remercier toutes les personnes qui m'ont conseillé et relués lors de la rédaction de ce rapport de stage, notamment ma belle-mère Élisabeth PIGA.

Sommaire

Remerciements	1
Introduction	3
1 Présentation du LISTIC	4
1.1 Implantation	4
1.2 Le personnel	5
1.3 Activités et services	6
1.4 L'équipe de travail	7
2 Présentation du besoin	8
2.1 Analyse du besoin	8
2.2 Listing des fonctionnalités	9
2.3 Moyens techniques et outils envisagés	9
3 Étude et réalisation	11
3.1 Planification	11
3.2 1 ^{re} partie, le traducteur	12
3.3 2 ^{de} partie, l'interface	16
3.4 Travail restant	18
4 Bilan	19
4.1 Bilan professionnel	19
4.2 Bilan personnel	20
Conclusion	21
Annexes	22

Introduction

Étudiant à l'Institut Universitaire de Technologie d'Annecy-le-Vieux dans le département INFO, j'ai eu la chance de faire mon stage dans le laboratoire de recherche LISTIC.

J'ai proposé ma candidature à M. Richard DAPOIGNY qui a bien voulu y donner suite et qui a donc naturellement été mon tuteur.

Ma mission consistait à réaliser un traducteur de théorèmes logiques depuis la Protothétique vers du Gallina. La Protothétique est un système de logique créé par LEŚNIEWSKI¹, qui a aussi créé un système d'écriture spécifique (« symbolisme authentique »). Gallina est le langage de Coq, un logiciel assistant de preuve. Ce traducteur a ensuite été intégré dans une interface graphique destinée à la gestion de ces théorèmes.

Cette mission me semblait relativement difficile et surtout très différente des autres stages, ce qui fut une motivation supplémentaire.

Je vais dans un premier temps présenter le laboratoire. Ensuite, je présenterai le besoin à l'origine de mon stage, ainsi que les fonctionnalités et moyens techniques imaginés pour répondre à ce besoin. La partie suivante, présentera ma mission et sa réalisation, de la conception aux tests. Enfin, dans la dernière partie, je ferai le bilan de mon stage, tant du point de vue professionnel que personnel.

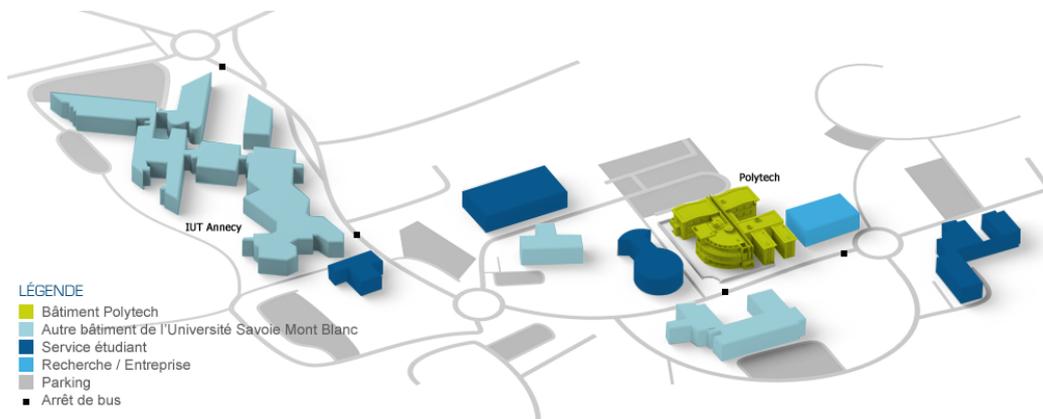
1. Stanisław LEŚNIEWSKI (1886–1939) est un mathématicien, philosophe et logicien polonais

1 Présentation du LISTIC

1.1 Implantation

Le Laboratoire d'Informatique, Systèmes, Traitement de l'Information et de la Connaissance (LISTIC), a été créé en 2003 dans l'optique de la recherche sur la fusion d'informations. La fusion d'informations constitue une chaîne de traitement depuis l'élaboration de l'information – à partir de données et de modèles – jusqu'à son exploitation – pour l'analyse ou le contrôle de systèmes réels.

Le laboratoire est implanté à Annecy-le-Vieux en Haute-Savoie, dans la région Auvergne-Rhône-Alpes. Il se situe dans le bâtiment de Polytech, sur le campus d'Annecy-le-Vieux, à quelques minutes à pied de l'IUT.



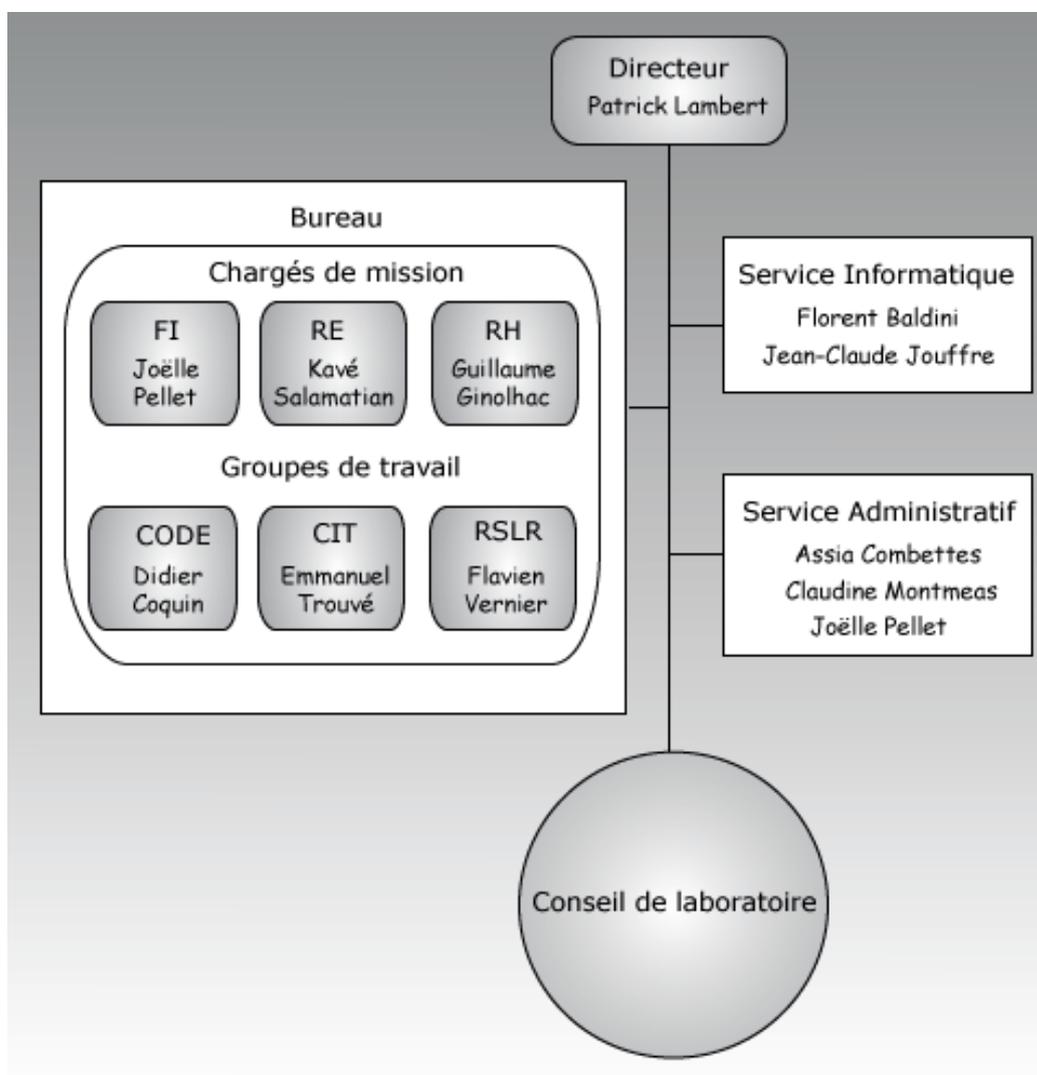
Cette implantation n'a pas influencé le choix de mon stage mais s'est révélée très pratique par la suite, le campus proposant de nombreux services pour les étudiants (notamment pour la restauration).

1.2 Le personnel

Le laboratoire est composé d'environ 70 personnes :

- 35 enseignants-chercheurs
- 18 doctorants en interne et 7 co-encadrés
- 5 administratifs et techniques
- 4 collaborateurs bénévoles

Voici l'organigramme du laboratoire, la cohésion et la qualité des recherches sont assurées par le conseil du laboratoire².



2. cf. annexe A.1

1.3 Activités et services

Le LISTIC est réparti dans trois groupes de travail, CODE : COmbinaison et DÉcision, CIT : Connaissance, Images et Télédétection et RSLR : Réseaux et Systèmes Logiciels Répartis.

Le LISTIC est réparti en trois groupes de travail :

- Connaissance, Images et Télédétection (CIT) :
Spécialisé dans les méthodes de traitement du signal (en particulier des images mais pas exclusivement), l'analyse vidéo, la fouille de données et l'extraction de la connaissance ainsi que l'apprentissage automatique. Il comprend aussi des chercheurs qui travaillent sur le raisonnement et les connaissances, les ontologies, la terminologie et la sémantique formelle.
- COmbinaison et DÉcision (CODE) :
Ce groupe est principalement axé sur les outils de représentation d'informations imparfaites. Plus particulièrement les intervalles, sous-ensembles flous, probabilités, possibilités, fonction de croyance ainsi que des méthodes d'agrégation et de fusion d'informations hétérogènes pour l'aide à la décision en contexte dynamique incertain.
- Réseaux et Systèmes Logiciels Répartis (RSLR) :
Ce dernier recherche les mécanismes logiciels adaptés aux architectures multi-cœurs, au routage IP haute performance, à la détection d'intrusions. En découle des compétences sur la qualité de service des systèmes logiciels, les systèmes logiciels dynamiques, services, cloud computing et les sciences de l'Internet et des réseaux.

L'activité du laboratoire étant reliée aux groupes explicités précédemment, les doctorants et chercheurs du laboratoire ont des compétences dans les domaines du traitement de l'information (données, signal, image, vidéo), des systèmes logiciels répartis, du développement logiciel, de l'expression de la performance, de l'amélioration continue, de l'analyse de grandes masses de données ainsi que celui de la représentation des connaissances, et cette liste, n'est pas exhaustive.

Les projets du LISTIC sont choisis pour être étudiés par les trois groupes de travail, sous des angles différents en fonction des compétences présentent au sein de chacun. Une des forces du LISTIC est donc d'aborder les projets de multiple façon et de mettre en évidence des compétences contextes qui permettent de résoudre des problèmes plus rapidement ou plus en profondeur.

1.4 L'équipe de travail

Durant ce stage, j'ai travaillé avec l'équipe Raisonnement et Connaissances du groupe CIT. Celle-ci a pour objectif de trouver une théorie mathématique pour la représentation des connaissances (*subsumptions* et *partOf relations*) de formes géométriques floues dans un contexte donné (l'implémenter en Coq et réaliser des inférences de manière automatique). Cette équipe est composée de Richard DAPOIGNY et de Patrick BARLATIER, tout deux chercheurs ainsi qu'enseignants à l'IUT d'Annecy.

C'est Patrick BARLATIER qui a assuré la maîtrise d'ouvrage durant ma mission. C'est lui qui a défini le projet, et qui fut donc mon interlocuteur principal. Nous avons énormément échangé par mail, et je devais notamment lui faire un rapport journalier pour l'informer de l'avancement de mon travail. Nous avons aussi une réunion hebdomadaire pour discuter plus longuement sur le projet et discuter de la suite de la mission de de mes objectif pour la semaine suivante. Notre méthode de travail se rapproche des méthodes agiles du fait que si le projet était globalement défini avant le stage, les exigences semainières pouvait facilement être adapté à l'avancement de la mission et au temps disponible.

2 Présentation du besoin

2.1 Analyse du besoin

S. LEŚNIEWSKI a proposé un système logique (d'ordre supérieur) fondé sur l'équivalence et les fonctions propositionnelles comme une alternative à la théorie des ensembles. Ce système appelé protothétique est une théorie mathématique ontologiquement neutre qui comprend plus de 400 théorèmes, quelques définitions et seulement trois axiomes.



MM. Richard DAPOIGNY et Patrick BARLATIER ont partiellement spécifiés la protothétique et les autres systèmes de Lesniewski en Coq. Ce travail est disponible sous forme d'une librairie disponible sur le site de l'INRIA (plus de 2000 lignes de code fonctionnel).

Une des applications possible de ce travail est la méréogéométrie de Tarski qui sera utilisée dans le cadre de la thèse Meriem HAFSI pour conceptualiser des raisonnements spatiaux symboliques.

Le principal problème est que les théorèmes de la protothétique écrits par LEŚNIEWSKI utilisent des symboles spécifiques ainsi que la notation polonaise. Cela a pour conséquence que personne n'a investigué profondément la protothétique principalement à cause du symbolisme authentique spécifié par LEŚNIEWSKI.

La solution est de proposer un compilateur permettant de passer du symbolisme authentique et ses relations au symbolisme logique standard (proposer par RUSSELL) avec ses relations. Ce fut donc le sujet de mon stage : dans un premier temps de proposer un traducteur, et dans un second une interface graphique qui implémente ce traducteur ainsi que le symbolisme authentique.

2.2 Listing des fonctionnalités

Le travail le plus important fut de traduire les expressions logiques – axiomes, lemmes, définitions – depuis le symbolisme authentique de LEŚNIEWSKI vers du Gallina, utiliser par Coq pour créer des preuves, plus proche du symbolisme de RUSSELL. Traduire une logique est plus facile qu’une langue naturelle car une traduction exacte existe, mais les exigences ne sont pas les mêmes. En effet, une traduction approximative n’est pas acceptable dans le cas d’une logique mathématique. En fait, cela s’apparente plus à une compilation qu’à une traduction au sens habituel du terme.

Les autres fonctionnalités relèvent de l’interface avec l’utilisateur. La première est de pouvoir saisir « visuellement » les expressions, cela implique d’intégrer le symbolisme authentique à l’interface et d’implémenter un système visuel pour insérer les caractères spéciaux. L’objectif de cette fonctionnalité est que l’utilisateur puisse recopier facilement une expression depuis un livre pour s’en servir dans une preuve (assisté par Coq), et cela sans avoir recours à une syntaxe intermédiaire. Enfin, notre logiciel doit être capable de stocker ces expressions. La protothétique, ainsi que les logiques qui s’appuient dessus, est un système complet, il est donc intéressant d’avoir accès aux expressions déjà démontrées lorsqu’on traduit une nouvelle.

2.3 Moyens techniques et outils envisagés

Pour pouvoir créer un traducteur, le couple d’outils Lex/Yacc³ semble presque obligatoire. Ces derniers étant implémentés dans de nombreux langages, le choix restait ouvert. Un langage fonctionnel semblait approprié et, dans l’optique d’une future interface graphique, le langage choisi dans un premier temps fut F#, principalement pour son intégration au *framework* .NET de Microsoft. Après une semaine de test sur un problème moins complexe (une calculatrice), nous avons décidé de changer de langage. Notre second choix s’est orienté vers OCaml, le langage de l’INRIA qui a servi de base à F#. Le principal avantage d’OCaml par rapport à F# est la simplicité de mise en œuvre de la technologie.



3. respectivement générateur d’analyseurs lexicaux et d’analyseurs syntaxiques



Pour l'interface, le choix a été beaucoup plus simple. La seule contrainte étant que le logiciel puisse s'exécuter sur Windows, les technologies proposées par Microsoft ont donc été privilégiées. Le *framework* .NET est le plus gros outil de Microsoft à la disposition des développeurs. Celui-ci contient notamment Windows Forms, une bibliothèque logicielle d'interface graphique très populaire. Néanmoins, j'ai décidé d'utiliser Windows Presentation Foundation, faisant partie elle aussi de .NET. Destiné à remplacer Windows Forms, cette technologie plus moderne apporte un lot d'avantages, comme une meilleure séparation du code entre interface et traitement ou une gestion plus simple du *responsive design*.

3 Étude et réalisation

3.1 Planification

La planification du projet ne s'est pas faite à l'aide d'un diagramme de Gantt mais a l'aide du logiciel ToDoList. Comme son nom l'indique, ce logiciel permet de gérer une liste de tâches à effectuer. Cet outil open source est assez complet et permet notamment de générer un diagramme de Gantt, mais dans notre cas, l'enchaînement temporel des différentes parties de la mission n'avait que peu d'importance. En effet, une liste des tâches avec des durées estimées ainsi que des priorités s'est montrée suffisante. Au moment où j'écris ce rapport, la mission n'est pas totalement terminée comme on peut le remarquer sur la capture de l'utilitaire.



Titre [Interface de gestion des Définiti... \Créatio...	!	%	Est.	Passé	⊖	Prévu
<input checked="" type="checkbox"/> Parseur Protothétique/Coq				11,00 D		
<input checked="" type="checkbox"/> Traduction basique (A1 et A2)				2,00 D		
<input checked="" type="checkbox"/> Détection des fonctions				5,00 D		
<input checked="" type="checkbox"/> Amélioration esthétique				2,00 D		
<input checked="" type="checkbox"/> Traduction des fonction selon un fic...				2,00 D		
<input type="checkbox"/> Interface de gestion des Définitio...	5	74%	8,18 D	16,00 D		
<input checked="" type="checkbox"/> Communication entre OCaml et C#				3,00 D		
<input type="checkbox"/> Création de l'interface	8	87%	2,78 D	12,00 D		
<input checked="" type="checkbox"/> Fenêtre de saisie				4,00 D		
<input type="checkbox"/> Fenêtre de gestion	8	60%	2,00 D	4,00 D		
<input checked="" type="checkbox"/> Modélisation des données (class...				2,00 D		
<input type="checkbox"/> Modélisation des données (XML)	2	10%	5,40 D	1,00 D		

3.2 1^{re} partie, le traducteur

Le traducteur – entre le symbolisme authentique de la protothétique et Gallina – fut la première moitié du stage, certainement la plus technique et la plus originale. J'utiliserai parfois le mot compilateur pour me référer à ce traducteur.

3.2.1 Une première base

En quelques jours seulement, une première version fonctionnelle du compilateur. Cette version, évidemment limitée, permettait déjà de traduire les deux premiers axiomes⁴.

Pour simplifier la suite de ce rapport, il faut que je présente une base technique du fonctionnement d'un compilateur, les prochains paragraphes ont pour but d'expliquer grossièrement la méthode utilisée pour programmer ce traducteur.

Un compilateur se compose d'un analyseur lexical et d'un analyseur syntaxique.

Analyse lexicale

La première partie du traducteur est l'analyseur lexical, c'est le module le plus simple des deux. En effet, il s'agit simplement d'un rechercher/remplacer avec des expressions rationnelles⁵.

Pour la suite, j'utiliserai un exemple simple : $\llbracket ab \rrbracket \overset{\circ}{\phi}(ab)$. Pour des raisons de compatibilité et surtout parce que ce caractère, $\overset{\circ}{\phi}$, n'existe pas dans unicode⁶, j'ai transformé l'exemple de la façon suivante : [ab | eqP(ab)]. L'analyseur lexical va simplement remplacer un par un chaque caractère (parfois un groupe de caractères) par un *token*. Pour l'exemple, la suite de *token* en sortie de l'analyseur sera : START PROP PROP MIDDLE FUNC LPAREN PROP PROP RPAREN END.

4. cf. annexe A.2

5. aussi appelé expression régulière (traduction erronée de l'anglais *regular expression*)

6. unicode est le répertoire universel de caractères le plus complet existant

Voici le code de cette partie :

```

{
open Parser      (* importation des token *)
}

rule token = parse
  [ ' ' '\t' '\n' ] { token lexbuf } (* skip blanks *)
  | eof              { EOF }

  | "("              { LPAREN }
  | ")"              { RPAREN }
  | "["              { START }
  | "|"              { MIDDLE }
  | "]"              { END }
  | [ 'A'-'Z' 'a'-'z' ]+ { PROP }

  | "eqP"            { FUNC }

```

Analyse syntaxique

La seconde partie du traducteur est l'analyseur syntaxique. C'est lui qui interprète le « sens » de l'expression à traduire.

Son principe de fonctionnement est simple mais très puissant : il trouve des *patterns* (structure, motif) et les remplace, de manière imbriquée et récursive. Voici comment ils sont définis :

```

start :
  expr EOF { "$1." }

```

Le point d'entrée est le *pattern* **start**, celui-ci correspond au *pattern* **expr** suivi de la fin du message (**EOF**). Il est traduit simplement en rajoutant un point après l'expression, qui elle va être interprétée de la façon suivante :

```

expr :
  PROP { "$1" }
  | PROP '(' list ')' # f(abc)->f a b c
    { "$1□$3" }
  | START init MIDDLE expr END # [a|a]->forall a:Prop, a
    { "forall□$2,□$4" }

```

On remarque ici qu'`expr` correspond à plusieurs motifs. Le dernier est particulièrement intéressant car il contient `expr`, l'analyseur syntaxique utilise donc de la récursivité.

Pour finir l'exemple, la suite de *token* sortie de l'analyseur lexical : `START PROP PROP MIDDLE PROP LPAREN PROP PROP RPAREN END` est traduite de la manière suivante : `forall a b :Prop, a ≡≡ b`.

Cette première version était simple à réaliser, la traduction ne nécessitant pas la détection de fonctions. En effet, la seule fonction présente dans les deux premiers axiome est l'équivalence, noté ϕ .

Cette étape a permis de mettre en place l'architecture du traducteur, par exemple un script de compilation, un fichier avec les fonctions « outils », ainsi qu'un fichier qui stocke une liste de fonctions propre au système (tel l'équivalence) et leurs traductions. Ce dernier permet une certaine souplesse ; en effet, pas besoin de recompiler le compilateur pour lui ajouter de nouvelles règles.

3.2.2 Un 2^e traducteur

La plus grosse difficulté rencontrée pour traduire le dernier axiome, est la manière qu'utilise LESNIEWSKI pour définir des fonctions. Certaines fonctions, comme l'équivalence, font parties du symbolisme authentique et ne pose pas de problème, mais dans l'exemple suivant : `⊢ ab ⊢ a(b)`, la proposition *a* est une fonction alors que *b* non. Pourtant, elles bénéficient de la même déclaration, ce qui n'est pas le cas en sortie : `forall (b :Prop)(a :Prop -> Prop), a b`.

Ce problème n'est pas (à ma connaissance) soluble en une seule analyse du fait que l'on découvre si une proposition est une fonction qu'après l'avoir définie. La solution a donc été de concevoir un deuxième traducteur qui sera appelé avant le principal pour inférer le type de chaque proposition. Nombre de langage de programmation utilise une inférence de type et il existe donc des algorithmes pour résoudre ce type de problème. Le plus courant est le système de typage d'HINDLEY–MILNER⁷ et son implémentation principale, l'algorithme W. Malheureusement, je n'ai pas réussi à le comprendre suffisamment pour l'utiliser dans le compilateur. La solution actuelle est donc

7. aussi connu sous les nom DAMAS–MILNER ou DAMAS–HINDLEY–MILNER

une inférence de type programmé « à la main » sans aucune connaissance profonde du domaine.

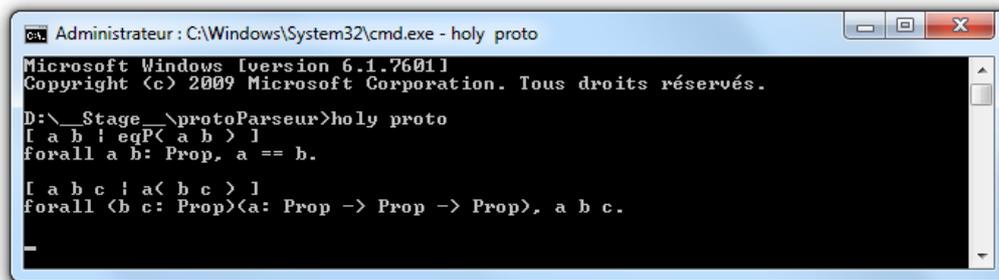
3.2.3 Les tests

Pour confirmer les fonctionnalités du programme, un jeu de test a été mis en place. Ce jeu est relativement simple et ne bénéficie d'aucune automatisation tel une implémentation via un *framework* de tests unitaire. Cette automatisation aurait pris du temps pas nécessairement disponible pour un gain de confort tout relatif.

En effet, une liste de cas tordu dans un fichier texte a été largement suffisant, et elle a par ailleurs permis de détecter une faille importante. Dans la première version de notre l'inférence de type, les fonctions en paramètre de fonctions, elles mêmes en paramètre d'autres fonctions n'étaient pas détectées, alors que que le traducteur est sensé les détecter quelque soit leurs profondeurs. Cela a donné lieu à une refonte de l'algorithme fait maison qui est actuellement complètement fonctionnel de ce point de vue.

3.2.4 « Mise en production » et environnement UNIX

Tel quel, le compilateur est un programme en console interactif, qui attend une entrée et renvoi la traduction.



```
ca. Administrateur: C:\Windows\System32\cmd.exe - holy proto
Microsoft Windows [version 6.1.76011]
Copyright (c) 2009 Microsoft Corporation. Tous droits réservés.

D:\_Stage_\protoParseur>holy proto
[ a b ! eqP< a b > ]
forall a b: Prop, a == b.

[ a b c ! a< b c > ]
forall <b c: Prop><a: Prop -> Prop -> Prop>, a b c.
```

Un problème a été rencontré par lors de l'utilisation du programme sous un environnement UNIX (Linux et OS X). En effet, les retours à la ligne dans les fichiers textes ne sont pas encodés de la même façon⁸ sous UNIX ou Windows. Même après une recompilation, le programme ne fonctionnait pas ; la solution a été de changer les fins de ligne avant d'exécuter un script de compilation spécial.

8. http://fr.wikipedia.org/wiki/Fin_de_ligne

3.3 2^{de} partie, l'interface

La seconde partie du stage correspond à la création de l'interface utilisateur.

3.3.1 L'affichage

Police d'écriture

Pour pouvoir éditer visuellement les expressions de la protothétique, il faut pouvoir afficher les caractères spéciaux qui composent son symbolisme authentique. Cela pose un problème, car n'étant pas définis dans unicode, aucune police actuelle ne les représente. Deux solutions étaient possibles : soit utiliser des images, soit créer une police d'écriture, ou du moins en étendre une déjà existante. La première solution est plus simple mais pose un certain nombre de problèmes. Le principal étant que ces images n'ont pas de représentation textuelle, et cela est une difficulté supplémentaire pour mettre en place un copier-coller, ou plus simplement récupérer le texte pour l'envoyer au traducteur. Cela est faisable au prix de nombreux efforts pour un résultat moyen à l'affichage.

En effet, si le texte est agrandi, les images vont, soit ne pas correspondre à la taille du texte, soit être pixélisées. Une des solutions à ce dernier problème a été de créer des images vectorielles⁹. C'est justement la méthode privilégiée pour définir une police. J'ai donc dessiné¹⁰ tous les caractères nécessaires (voir l'image ci-dessus) à l'aide du logiciel FontForge et je les ai ajoutés à Lucida Console, le choix de la police étant celui de mon tuteur.

0	1	2	3
—	└	┘	┐
⊔	⊕	⊖	⊗
⊘	⊙	⊚	⊛
⊜	⊝	⊞	⊟
⊠	⊡	⊢	⊣
⊤	⊥	⊦	⊧
⊨	⊩	⊪	⊫

richtextbox

L'objet graphique utilisé pour l'édition d'expressions ainsi que pour le résultat de la traduction est un `RichTextBox`. C'est un champ de texte qui utilise le format *Rich Text Format* de Microsoft, celui-ci permet de changer la mise en forme – couleur, taille, gras, police... – d'une partie du texte seulement. Cela permet notamment de changer de couleur une paire de parenthèses lors de la sélection de l'une des deux. Cette aide est vraiment utile pour vérifier

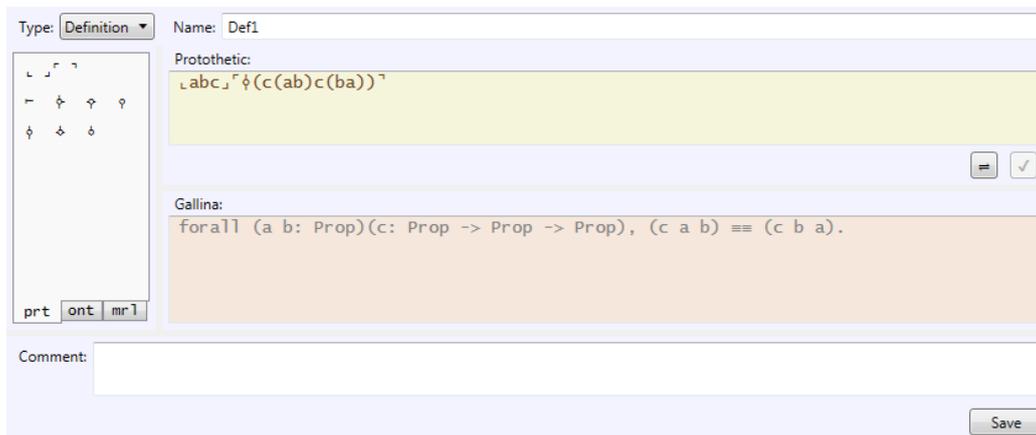
9. image numérique composée d'objets géométriques individuels (droite, arcs, courbes, polygones, etc.) auxquels on peut appliquer différentes transformations

10. cf. annexe A.3

`⊥ pqr ⊃ ⌈ φ (φ (φ (pq) φ (rp)) φ (qr)) ⌋`

rapidement la parité des parenthèses, ce qui est utile avec des expressions mathématiques. L'objet `RichTextBox` étant plus complexe que le champ texte de base, j'ai développé une classe pour encapsuler cette objet et simplifier les accès a ses propriétés pour pouvoir m'en servir de la même façon qu'un champ texte. C'est cette encapsulation qui contient les fonctions nécessaires à la mise en valeur des parenthèses.

3.3.2 Onglets



L'interface comprend une partie saisie et traduction d'une expression (capture ci-dessus). Cette partie a, en premier lieux, été développer comme une fenêtre à part entière, mais le logiciel est plus complet car il comprend une partie stockage et gestion des expressions. Ce contrôle utilisateur devais donc être intégré à une fenêtre plus grande. La première idée fut d'utiliser des MDI¹¹ mais Microsoft ayant progressivement abandonné cette technologie au profit des onglets (comme pour les navigateur web), celle-ci n'est pas disponible avec WPF. Heureusement, un certain nombre de développeurs ont déjà eu ce problème et internet regorge de morceaux de code pour adapter une fenêtre en onglet.

L'interface complète est en annexe (A.4).

11. le MDI (Multiple Document Interface) désigne l'organisation d'une interface graphique où des fenêtres parentes contiennent en leur sein des fenêtres enfants

3.4 Travail restant

Au moment de la rédaction du rapport, le stage n'est pas fini et le projet non plus. Les expressions sont actuellement stocker dans un fichier texte sans réelle structure. La dernière semaine de stage vas me permettre de transformer ce document en un XML structuré et plus juste dans la représentation des objets utilisé pour enregistrer ces traductions.

Pour finir, le projet initial comprenais également un outil pour « sérialiser » le travail existant, cela dans le but de ne pas avoir besoin de retraduire les 2000 ligne d'expression déjà traduite et prouvez avec Coq. Cette partie du projet sera confié à des étudiant du DUT Info d'Annecy pour leur projet tutoré de 2^e année.

4 Bilan

Ces 11 semaines de stage m'ont apportées énormément, tant sur le plan professionnel que personnel.

4.1 Bilan professionnel

Travailler au LISTIC m'a permis de devenir plus autonome et de mieux gérer mon temps de travail. En effet, j'ai principalement travaillé seul lors de cette mission. Cette autonomie m'a appris à mieux organiser mon temps de travail, et le cadre d'un laboratoire de recherche a été un avantage. Les horaires sont relativement souples, et cela permet de mieux gérer des éventuelles contraintes externes.

Mon autonomie sur la mission m'a aussi également obligé à être plus rigoureux, notamment dans mes rapports quotidiens au demandeur du projet. Cet échange nous a permis de suivre mon travail et de préparer plus facilement les réunions hebdomadaires pour définir la suite du stage.

De plus, les contraintes d'un projet de recherche sont plus fluctuantes que pour un projet « classique » ; c'est à la fois un avantage et un inconvénient. L'objectif final du stage n'étant pas totalement défini au début on ne se lasse pas de notre mission qui évolue au fil des semaines, et par conséquent est totalement unique. Mais cette évolution peut être mal vécue. Il est souvent plus rassurant d'avoir un point de vue global sur le projet alors que dans mon cas, j'avais une visibilité seulement supérieure à une semaine.

Ce stage reste une expérience de travail unique et particulièrement enrichissante.

4.2 Bilan personnel

Personnellement, travailler au LISTIC a été très enrichissant, et c'est une expérience que je n'hésiterai pas à refaire. L'ambiance est détendue et particulièrement plaisante.

Cette ambiance calme est très propice au travail individuel et n'est pas un obstacle à la collaboration. Les personnes présentes au LISTIC – les chercheurs, le personnel ou les autres stagiaires – sont très ouvertes au dialogue et sont prêtes à apporter leurs compétences ou leurs expériences.

J'ai réellement apprécié cette atmosphère particulière, et ce stage m'a conforté dans mon idée de poursuivre mes études vers la recherche scientifique. Je pense que travailler au milieu de personnes de différentes cultures et niveaux d'études est un plus que d'autre stage ne m'aurait pas permis.

Conclusion

Mon stage au laboratoire LISTIC m'a permis de découvrir le travail en recherche. C'est une occasion rare qui m'a été proposée, et j'ai su en profiter pleinement durant ces dix semaines de stage.

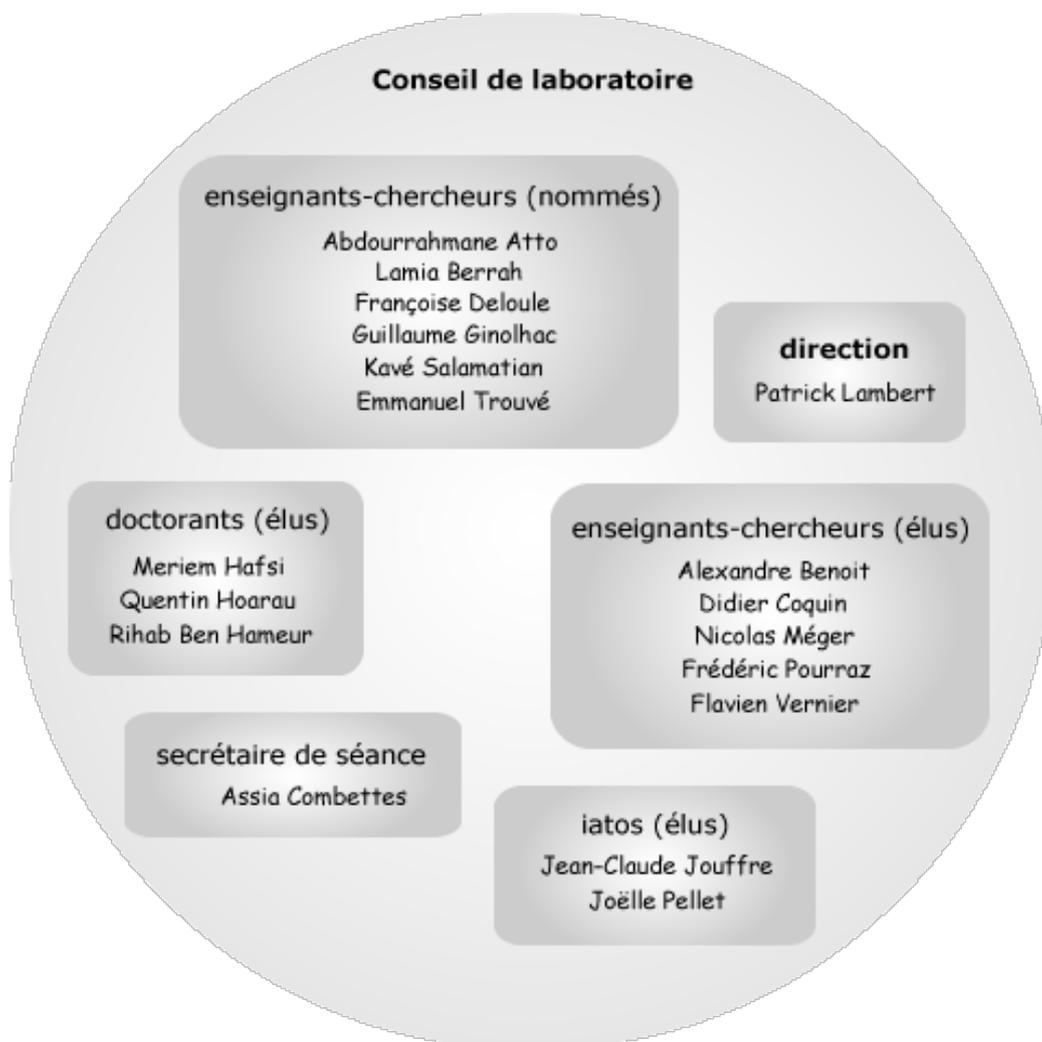
Le travail avec les chercheurs n'est pas toujours facile. Les contraintes changent rapidement et il faut savoir s'adapter. Ce fut l'occasion pour moi de découvrir des technologies peu utilisées et d'évoluer dans un environnement peu commun. Mais aussi de me rendre compte des contraintes inhérentes au développement d'une solution informatique.

Je me considère comme chanceux d'avoir eu la possibilité de faire ce stage en recherche. Finalement, mon travail a permis de développer un compilateur, et même si l'interface n'est pas finie, le cœur de l'application est fonctionnel et sera utilisé « pour faire avancer la science ».

Le monde de la recherche est un monde à part. Le découvrir de l'intérieur et y participer a renforcé l'envie pour moi de poursuivre dans cet univers.

Annexes

A.1 Conseil du laboratoire



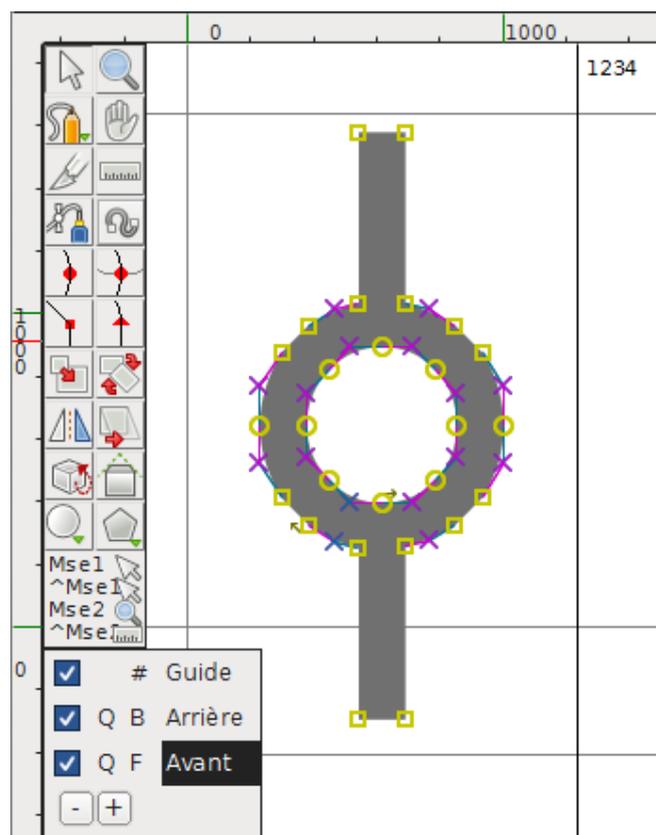
A.2 Les trois premiers axiomes de la protothétique

$$A1 \quad \lrcorner pqr \lrcorner \phi \left(\phi \left(\phi(pr) \phi(qp) \right) \phi(rq) \right)$$

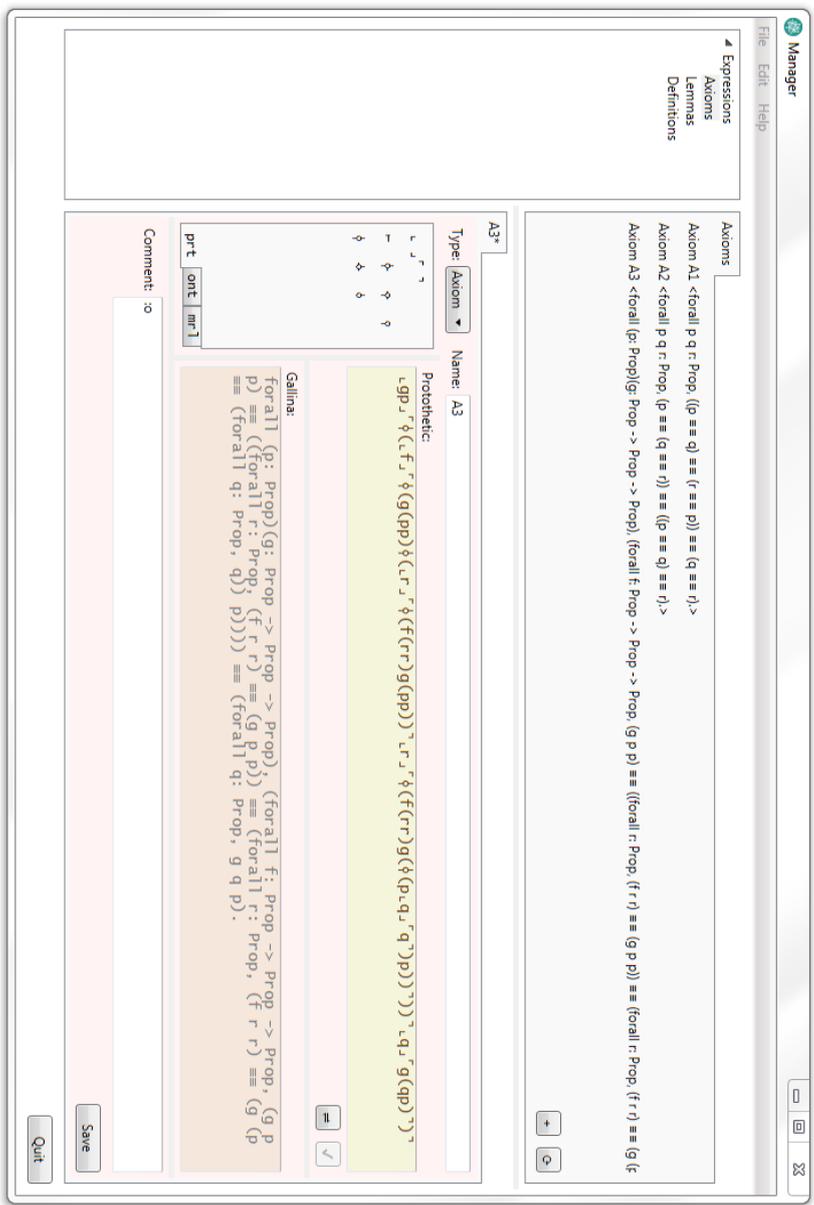
$$A2 \quad \lrcorner pqr \lrcorner \phi \left(\phi(p \phi(qr)) \phi(\phi(pq)r) \right)$$

$$A3 \quad \lrcorner gp \lrcorner \phi \left(\lrcorner f \lrcorner \phi \left(g(pp) \phi \left(\lrcorner r \lrcorner \phi \left(f(rr)g(pp) \right) \lrcorner r \lrcorner \phi \left(f(rr)g(\phi(p \lrcorner q \lrcorner q \lrcorner p)) \right) \right) \right) \lrcorner q \lrcorner \phi(g(pp)) \right)$$

A.3 Édition d'un caractère dans FontForge



A.4 Fenêtre complète du logiciel



Laboratoire LISTIC
Polytech Annecy Chambéry
5, chemin de bellevue
74944 Annecy-Le-Vieux

COLLET Benjamin
IUT d'Annecy le vieux
DUT INFO
Année 2015/2016

Résumé : La traduction d'un système peu connu de logique (la protothétique) en langage moderne, pour permettre d'automatiser ses preuves, demande du temps et pourrait être informatisé. Pour faciliter la tâche des chercheurs, j'ai développé un logiciel de gestion d'expression qui inclus un traducteur.

Mots clefs : Compilateur, Lex/Yacc, OCaml, Coq, Interface graphique, WPF

Abstract: The translation of an exotic logic system (protothetic) in modern language, with the aims of automatizing proof, is very time consuming and a software can help. To help researchers, I developed a expressions manager software with a build-in translator.

Keywords: Parser, Lex/Yacc, OCaml, Coq, Graphical user interface, WPF

DAPOIGNY Richard

DELOULE Françoise