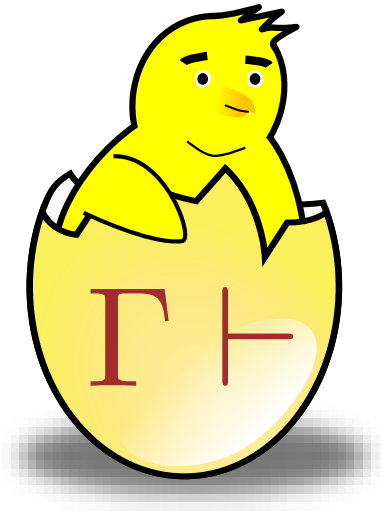


Rapport final du projet Coquille

M1 Informatique Fondamentale - ENS Lyon

4 janvier 2009

Table des matières

Chapitre 1

Le projet Coquille en général

1.1 Introduction

1.1.1 Présentation générale

L'objectif de Coquille (*COQ User-Interactive Library Learning Expert*) est la réalisation d'un environnement de preuve assistée par ordinateur dotant l'utilisateur des outils nécessaires à l'implémentation de résultats proches de ceux démontrés dans les classes préparatoires. Il doit être élaboré à partir de l'assistant de preuves Coq.

L'objectif est en somme d'élargir la portée de Coq aux mathématiques de classes préparatoires, et à des mathématiciens non forcément experts en calcul des constructions et en théorie des types.

1.1.2 Motivation

La preuve assistée par ordinateur ouvre de nouvelles perspectives pour les mathématiques modernes. Le chemin parcouru depuis `Automath`[3] (fin des années 60) permet de parler réellement de preuve interactive : l'utilisateur n'a plus à fournir un λ-terme du bon type mais peut le construire *via* l'utilisation de tactiques et de lemmes définis dans la bibliothèque standard ou par ses soins.

La principale limite des assistants de preuves n'est donc plus la complexité de la théorie sous-jacente[1] mais l'absence de certaines bibliothèques de base (une bonne partie du programme des classes préparatoires n'est pas couverte), la nécessité de démontrer de nombreux sous-buts (relativement) triviaux ainsi que l'austérité des interfaces (ce qui peut rebuter les non-informaticiens).

Partant de ces constatations, le projet Coquille tâchera de développer des outils permettant de démontrer simplement des théorèmes énoncés en classes préparatoires.

1.1.3 Public visé

Nous souhaitons avant tout fournir un outil utilisable dans le contexte des classes préparatoires, mais il serait très fortement envisageable de le diffuser dans le monde de l'industrie et de la recherche (par exemple pour aider à confirmer des conjectures ou même des preuves lourdes en calculs et apporter une surcouche de fiabilité aux chercheurs).

1.1.4 Approches

Le projet Coquille a trois composantes principales et chacune a pour but de combler une partie des limites évoquées précédemment.

[1] Il est tout à fait possible de démontrer de nombreux théorèmes sans comprendre toutes les subtilités du calcul des constructions.

Ergonomie Le logiciel doit être ergonomique : même si le public visé est assez scientifique pour vouloir prouver un théorème de mathématiques, il n'est pas forcément informaticien. Le projet Coquille développera donc une interface intuitive et fonctionnelle pour Coq.

Simplicité La démonstration de lemmes devra être la plus simple possible, nous tâcherons donc d'automatiser au maximum le processus. Les problèmes pouvant être difficiles (indécidables par exemple), il n'est bien entendu pas exclu que l'ordinateur demande de l'aide à l'utilisateur.

Rapidité Nous souhaitons proposer à l'utilisateur un certain nombre de bibliothèques et de tactiques de haut niveau correspondant aux différents chapitres du programme des classes préparatoires. L'existence de ces outils permettra d'implémenter rapidement des preuves complexes en réutilisant des résultats déjà existants.

Pour parvenir à ce résultat, ce projet de recherche s'est organisé autour de trois axes.

Résultats mathématiques majeurs Les groupes de travail *Preuves* et *Tactiques* développent des bibliothèques regroupant les concepts évoqués en classes préparatoires ainsi que les résultats majeurs les concernant.

Démonstrations automatiques Le groupe de travail *Apprentissage* a pour but de permettre la résolution automatique de résultats simples *via* l'adaptation d'un algorithme d'apprentissage classique au contexte d'une démonstration en Coq.

Prise en main facile Le groupe de travail *IDE* développe une interface intuitive et ergonomique permettant une utilisation simple de Coq et de l'intégralité des résultats démontrés au sein du projet Coquille.

1.1.5 Ressources humaines

Ce projet est développé par 22 élèves du M1 d'informatique fondamentale de l'ENS Lyon. Les tâches se sont réparties entre quatre groupes de travail (WP) dont nous allons décrire les développements. La liste des participants et découpage des groupes de travail est en annexe.

Chapitre 2

Le WP preuves

2.1 Logique

2.1.1 Indénombrabilité de $\mathbb{R}$ (`Runcountable`)

Motivations

Le résultat d'indénombrabilité de $\mathbb{R}$ est un résultat fondamental en théorie des ensembles. À ce qu'il semble, il n'a toujours pas été démontré en Coq avec l'axiomatique classique des réels[1]. Quoi qu'il n'ait pas d'intérêt en lui-même au sein d'une bibliothèque d'analyse réelle, ce théorème mérite clairement de figurer parmi les résultats de `Rlogic`.

Énoncé et preuve

Ce théorème se présente sous deux formes, l'une plus forte que l'autre :

`R_uncountable_strong` : $\forall f : \mathbb{N} \to \mathbb{R}, \forall x < y, \{r \mid \forall n \in \mathbb{N}, r \neq f(n) \wedge \ r \in [x, y]\} \neq \emptyset$
`R_uncountable` : $\forall f : \mathbb{N} \to \mathbb{R}, \{r \mid \forall n \in \mathbb{N}, r \neq f(n)\} \neq \emptyset$

La preuve n'utilise pas la version générale de l'argument diagonal de Cantor, beaucoup trop dur à formaliser en Coq, mais utilise la première preuve de ce résultat que Cantor ait fourni. Elle repose sur un argument purement topologique, à savoir la complétude de $\mathbb{R}$.

2.1.2 Implications classiques de l'axiomatique réelle (`Rmarkov`)

Motivations

L'axiomatique réelle de Coq est *classique*, en ce sens où elle permet de prouver des propriétés que la logique intuitionniste (et par là même Coq dans le contexte vide) ne permet pas. On ignore encore exactement quelles sont les limites de l'expressivité classique conférée par cette axiomatique, mais on peut d'ores et déjà la comparer à une axiomatique intuitionniste telle que décrite dans [2].

D'un point de vue logique, ceci nous permet de quantifier ce pouvoir expressif, ce qui est toujours intéressant. D'un point de vue pratique, cela nous permet de tirer des ponts entre les différentes représentations des réels, qu'elles soient constructives ou non.

Une preuve certaine que cette question possède un intérêt est apportée par le fait qu'une partie de notre démonstration a été faite avant nous par des chercheurs il y a quelques mois, et sert de base à un développement très intéressant entre C-Corn et Coq [5].

[1] Cependant, il en existe une preuve dans C-Corn.

Énoncés

L'axiomatique réelle nous permet de prouver les deux formules suivantes :

$$\begin{array}{ll} \texttt{R\_markov} : & \forall P : \mathbb{N} \to \mathbb{B}, \neg(\forall n \in \mathbb{N}, P(n)) \Rightarrow \exists n \in \mathbb{N}, \neg P(n) \quad \text{(MP)} \\ \texttt{R\_sequence\_dec} : & \forall P : \mathbb{N} \to \mathbb{B}, (\forall n \in \mathbb{N}, P(n)) \vee \neg(\forall n \in \mathbb{N}, P(n)) \quad \text{(WLPO)} \end{array}$$

Le premier est le principe de Markov, le second est le principe d'omniscience limité faible. Il est clair que MP n'est que légèrement classique (même si $\nvdash_{CIC}$ MP, il est constructif au sens algorithmique du terme), alors que WLPO est violemment classique (il permet de résoudre le problème de l'arrêt).

Contrairement à l'article évoqué précédemment, nous avons montré que le principe de Markov était entièrement déductible de l'axiomatique réelle.

Ce résultat permettrait d'écrire une bibliothèque d'analyse réelle auto-contenue, qui ne ferait pas appel à des axiomes classiques. En effet, la plupart des propriétés analytiques sont séquentielles, et MP et WLPO permettent de manipuler les suites classiquement[2].

Par ailleurs, nous avons tenté de tirer la quintessentielle moelle classique de l'axiomatique réelle. Hugo Herbelin pensait que l'on pouvait déduire le fait que $\mathbb{R}$ était archimédien des autres axiomes[3]. Il nous a communiqué une preuve intéressante qui montrait que l'on pouvait déduire le tiers exclu faible (WEM) dans `Type` à partir de l'axiomatique des réels :

$$\texttt{Reals} \vdash \forall A : \texttt{Prop}, \{\neg A\} + \{\neg\neg A\}$$

L'axiome archimédien et MP sont de fait équivalents[4]. Nous avons des indices qui nous laissent penser que cet axiome ne peut pas être réduit aux autres. D'abord, parce que MP et WLPO sont complètement orthogonaux en logique intuitionniste. Ensuite parce que WEM a beau impliquer WLPO, WEM n'implique pas non plus MP. Or l'argument utilisé pour prouver WEM à l'aide de l'axiomatique des réels est peu propice à la généralisation pour MP. En effet, de quelque bord qu'on l'attaque, la technique employée ne donne que des résultats par double négation, ce qui est fâcheux pour MP (on veut justement éliminer cette double négation).

Évidemment, nous n'avons pas de modèle pour prouver cela, mais nous conjecturons que l'on ne peut pas se passer de l'axiome archimédien. On pourra trouver dans [1] une série de résultats de hiérarchisation des propriétés semi-classiques qui nous confortent dans cette supposition.

Développements éventuels

L'idée de bibliothèque réelle auto-contenue pourrait être complètement implémentée, et cela en expurgeant toutes les références au module `Classical` de la bibliothèque `Reals`. Cependant, la tâche risque de s'avérer longue pour un résultat à l'intérêt douteux.

2.2 Réels

2.2.1 Suites réelles (`Rsequence`)

Motivations

Les suites sont parmi les outils de base de l'analyse réelle, et le moins qu'on puisse dire, c'est que la bibliothèque standard de Coq n'est ni très étendue dans ce domaine, ni très cohérente[5]. Il y a donc une masse de travail non-négligeable.

Pour ne pas reproduire les défauts de la bibliothèque standard, nous nous sommes entendus pour suivre à la lettre les conventions de nommages recommandées dans la proposition de Hugo Herbelin[4], même si cela s'avère parfois fastidieux. Nous avons également réparti les lemmes en différentes sous-bibliothèques thématiques afin de garantir une consultation aisée et de rendre possible une utilisation modulaire.

[2] Nous n'avons pas encore trouvé de propriété purement analytique sur $\mathbb{R}$ qui ait besoin de plus de MP et WLPO pour être prouvée. Évidemment, si on accepte l'axiome du choix, c'est une autre affaire.

[3] L'axiomatique de Coq suppose explicitement que c'est un corps archimédien.

[4] Nous en avons une preuve.

[5] Comme à peu près tout `stdlib`.

Contenu de la bibliothèque

Elle définit des propriétés de base, et opère quelques renommages sur les lemmes de `stdlib`. Elle se veut très généraliste, sans pour autant assommer l'éventuel utilisateur sous des monceaux de résultats à la `techX`.

Convergence Nous avons renommé la convergence vers un réel afin de respecter un schéma plus naturel. Nous avons aussi défini formellement la divergence vers l'infini (ce qui n'existe pas de base dans la bibliothèque standard). `Rsequence` contient une quantité assez impressionnante de lemmules sur la compatibilité des opérations avec la limite. Ces résultats sont d'ailleurs utilisés par une tactique de décision de convergence.

Relations de Landau Grands classiques des propriétés séquentielles qui souffrent pourtant de cruelles absences de `stdlib`. Nous avons donc défini les relations de Landau petit-o, grand-O et équivalence de suite. Nous en avons montré des propriétés importantes : quasi-ordre, équivalence, compatibilités entre elles, etc. Nous avons en outre montré des compatibilités avec les propriétés à la limite et les opérations usuelles.

Suites usuelles Afin de rester dans un esprit pratique, notre bibliothèque définit des suites appelées à être utilisées un peu partout : suites constantes, polynomiales, exponentielles et factorielles. Certains résultats de comparaison ont été prouvés, et nous avons aussi décrit leur comportement asymptotique.

Autres `Rsequence` contient des propriétés inclassables, néanmoins fort utiles. On peut compter parmi celles-là le fait que la convergence et les relations de Landau soient asymptotiques, c'est-à-dire que tout lemme permettant de prouver une telle propriété peut être généralisé en montrant que les hypothèses ne sont vérifiées qu'à partir d'un certain rang[6]. Nous avons aussi quelques résultats sur les suites partielles.

2.2.2 Séries réelles (`Rseries`)

Contenu de la bibliothèque

Les séries sont un prolongement naturel des suites, et toutes les définitions sont donc calquées sur celles des suites, notamment celles de convergence, divergence, et les relations de Landau. En particulier, les liens entre convergence, divergence et les relations de Landau ont été faits et beaucoup de résultats concernent les séries à termes positifs puisque beaucoup de problèmes s'y ramènent.

Série $\zeta(2)$

La convergence de la série des inverses des carrés des naturels est une propriété connue. La somme de cette série vaut $\frac{\pi^2}{6}$ et ce résultat fait partie des résultats d'analyse répandus et démontrés de diverses manières.

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

Cependant les preuves de ce résultat sont très calculatoires ce qui les rend difficiles à formaliser. La preuve utilisée dans `Rzeta2` utilise la définition de π dans la bibliothèque standard sous forme d'une série, celle d'arctan, et a motivé la démonstration de nombreux résultats intermédiaires sur les séries, comme la séparation d'une somme infinie selon la parité de ses termes. Elle a nécessité l'introduction des sommes de suites relatives $\mathbb{Z} \to \mathbb{R}$, ainsi que les sommes doubles correspondantes, parfois privées de leurs termes diagonaux. Une partie non négligeable de la preuve est consacrée à des majorations fines et à du calcul.

[6] Pour plus de clarté, se référer à `Rseq_asymptotic`.

2.2.3 Séries entières réelles (`Rpser`)

Motivations

Les séries entières sont une partie intégrante du programme des classes préparatoires. Elles sont pratiquement totalement absentes de la bibliothèque standard (une notation `Pser` traduit le fait que la suite des sommes partielles converge mais les rayons de convergence ne sont nullement formalisés). Leur forme très particulière permet de plus de démontrer des théorèmes puissants sans hypothèses trop restrictives.

On pourrait en déduire la dérivabilité et la continuité de certaines fonctions de la bibliothèque standard (exp, cos, sin) qui sont actuellement prouvées « à la main » et qui utilisent des hypothèses inutiles[7].

Définitions

Une partie importante de nos deux premières semaines de travail a été de trouver les définitions des concepts nous assurant une manipulation aisée de ceux-ci (elles ont fait l'objet de nombreuses retouches au fur et à mesure que nous progressions dans les preuves et que nous constations leurs imperfections).

Nous avons abouti à des représentations très satisfaisantes dont, entre autres, les suivantes :

Série entière On choisit de représenter la série entière $\sum_{n\in\mathbb{N}} a_n z^n$ par la suite $(a_n)_{n\in\mathbb{N}}$. Les fonctions `gt_Pser` et `gt_abs_Pser` sont respectivement de la forme $\lambda a_n.\lambda x.\lambda n.a_n x^n$ et $\lambda a_n.\lambda x.\lambda n.\,|a_n x^n|$ et permettent donc de parler aisément des sommes partielles en n'ayant que la suite $(a_n)_{n\in\mathbb{N}}$ en paramètre.

Rayon de convergence Le rayon de convergence $\rho\,(a_n)$ est défini de la manière suivante :

$$\rho\,(a_n) = \sup\{r \mid (a_n r^n)_{n\in\mathbb{N}} \text{ est bornée}\}$$

On utilise, en pratique, une version plus faible : `Cv_radius_weak` a_n $r \triangleq (a_n r^n)_{n\in\mathbb{N}}$ est bornée.

Manipulation des définitions

Quiconque a travaillé sur la bibliothèque des réels peut mesurer à quel point les lemmes très simples permettant de manipuler les définitions sont vitaux. On se donne donc ici quelques lemmes qui permettront de travailler dans des cas particuliers (les démonstrations d'ordre général que nous avons faites par la suite ne nécessitaient pas vraiment la présence de ce genre de choses).

Opérations simples On prouve que `Cv_radius_weak` supporte l'affaiblissement[8] et on en déduit que `Cv_radius_weak` supporte les opérations simples sur les suites (opposé, somme, différence) pour un r bien choisi[9].

Lien entre `Pser` et convergence de suite On prouve qu'il existe un lien entre `Pser` (définition préexistante à notre bibliothèque) et la convergence des sommes partielles.

Théorèmes fondamentaux

Nous avons choisi de commencer par implémenter les théorèmes fondamentaux traitant des séries entières : lemme d'Abel, critère de convergence de d'Alembert, caractérisation du rayon de convergence et dérivabilité de la série sur son disque de convergence. C'est là que se situe la majeure partie du travail sur les séries entières (la dérivabilité a notamment fait appel à des lemmes qu'il a fallu démontrer dans `RFsequence`).

[7] Les preuves de dérivabilité de cos et de sin dépendent par exemple de l'axiome surnuméraire $\sin\left(\frac{\pi}{2}\right) = 1$.

[8] Si $(a_n r^n)_{n\in\mathbb{N}}$ est bornée et que $|r'| \leq |r|$ alors $(a_n (r')^n)_{n\in\mathbb{N}}$ l'est également.

[9] Le min des $|r|$ des suites en présence fonctionne.

Lemme d'Abel Si $(a_n r^n)_{n\in\mathbb{N}}$ est bornée, alors :

1. $$\forall x, |x| < r \Rightarrow \sum_{n\in\mathbb{N}} a_n x^n \text{ admet une limite finie}$$

2. $$\forall r', 0 \leq r' < r \Rightarrow \sum_{n\in\mathbb{N}} a_n x^n \text{ converge normalement sur } D(0,r)$$

Et une sorte de réciproque : Si $\sum_{n\in\mathbb{N}} a_n x^n$ converge alors `Cv_radius_weak` a_n x.

Critère de d'Alembert

$$\text{Si } \frac{a_{n+1}}{a_n} \to \lambda \text{ alors, } \left(\forall r, 0 \leq r < \frac{1}{\lambda} \Rightarrow \texttt{Cv\_radius\_weak}\ a_n\ r\right)$$

Caractérisation du rayon de convergence On utilise cette fois-ci la définition exacte du rayon de convergence (et non la version affaiblie). Si $\sum_{n\in\mathbb{N}} a_n x^n$ converge simplement et que $\sum_{n\in\mathbb{N}} |a_n x^n|$ diverge, alors x est le rayon de convergence de la série entière.

Dérivabilité de la somme On démontre que toute série est dérivable (donc continue) sur son disque de convergence. Ce résultat est obtenu est utilisant le fait que la suite des dérivées des sommes partielles converge uniformément (car normalement) sur ce même disque.

Étant donné que la dérivée d'une série entière est une série entière (on a d'ailleurs explicité la fonction `An_deriv` $(\lambda a_n.(n+1)a_{n+1})$ qui donne la suite associée à la série dérivée), on vient de prouver que toute série entière est C^∞ sur son disque de convergence.

Séries de Taylor

La bibliothèque contient également quelques développements en série de Taylor des fonctions usuelles. Ces résultats se trouvent dans le fichier `RTaylor`.

2.2.4 Intégrales de Riemann réelles (`Rintegral`)

Motivations

La bibliothèque standard contient les bases sur le sujet, mais les lemmes et définitions sont peu maniables et la convention de nommage est loin d'être respectée puisqu'il est impossible d'inférer le nom d'un lemme et inversement de savoir ce que fait un lemme dont le nom est du type `RiemannInt_P42`. Nous nous sommes donc attachés à fournir des lemmes et des définitions de plus haut niveau à partir des résulats existants tout en essayant de respecter les conventions de nommage.

Définitions et notations

Nous ne manipulons ici que des intégrales réelles définies sur des intervalles. `Rint f a b I` signifie que la fonction `f` est intégrable entre `a` et `b` (*i.e.* existence d'une preuve) et que sa valeur est `I`. Cette définition est similaire à celle donnée pour la convergence des suites et des séries : la valeur de l'intégrale est un argument et non pas une valeur donnée par une fonction.

Lien avec les séries

Nous avons fait le lien entre intégrales et séries, ce qui a entre autres permis de donner un équivalent de la série harmonique.

Propriétés de base

La bibliothèque contient les résultats concernant les propriétés algébriques des intégrales (somme, multiplication par un scalaire) ou les manipulations des bornes (échange des bornes, thèorème de Chasles, intégrale sur un sous-intervalle).

Autres résultats

Quelques propriétés d'inégalité (comme Cauchy-Schwarz ou l'inégalité de la moyenne) sur les intégrales sont maintenant contenues dans la bibliothèque, tout comme d'autres résultats forts comme la nullité d'une fonction continue, positive et d'intégrale nulle ou encore le théorème fondamental de l'analyse. Ce dernier était en effet déjà démontré dans la bibliothèque standard, mais ses hypothèses étaient trop fortes (en particulier, la fonction devait être $\mathcal{C}^1$ sur $\mathbb{R}$, bien que l'on ne l'intègre que sur un intervalle. Ceci a permis de donner la valeur de nombreuses intégrales usuelles comme les polynômes ou les fonctions trigonométriques.

2.3 Complexes

Motivations

L'analyse complexe est une importante partie du programme des classes préparatoires. Il est donc absolument nécessaire de définir les complexes proprement et de développer un très grand nombre de lemmes permettant de manipuler les concepts de base. Ce sont ces lemmes qui permettent ensuite de développer des preuves en évitant de revenir trop souvent aux axiomes (ce qui est très lourd).

Définition et notations

Les complexes ont été définis comme une paire de réels. Ainsi, aucun axiome n'a été ajouté : on se base uniquement sur l'axiomatique des réels.

Nous avons défini une fonction de $(R \to R \to C)$ afin de pouvoir créer des complexes. Cette fonction s'est vu associer la notation `+i`, ainsi un complexe peut s'écrire sous la forme `a +i b`. Nous avons aussi permis l'utilisation des numérables à l'aide d'une coercion de $\mathbb{R}$ dans $\mathbb{C}$.
Nous avons tenté d'utiliser des définitions analogues à celles utilisées dans les réels lorsque cela était possible pour faciliter l'utilisation de nos bibliothèques.

Description

Bases : Définitions, projecteurs et injections des entiers et des réels.

Fonctions de base : Puissance entière, norme et lemmes de manipulation (réécriture, majorations).

Analyse complexe : Continuité et dérivabilité.

Polynômes : Lemmes sur des polynômes de degré 2 permettant de résoudre certains problèmes dans les réels.

Trigonométrie : Existence de la représentation polaire et formule d'Euler.

Suites Complexes : Définition et lemmes sur les suites ainsi que les séries et séries entières.

2.3.1 Propriétés de $\mathbb{C}$ et fonctions de base

Motivations

Il paraît essentiel si l'on veut définir les complexes de façon à ce qu'ils soient utilisables d'écrire des petits lemmes sur la compatibilité de l'addition, multiplication, division, etc. Dans la même optique d'utilisabilité de la bibliothèque complexe, nous avons créé des tactiques permettant d'utiliser les choses connues dans les réels.

Description

Dans `Cbase`, on démontre que les opérations d'addition et de multiplication que l'on a définies font de $\mathbb{C}$ un corps. Cela nous donne un début de bibliothèque de lemmes utilisables pour prouver des théorèmes dans $\mathbb{C}$.

Nous avons des tactiques permettant de passer dans les réels :
- `CusingR` détruit tous les nombres complexes de l'environnement et fait `auto` sur les réels ainsi créés ;
- `CusingR_f` fait la même chose que `CusingR` mais fait `field` au lieu de `auto` ;
- `CusingR_simpl` est utilisable sur une égalité. On passe dans les réels à travers `Cre` et `Cim` respectivement fonctions d'obtention de la partie réelle et complexe.

2.3.2 Propriétés de $\mathbb{C}$ (`Cmet`, `Ctacfield`)

Motivations

La bibliothèque standard propose des résultats généraux sur certaines structures. Afin d'utiliser au mieux ces résultats, on doit évidemment prouver que $\mathbb{C}$ remplit les conditions nécessaires.

Description

- `Cmet` On démontre que $\mathbb{C}$ est un espace métrique.
- `Ctacfield` On démontre que $\mathbb{C}$ est un anneau, puis un corps. Cela nous permet d'obtenir les tactiques `ring_simplify`, `ring`, `field_simplify` et `field` qui sont très utiles pour résoudre des égalités (ces tactiques sont compatibles avec `Cpow` qui est la puissance entière).

2.3.3 Des fonctions et définitions de base

Motivations

Une bibliothèque sur les complexes ne peut pas se passer des fonctions de base (que ce soit la puissance, la norme, etc.) et des lemmes qui permettent leur manipulation (compatibilité avec les opérateurs de base, inégalités, etc.).

Description

- `Cpow` est l'équivalent de `pow` pour les nombres complexes.
- `Cnorm` est la fonction renvoyant la norme d'un nombre complexe (on a notamment prouvé l'inégalité triangulaire).
- `sum_f_C0` est la somme finie.

2.3.4 Suites complexes (`Csequence`)

Motivations

Il est nécessaire de disposer de résultats de base sur les suites à valeurs complexes pour pouvoir construire une bibliothèque traitant des séries entières dans $\mathbb{C}$.

Description

Cette bibliothèque est la petite sœur de la bibliothèque sur les suites à valeurs réelles. Elle ne fait que transposer les théorèmes concernant la convergence des suites et la compatibilité de ces convergences avec les opérateurs de base (opposé, inverse, somme, différence, multiplication).

2.3.5 Séries entières complexes (`Cpser`)

Motivations

Les séries entières sont la base de la trigonométrie telle qu'elle est faite en Coq. Le programme de classes préparatoires se base aussi largement sur ce genre de série. Il apparaît donc essentiel de définir les séries entières en Coq.

`Cpser` est une généralisation de `Rpser` et doit être conçu de manière plus modulaire : le but n'est plus seulement de faire tomber des résultats théoriques majeurs mais de fournir également des lemmes intermédiaires réutilisables lorsqu'on voudra s'intéresser à des cas particuliers.

Description

Cette bibliothèque commence par établir des liens entre les séries entières réelles et les séries entières complexes afin de simplifier les preuves de convergence. On utilsera également un résultat majeur de `Csequence` qui prouve que si $(z_n)_{n\in\mathbb{N}}$ vérifie le critère de Cauchy, alors $(Im(z_n))_{n\in\mathbb{N}}$ et $(Re(z_n))_{n\in\mathbb{N}}$ le vérifient également (et réciproquement).

Elle donne ensuite des preuves modulaires des lemmes d'Abel et d'Alembert : la démonstration est découpée en lemmes intermédiaires très généraux qui sont aisément réutilisables.

Application

Cette bibliothèque peut être utilisée pour définir des fonctions telles que l'exponentielle complexe et les fonctions trigonométriques qui en découlent. Cette approche permet de démontrer automatiquement la dérivabilité des fonctions ainsi définies.

2.3.6 Résultats intéressants

Développement en sommes finies de $(a+b)^n$

Ce résultat semble déjà avoir été démontré dans les réels sous le nom de `binomial`. Il n'est pas possible de le réutiliser directement pour prouver le résultat dans les complexes à cause de la notation `+i` qui fait apparaître de nouvelles difficultés.

Existence de racines d'un trinôme

Dans `Croot_n`, nous nous sommes intéressés aux polynômes. Nous avons entre autres prouvé l'existence d'une racine carrée pour tout nombre complexe. On a aussi montré que tout polynôme complexe admet 2 racines.

On a déduit l'existence ou non de racines pour un polynôme de degré 2 dans les réels ainsi que la positivité lorsque $\Delta < 0$.

Représentation polaire

Nous avons défini les complexes comme des paires de réels. Nous ne savions pas *a priori* qu'il existait une représentation polaire pour ces complexes. Nous l'avons donc montré. Ce résultat fait intervenir l'arctangente (cf. `Reals`).

Formule d'Euler

Nous avons défini l'exponentielle complexe comme une somme infinie (la même définition que l'exponentielle réelle). Nous avons donc montré la formule d'Euler : $e^{i\theta} = \cos\theta + i\sin\theta$.

Produit de Cauchy complexe

Existence de la racine n-ième d'un nombre complexe

Nos résultats précédents nous ont permis de montrer que pour tout nombre complexe z et tout entier n supérieur à 0, il existe un nombre complexe z_1 tel que $z_1^n = z$.

2.4 Topologie

2.4.1 Objets topologiques (`Topology`)

Motivations

L'approche standard de définition des objets mathématiques dans Coq est basée sur la définition constructive d'éléments moins généraux vers d'autres plus généraux. Les principes constructifs justifient cette approche. Cependant certains objets ne peuvent être construits ; par exemple dans `Reals`, le fait que $\mathbb{R}$ est un corps n'est pas une propriété mais un axiome.

Ainsi, pour faciliter la définition d'objets plus généraux, on peut tenter de définir les objets les plus généraux dans un premier temps. Un besoin suggéré par la manipulation des suites, séries ou même des complexes est une homogénéisation des tactiques pour simplifier les expressions qui supportent des propriétés de groupe, d'anneau ou encore d'espace vectoriel. On cherche à généraliser la notion d'espace.

Description

Dans `Topology` sont définis les espaces topologiques. Il est nécessaire d'introduire la notion d'union d'une famille d'ensembles indexée par un ensemble quelconque (dénombrable ou non). On y montre que les topologies discrète $(V, \mathcal{P}(V))$ et triviale $(V, \{\emptyset, V\})$ sont effectivement des topologies, et on définit la notion de séparation (T_2, de Hausdorff).

Dans `Metrics`, on définit les espaces métriques. Notamment, il y est prouvé qu'un espace métrique définit une topologie (dont les ouverts sont les unions de boules ouvertes) et que l'espace topologique ainsi construit est séparé.

La notion de continuité est introduite dans `Continuity`, dans la version topologique et la version métrique. On y montre que, sur un espace métrique, la topologie induite par la distance définit la même continuité que la continuité métrique.

Les espaces vectoriels sont introduits dans `Vectors`, ainsi que la notion de base d'un espace. La définition du produit scalaire est dans `Inner_product`.

Une importante attention est portée à la réutilisation des objets car il s'agit typiquement d'objets sujets à l'héritage. Par exemple, un espace hermitien est un espace vectoriel sur $\mathbb{C}$, de dimension finie et muni d'un produit scalaire. L'utilisation des *typeclasses* semble incontournable.

Développements futurs

Comme `Vectors` pourra donner naissance à des tactiques de manipulation spécifiques aux espaces vectoriels, l'héritage (l'ajout d'un produit scalaire par exemple) pourra augmenter le nombre de tactiques et de théorèmes disponibles. Le but inhérent est d'écrire et de prouver des théorèmes utilisant des objets les plus généraux possibles pour étendre leur champ d'application.

Prévisions (l'ordre est indicatif de la priorité) :
- suites de Cauchy (cas métrique) ;
- complétion ;
- compacité ;
- dimension finie, application ;
- cas de $\mathbb{R}$;
- σ-algèbres, mesure ;
- espace vectoriel topologique ;
- axiome du choix, équivalents, conséquences.

2.5 Arithmétique

2.5.1 Motivations

L'arithmétique est une branche difficile des mathématiques dont la plupart des preuves reposent soit sur des résultats difficiles d'algèbre soit sur des preuves élémentaires mais dont de nombreux éléments de la preuve sont « laissés au lecteur » ou peu formels. De plus, de nombreux éléments considérés comme élémentaires en théorie des nombres (coefficients binomiaux, manipulations de sommes finies, résultats sur la primalité, etc.) demandent un grand effort pour être formalisés et utilisés.

Le but de ce développement est de formaliser quelques uns de ces concepts et de les utiliser afin de prouver un résultat non trivial de théorie des nombres, à savoir le petit théorème de Fermat.

2.5.2 Description

Tous les théorèmes traitent des entiers naturels (`nat`). Bien que la plupart puissent être étendus au cas des entiers relatifs (`Z`), ceux-ci ne sont pas nécessaires pour obtenir ces théorèmes. De plus, la plupart des preuves sont basées sur un raisonnement par induction qui est naturel dans `nat` et qui l'est moins dans `Z`.

Les développements sur l'arithmétique ont été pensés de façon modulaire, ceux-ci regroupent les thématiques suivantes :

- induction : généralisation de la récurrence sur `nat` à la récurrence à n-pas ;
- fonction puissance : définitions de la mise d'un entier à la puissance d'un autre ;
- divisibilité : définition de la divisibilité, de la primalité, du plus grand diviseur commun, du plus petit (resp. grand) diviseur premier d'un nombre, décidabilité de la divisibilité...
- sommes et produits finis : définitions et propriétés, notamment sur le fait de « couper une somme en deux » ;
- coefficients binomiaux : définitions à partir de la relation de Pascal, liens avec l'expression sous forme de factorielles, divisibilité...
- formule du binôme de Newton : preuve du théorème de Newton dans le cas où toutes les variables sont des entiers naturels ;
- petit théorème de Fermat : preuve élémentaire du théorème basé sur la formule du binôme de Newton.

2.5.3 Développements futurs

Il faudrait étendre ces résultats au cas des entiers relatifs. Une autre direction serait de faire une preuve en utilisant des résultats sur les groupes ou encore de généraliser le binôme de Newton à un anneau commutatif (ce serait alors un résultat d'algèbre).

Chapitre 3

Le WP IDE

3.1 Contexte initial

3.1.1 État de l'art : CoqIDE

Pas vraiment user-friendly

CoqIDE possède de nombreux défauts mineurs qui n'apparaissent pas au premier coup d'œil, mais après une utilisation fréquente de ce logiciel. On donne ici une liste de tels détails :

- l'impossibilité de fermer une fenêtre active : il faut sélectionner un autre onglet et effectuer une validation de code ;
- l'impossibilité d'ouvrir un fichier temporaire : il faut obligatoirement sauvegarder le nouveau fichier ;
- une gestion calamiteuse de la coloration des fonctions de recherche et de remplacement : la coloration qui persiste ;
- l'impossibilité de copier-coller depuis un autre cadre que celui d'édition ;
- l'absence de coloration des délimiteurs ouvrants et fermants correspondants (parenthèses, etc.) ;
- le cruel manque d'outils complémentaires que tout IDE devrait proposer : console, bibliothèque, etc. ;
- divers bugs de l'éditeur (liés au copier-coller) ;
- des bugs d'affichage incompréhensibles, sans doutes liés au *binding* OCaml-GTK ;
- le manque de personnalisation de l'interface[1].

On peut en outre constater d'autres défauts de conception plus importants, tels qu'une gestion des *threads* catastrophique[2] et, ce qui en découle, l'impossibilité de valider plusieurs documents à la fois.

Très opaque

En effet, le code de CoqIDE n'est pas très accessible, sa structure pas évidente, et il aurait probablement fallu plus de temps à le comprendre qu'à le développer. Plus exactement :

- il n'y a aucune information sur le code, à part une maigre trame de structure dans la documentation du code source, mais cela reste très léger ;
- (peu modulaire) il n'y a aucune séparation entre CoqIDE et le noyau de Coq, et CoqIDE n'a pas de structure propre indépendante de Coq (cf. les détails de la discussion avec Vincent Gross).

3.1.2 Les besoins du projet Coquille

Mise à part la nécessité d'un IDE pour Coq, qui reprenne donc les principales fonctionnalités de CoqIDE à savoir l'édition et coloration, l'envoi d'instructions à Coqtop étape par étape et la récupération des résultats ou messages d'erreurs, nous sommes restés à l'écoute des attentes des autres groupes de travail (WP) :

[1] Ah, cette couleur, cette couleur !

[2] Coqtop et CoqIDE sont si intimement liés que tout plantage de l'un induit un plantage de l'autre...

WP Apprentissage

Les besoins principaux concernaient l'apprentissage.

- La possibilité pour l'utilisateur de donner une preuve « type » ou « classique » au programme d'apprentissage, qui se chargera de parser le code et d'apprendre la preuve. En somme, cela revient à un bouton « Apprendre cette preuve » qui pourrait s'appliquer à une sélection de code, ou à un théorème en particulier.
- Offrir à l'utilisateur une proposition de preuve ou de tactique, par un bouton « Propositions » par exemple, qui s'en suivrait par une liste de tactiques disponibles et adéquates.
- On pourrait envisager des retours à l'apprentissage pour dire que la proposition a été fructueuse ou que, au contraire, elle est absurde.

WP Preuves / Tactiques

Ce groupe a fourni des librairies et des tactiques sous forme de fichiers en langage Coq directement utilisables par des commandes Coq, donc il n'y avait *a priori* aucune attente de la part de ce WP.

3.1.3 Que faire ?...

Tenter de modifier CoqIDE ?

Ce n'était clairement pas envisageable. En effet, non seulement le code de CoqIDE est complètement opaque (il aurait fallu en comprendre la structure, les subtilités, gérer le dialogue avec Coq sans modularité) mais en plus cela ne nous semblait pas très formateur de reprendre son code. On aurait pu certes reprendre CoqIDE et le structurer d'avantage, mais cela aurait été monstrueusement long.

Créer des plugins pour les éditeurs classiques ?

Mis à part CoqIDE, il n'y a pas d'éditeur dédié à Coq. En admettant que l'on eût fait un *plugin* pour un éditeur classique (NetBeans, Code::Blocks), il aurait été dur de dialoguer avec Coqtop, et un *plugin* ne permet pas de faire un interpréteur. Or, il nous semble vital d'avoir un interpréteur à disposition lorsqu'on code en Coq.

Coder un IDE *from scratch* ?

C'est l'alternative que nous avons choisie. Tout d'abord, il aura été plus facile de bien structurer le code et de séparer le dialogue avec Coq. Cela aura permis également d'intégrer facilement la communication avec les fonctionnalités proposées par l'apprentissage. De même gérer des détails comme la mise en page, la coloration, le *folding* de preuves, nous paraît plus facile si on reprend l'IDE à zéro.

Enfin, dans le cadre d'un tel projet, il est plus formateur de coder un éditeur à partir de rien et de se poser des questions sur la façon de le coder ou de le structurer dans un but précis plutôt que de perdre du temps à modifier un éditeur que nous ne comprenons pas.

3.2 Quelques choix à faire

3.2.1 Le Langage utilisé

Après réflexion, il a été décidé d'utiliser le C++.

Durant la première semaine du Projet, deux parties du WP IDE se sont affrontées en proposant un mini-éditeur codé soit en Java avec Swing soit en C++ avec Qt. Au cahier des charges, ils devaient proposer :

- l'édition de texte avec les menus basiques (copier, coller, etc) ;
- la coloration syntaxique (désactivable) des parenthèses et de certains mots-clés ;
- un bouton « insertion » qui insère du texte après le curseur ;
- un bouton pour afficher l'intégralité du texte dans une nouvelle fenêtre.

Après comparaison sur des critères prédéfinis, il en est ressorti que les deux langages sont très similaires tant en rapidité de codage (ils ont une API pour dessiner les fenêtres chacun par exemple), en structure, et sont assez abordables. Les deux mini-IDE étaient équivalents à la différence que celui en C++ était plus rapide et montrait plus de flexibilité dans la coloration syntaxique. Le mini-IDE en java utilisait des bibliothèques annexes (JLex) pour la coloration syntaxique.

Nous avons donc choisi (à peu de différences) le C++. S'ajoutait le fait que parmi nous plus de personnes étaient compétents en C++ et novices en Java.

3.2.2 Les Bibliothèques utilisées

Qt

Qt nous a séduits pour la gestion des fenêtres. En effet, Qt Creator facilite amplement la création de fenêtres, le code Qt est très clair, propre et structuré ; le *slots* sont gérés proprement en comparaison avec GTK par exemple. D'autre part, la documentation est vraiment très abordable de sorte que l'on a appris très rapidement sur le tas à se servir de Qt.

QCodeEdit

Nous avons passé beaucoup de temps à chercher une bibliothèque d'édition de texte fournissant un *plugin* de *code folding*. Il s'est avéré qu'il en existe peu, et QCodeEdit est l'une d'entre elles. Nous nous sommes aperçus ensuite que cette bibliothèque, très puissante, permet de faire bien d'autres choses, ce qui nous a beaucoup aidé.

QTermWidget

Pour intégrer un terminal à l'IDE, ce fut un peu le même combat que pour trouver QCodeEdit. Nous voulions un terminal qui puisse s'inclure dans un QWidget, et qui soit utilisable exactement comme un terminal UNIX.

3.2.3 Le dialogue avec Coq

Le choix s'est tout simplement restreint à Coqtop, car nous n'en avons pas trouvé d'autre.

3.3 Les problèmes rencontrés

3.3.1 Au niveau du langage

Qt et le C++ n'ont pas posé de problèmes majeurs. Nous étions déjà habitués à coder en C++, et les documentations de Qt et Qt Designer sont vraiment très abordables.

3.3.2 Au niveau des bibliothèques

Nous avons eu quelques petits problèmes de codage en utilisant la librairie QCodeEdit.

Premièrement, c'est une bibliothèque plutôt destinée à être installée par l'utilisateur indépendamment de toute compilation. Nous voulions en inclure directement le code source dans notre programme, il nous a donc fallu batailler ferme.

Ensuite, cette bibliothèque est un projet entier à elle seule, et il a été long d'apprendre à la maîtriser car le code, quoique très bien documenté, reste du code écrit par quelqu'un d'autre, ce que tout le monde connaît comme très dur à relire.

Un grand merci à Hugues Luc Bruant (ENSIMAG), responsable de cette bibliothèque, dont les réponses à nos mails étaient toujours rapides, claires et utiles, ce qui nous a sans nul doute sauvé de nombreuses mauvaises situations.

3.3.3 Au niveau du dialogue avec Coq

Voici les principaux problèmes, ceux qui nous ont pris le plus de temps.

La discrimination des erreurs

Une réponse de Coqtop, qu'elle soit bonne ou mauvaise, est donnée sur `stdout`. Seul le prompt passe par `stderr`. Et, puisque Coqtop ne renvoie aucun autre signal, impossible *a priori* de savoir si une réponse est une erreur ou pas.

Les réponses vides

Certaines commandes de Galina ne renvoient absolument rien quand elles fonctionnent, et une erreur quand elles échouent.

- Si on attend une réponse pour pouvoir la discriminer en erreur ou non, on peut attendre très longtemps.
- Si on choisit un temps maximal d'attente, rien ne nous dit qu'une commande ne va pas prendre exactement 5 secondes de plus que ce laps de temps.

L'annulation

Un véritable casse-tête...

Premièrement, la commande d'annulation d'une commande de base n'est pas la même selon que l'on est au sein d'une preuve ou non.

Ensuite, certaines commandes sont purement inutiles et ignorées par Coqtop. C'est le cas de la commande `Proof.`, qu'il ne faut surtout pas tenter d'annuler sous peine d'annuler la commande précédente.

Enfin, dans certains cas, il est tout simplement impossible actuellement d'annuler une seule opération. D'après ces messieurs de l'INRIA eux-mêmes, le seul moyen d'annuler la commande `Qed` validant un théorème nommé `truc`, par exemple, est d'exécuter `Reset truc`, puis de renvoyer une à une les commandes situées entre la définition de `truc` et le fameux `Qed`. Si jamais la preuve est longue, ou comporte des calculs longs pour l'ordinateur, c'est complètement insensé.

Pire encore, dans le cas de preuves imbriquées, `Reset truc` annule toutes les preuves jusqu'à revenir en mode Galina.

Une solution envisagée était d'utiliser les commandes `Write State` et `Restore State` à chaque étape pour pouvoir revenir en arrière plus facilement. Seulement, Coqtop est tellement surprenant que :

- `Write State` ne fait pas que sauvegarder l'état, il réinitialise Coqtop ;
- `Restore State` ne fonctionne pas du tout !

Nous avons donc contacté Vincent Gross (INRIA), et de cette discussion il ressort que : « *Il n'y a à l'heure actuelle aucune séparation entre Coq et ses interfaces de communication (Coqtop, CoqIDE)* », et que si on a du mal à travailler de cette manière, c'est tout simplement parce que « *Il n'existe aucune API* ». Aucun moyen, donc, de faire notre propre interface sans faire des choses (très) sales.

3.4 Le résultat actuel

3.4.1 Ce que Coquille fait et que CoqIDE ne fait pas !

Au niveau du code

- Marquer les numéros de ligne (cf. Fig 3.1).
- Le *code folding* : replier des lignes de code en une seule pour améliorer la lisibilité (cf. Fig 3.2).
- Des raccourcis clavier plus « classiques », et personnalisables.
- La possibilité de faire des *Redo* après des *Undo* (cf. Fig 3.3).

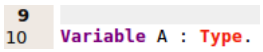


FIG. 3.1 – La numérotation des lignes

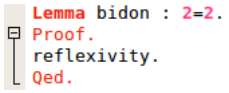


FIG. 3.2 – Le *code folding*

FIG. 3.3 – Le *Redo*

Au niveau du langage

- La gestion de `Ltac Debug` (cf. Fig 3.4). C'est une interface que nous avons mise au point et qui permet de voir en détail l'exécution d'une tactique en Coq, avec une interface de parcours des résultats.
- Gérer plusieurs instances de Coqtop, une par onglet ouvert, sans annuler tout à chaque changement d'onglet.
- Un affichage des résultats en version classique ou *LaTeX-like* (cf. Fig 3.5).
- L'action *Send* / *Unsend* d'envoi d'une commande est considérée comme une action comme les autres, donc *Undo* / *Redo* peut agir dessus.

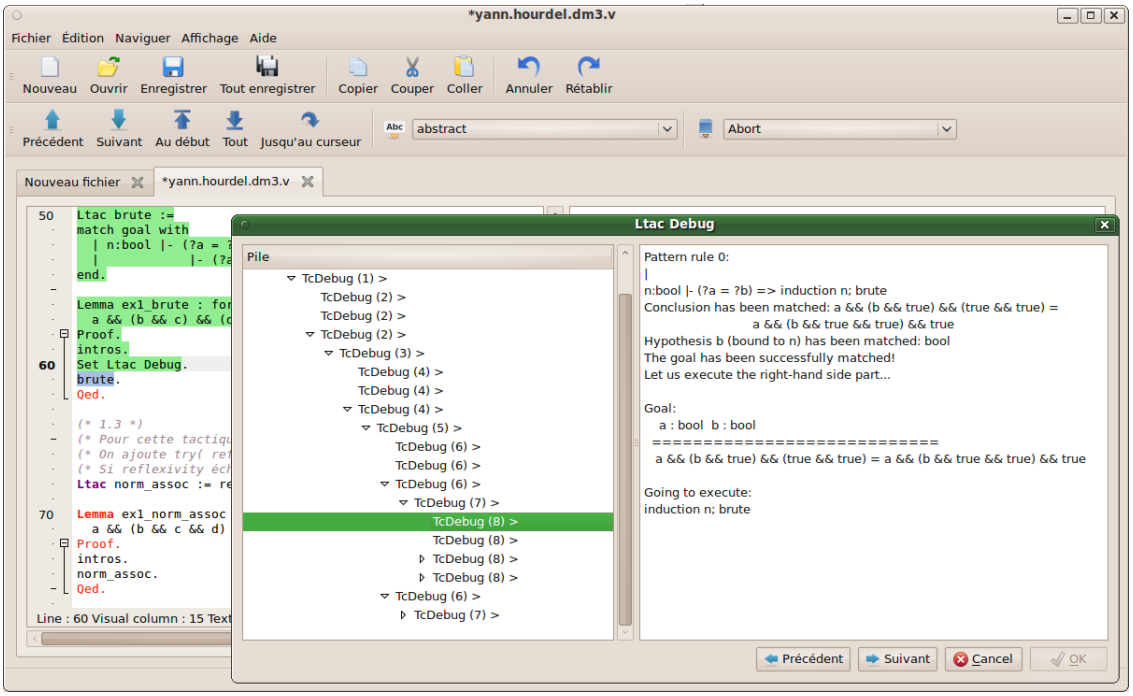


FIG. 3.4 – Le mode `Ltac Debug`

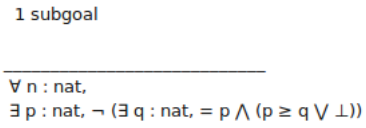


FIG. 3.5 – L'affichage *LaTeX-like*

3.4.2 Ce que CoqIDE fait et que COQUILLE ne fait pas (encore)

Au niveau du code

- Lister les actions disponibles par clic droit sur une hypothèse ou un but.

Au niveau du langage

- Le Proof Wizard.
- La gestion de la compilation et des exportations : CoqIDE permet de compiler / exporter un fichier sans passer par ligne de commande.

3.4.3 Ce que ni l'un ni l'autre ne font bien

Au niveau du code

- L'auto-complétion.

Au niveau du langage

- La gestion de l'aide.
- La gestion des `Write State` / `Restore State`. En théorie, la commande `Write State` est censée sauvegarder l'état de l'interpréteur (variables, commandes envoyées, etc), tandis que `Restore State` est censé restaurer cet état. En pratique, `Restore State` ne marche pas dans Coqtop.

3.4.4 Aperçu en couleur

Sur la figure 3.6, on peut voir la fenêtre principale sans panneau optionnel. La partie gauche est un éditeur de document, par lequel on dialogue avec Coq. Celle de droite contient 2 récepteurs de texte, qui affichent les réponses de Coq.

Sur la figure 3.7, on peut voir la fenêtre principale avec des panneaux optionnels. Le panneau de gauche contient un explorateur de fichiers et un navigateur de documents ouverts. Celui de droite est le panneau de documentation. Ces panneaux sont bien entendu déplacables à d'autres emplacements autour de la fenêtre, et peuvent également être extraits, comme sur la figure 3.8.

Sur la figure 3.9, on peut voir la fenêtre de modification des préférences. L'onglet « Page » contient les préférences d'affichage des éléments (police, taille...). L'onglet « Raccourcis » permet de modifier les raccourcis des actions les plus utilisées. L'onglet « Préférences » permet de choisir quelques options avancées d'affichage (défilement du curseur...).

3.5 Les objectifs

La majorité des objectifs que le WP IDE s'étaient fixés à l'origine ont été largement atteints, puisqu'on dispose d'un IDE fonctionnel si on s'interdit la gestion de l'environnement (`Write State` & cie), qui reprend CoqIDE avec plus de fonctionnalités et qui est plus pratique à utiliser dans l'optique du projet.

Il serait envisageable de continuer le projet si l'INRIA finit de développer une interface de dialogue avec Coq utilisable. En attendant, on ne peut que corriger les bugs.

Au niveau de la distribution, nous avons déjà mis Coquille sous forme de paquets (`.deb` & cie). Il nous reste donc principalement à distribuer et diffuser Coquille dans la communauté Coq et à y inclure un petit manuel d'utilisation. Un manuel programmeur basique est déjà mis à disposition.

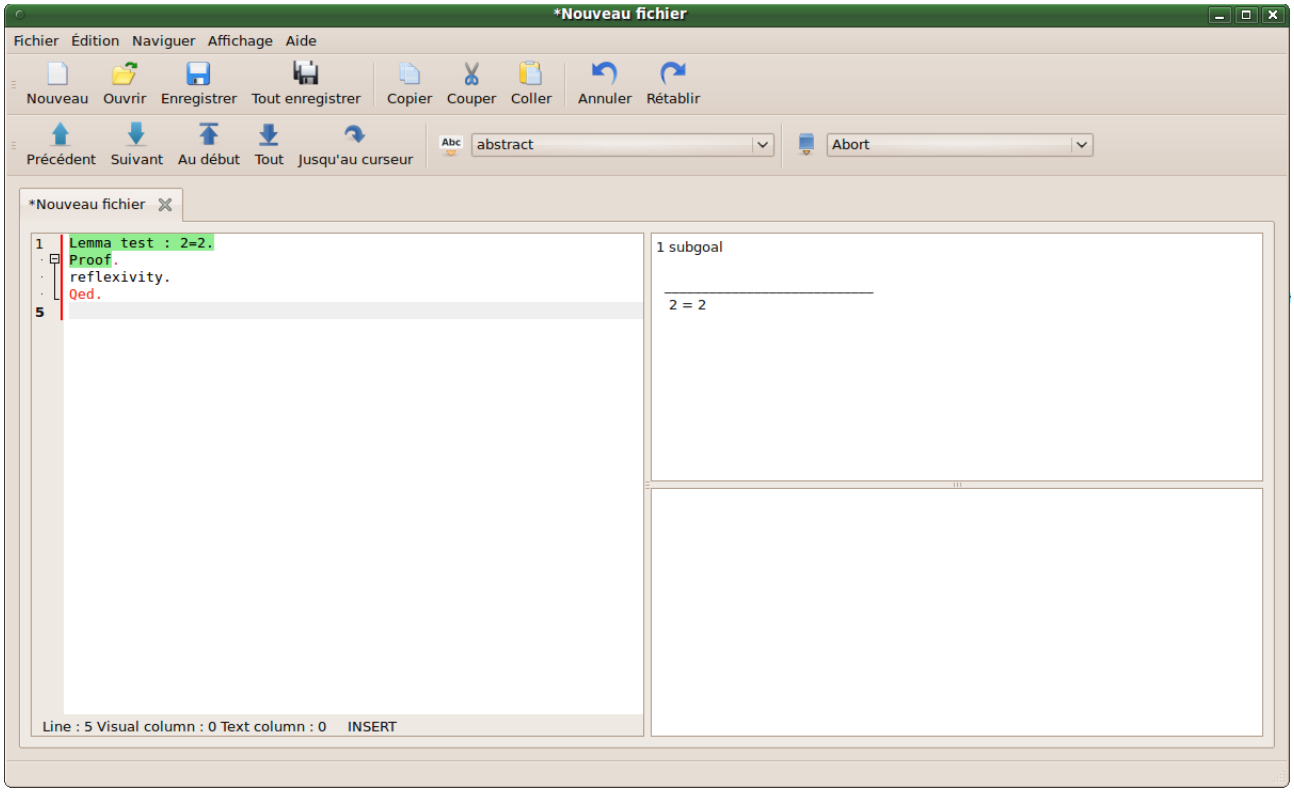


FIG. 3.6 – La fenêtre principale

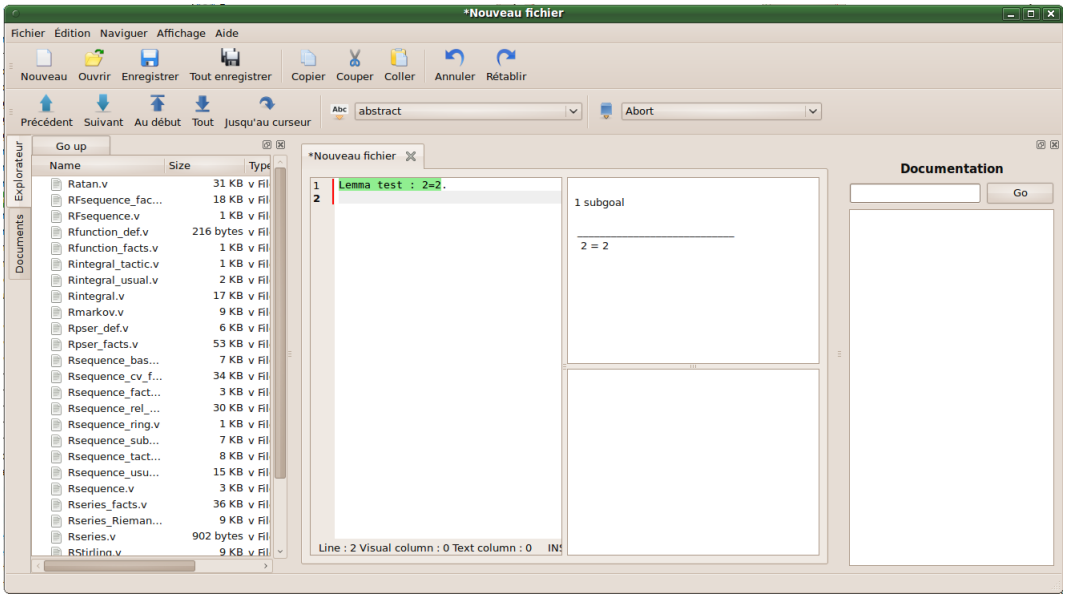


FIG. 3.7 – Les panneaux optionnels

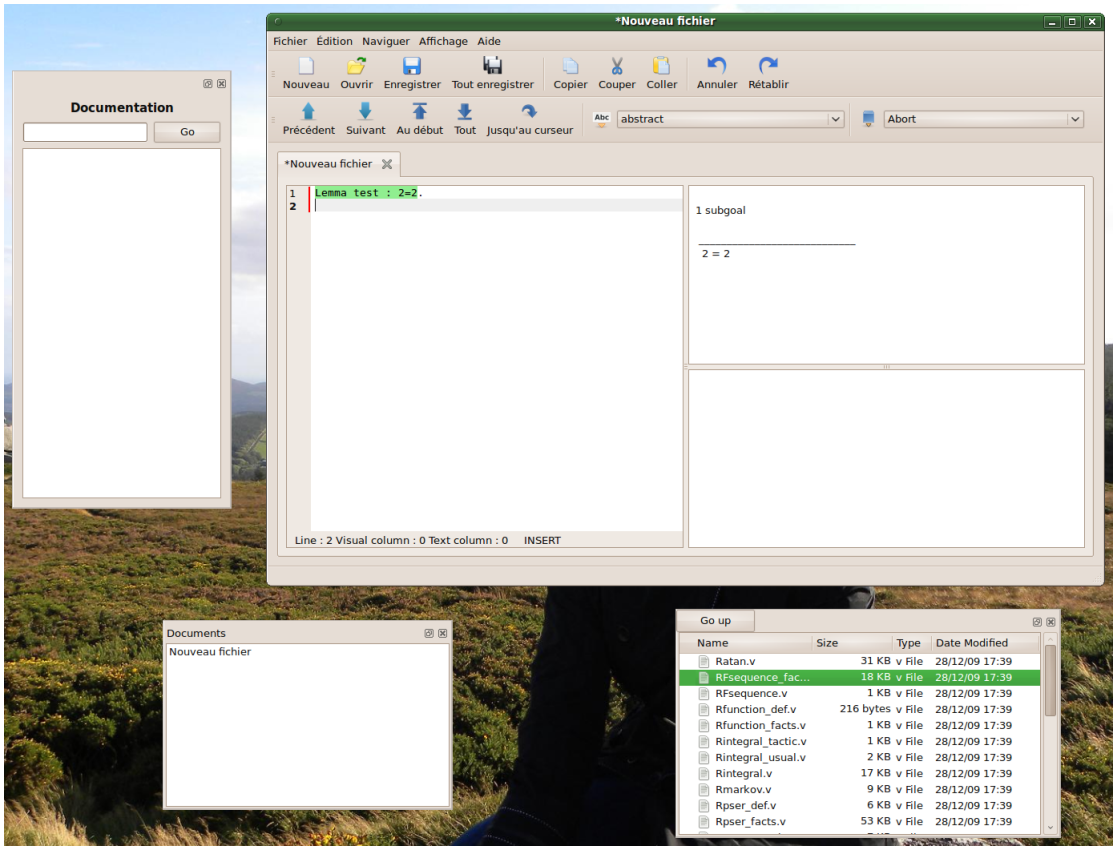


FIG. 3.8 – Les panneaux optionnels à l'extérieur

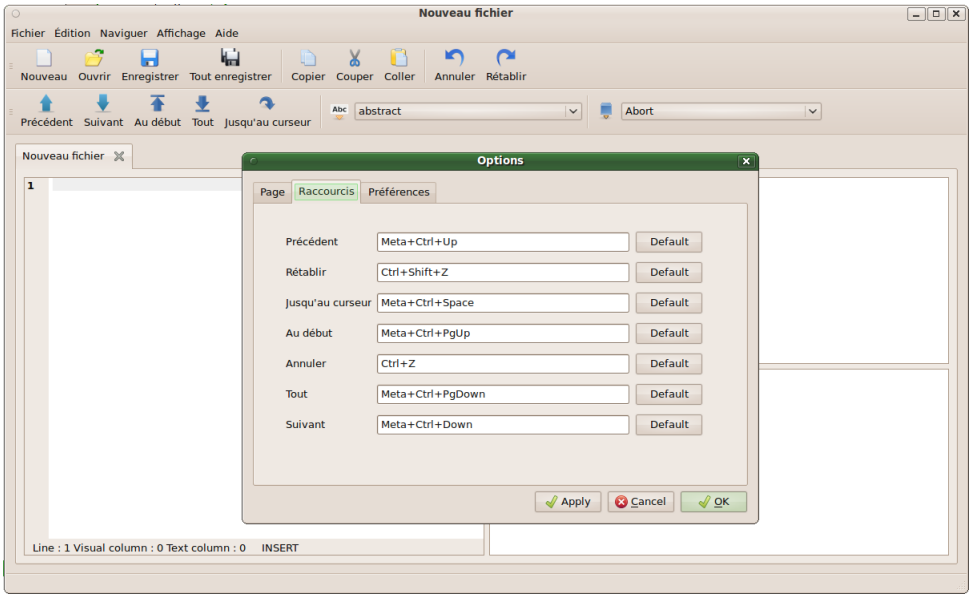


FIG. 3.9 – La fenêtre de modification des préférences

Chapitre 4

Le WP Apprentissage

4.1 Objectifs

Lorsque l'on prouve des théorèmes, on a souvent l'impression de faire la même chose, on obtient des réflexes dans de nombreuses situations. Dans les domaines les plus utilisés, il existe maintenant des tactiques résolvant ces opérations répétitives (`auto` de manière générale, `omega` pour les entiers naturels, `field` dans les corps...).

Cependant, lorsque l'on travaille sur des outils plus rares, ou des outils que l'on a soi-même définis, il n'existe pas de tactiques prédéfinies.

Une solution serait de construire soi-même ses propres tactiques lorsque l'on définit de nouveaux concepts. Malheureusement, cela est difficile. D'une part, parce que `Ltac` est assez difficile à maîtriser, d'autre part parce qu'il est difficile de transformer un savoir-faire composé de réflexes en une description de ce savoir-faire.

L'objectif de ce groupe de travail est de remédier à ce problème. On cherche donc à créer automatiquement, à partir de nombreuses démonstrations, des tactiques permettant de faire des démonstrations similaires.

4.2 Les arbres de décisions

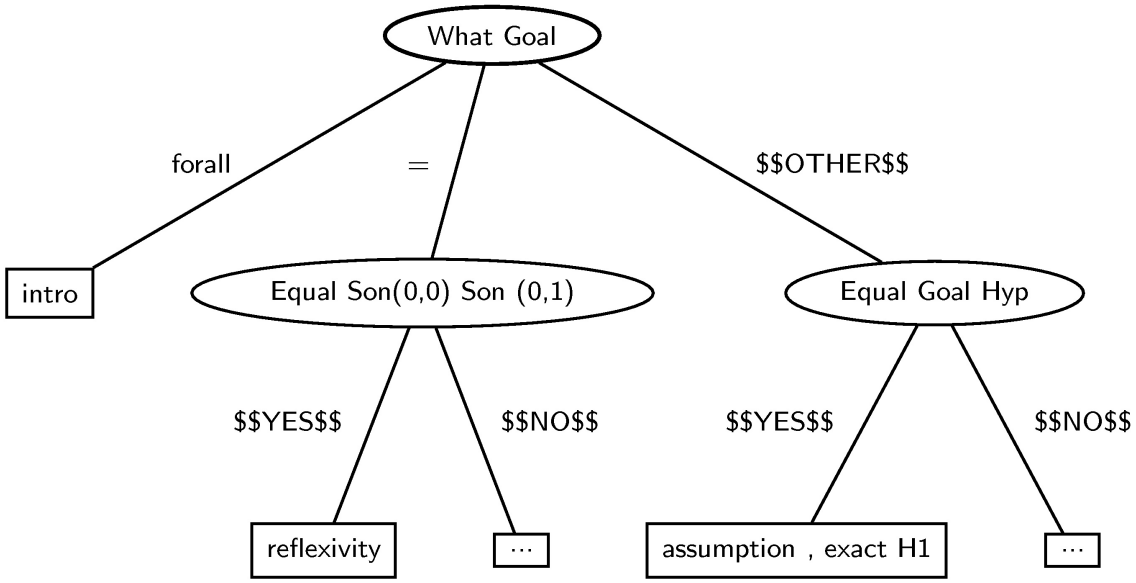
Pourquoi des arbres de décision Après une revue des méthodes classiques d'apprentissage, nous avons résolu des théorèmes en Coq en analysant à chaque étape notre raisonnement. Nous avons remarqué que nous réflechissions le plus souvent comme des arbres de décision. Nous nous interrogeons sur la structure des hypothèses et du but, à commencer par les nœuds de l'arbre syntaxique de moindre profondeur. La racine du but nous donne assez souvent, à elle seule, la tactique à utiliser. La plupart du temps, si le but est `T & U` on utilise `split`, si c'est `forall x, T` ou `T -> U`, on utilise `intro`.

Définition des arbres de décision Un arbre de décision est défini par induction comme étant :
- une liste de réponse ;
- une question sur la situation, et pour chaque valeur un arbre de décision.

Possibilités pour les questions D'après ce que nous avons dit plus haut, il est logique de poser les questions sur le haut des arbres. Donc on s'autorise à interroger le nœud à la racine des arbres syntaxiques des hypothèses et du but, ainsi que les fils des arbres déjà interrogés. Les questions qu'on leur pose sont `What addr` (quel est le *token* à l'adresse `addr` ?) et `Equal addr1 addr2` (est-ce que les *tokens* à aux adresses `addr1` et `addr2` sont les mêmes ?).

Exemple Ici, si le but est de la forme `T = T`, on répondra `reflexivity`. `Son(i,j)` désigne le j-ième fils de l'arbre interrogé à la profondeur i.

Si la valeur ne correspond à aucune autre, on emprunte la branche `$$OTHER$$`.



4.3 Organisation des modules

4.3.1 Fouille sur le net / Résolution d'exercices

Un premier travail est de rassembler un grand nombre de démonstrations dans un même domaine. Nous nous sommes essayés sur deux domaines : la logique du premier ordre et les comparaisons d'entiers. Pour des tactiques complètes pour ces domaines, il faudrait plus de lignes de démonstrations. Mais ces collections suffisent à tester notre algorithme.

4.3.2 *Parsing*, discussion avec Coqtop

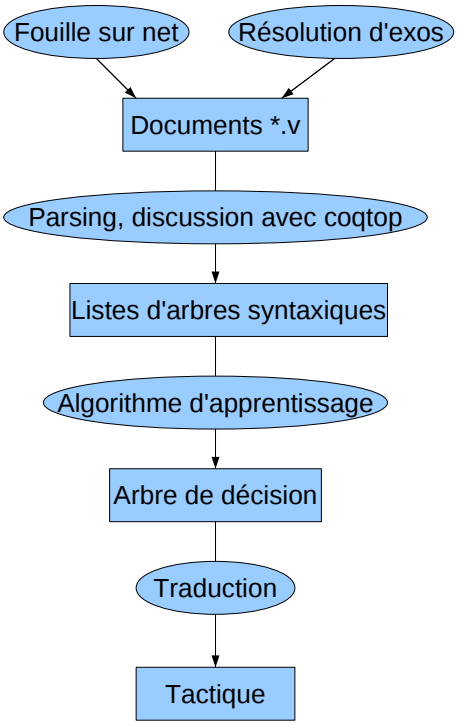
Les fichiers que l'on récolte sont des fichiers coqs normaux. Il faut ensuite en extraire les exemples que l'on va apprendre et les mettres sous la forme que l'algorithme d'apprentissage prend en paramètre.

4.3.3 Algorithme d'apprentissage

L'algorithme d'apprentissage transforme ces données expérimentales en un arbre de décision.

4.3.4 Traduction

L'arbre est transformé automatiquement en une tactique, qui pourra ensuite être utilisée telle quelle.



4.4 Parsing, discussion avec coqtop

Il faut aussi analyser les fichiers de preuves `.v` pour déterminer les tactiques. Cette analyse est spécifique et on n'a pas besoin d'analyser les fichiers vernaculaires d'une manière aussi compliquée que celle de Coq, qui utilise un nombre important de fichiers de grammaires. Il est important également de prendre en compte les nombreuses tactiques comme `apply (H H2) in H3` qui ne sont pas de simples chaînes de caractères monolithiques (comme `assumption` ou `constructor 2`) et il est pour cela nécessaire de repérer les identificateurs dans une chaîne de caractères. Comme Coq est très permissif sur les caractères autorisés dans les identificateurs, il est possible que certains accents ne soient pas pris en compte durant l'apprentissage (ce qui risque de générer un résultat moins précis à l'apprentissage, mais ne génère pas d'erreur bloquante). Il est à remarquer que la prise en compte dans les tactiques des identificateurs présents dans le contexte (et plus généralement le mode d'apprentissage très permissif) permet d'appréhender différentes formes (éventuellement imprévues) du langage de tactique, comme `[ .. | .. | .. ]`, `;` ou `||` par exemple. Pour les fichiers `.v`, on utilise un système qui combine également l'analyse *ad hoc* et les analyseurs `ocamllex`/`ocamlyacc` pour le schéma global d'un tel fichier (mais pas pour les termes utilisés dans les tactiques).

Finalement, après avoir combiné les contextes avec les tactiques et vérifié leur cohérence numérique, le traitement des données nécessite la transformation des identificateurs en indices de De Bruijn, la mise en une forme un peu plus canonique des termes et des expressions, la mise en forme spécifique à l'algorithme d'apprentissage en prenant soin de conserver une forme la plus générale et abstraite possible (le moins de variables libres possibles) pour espérer de meilleures performances d'apprentissage.

4.5 Apprentissage

Entropie

On considère un ensemble $\{x_i\}_{i\in I}$. On tire au hasard n fois un x_i (x_j est pris avec la probabilité p_j). Le meilleur code décrivant la suite des éléments tirés a une espérance de longueur de $n\sum_{i\in I} -p_i \log p_i$. On définit alors l'entropie du tirage comme $\sum_{i\in I} -p_i \log p_i$, soit environ la longueur moyenne du code décrivant quel élément a été tiré. En particulier, on remarque que si une seule réponse est possible, l'entropie est nulle.

Application de l'entropie aux arbres de décision

Nous voulons que, à chaque feuille de l'arbre de décision, l'entropie soit minimale. C'est-à-dire que, pour les exemples conduisant à une même feuille, la tactique à utiliser soit toujours la même. Il faut donc diminuer l'entropie à chaque étape. On veut que l'arbre soit de taille minimale, donc il faut maximiser la perte d'entropie moyenne sur le nombre de nœuds total de l'arbre de décision.

La maximisation exacte n'est *a priori* même pas dans $\mathcal{NP}$. Il paraît donc préferable d'utiliser une heuristique. À chaque nœud de l'arbre de décision, on cherche à maximiser la perte d'entropie sur le nombre de fils. Pour cela, pour toutes les questions possibles, on mesure l'entropie moyenne après la question.

Difficultés surmontées

Certes, nous nous basons sur un algorithme déjà connu. Mais notre situation est plus compliquée et il a fallu résoudre un certain nombre de problèmes.

Multiplicité des réponses Un même exemple peut avoir plusieurs réponses à une même question. En effet, si il y a plusieurs hypothèses, il y aura plusieurs réponses à la question `What Hypothese ?`.

Renumérotation des hypothèses Lorsque les exemples arrivent à l'algorithme d'apprentissage, la réponse `apply H0` sera écrite `("apply %", [i])` avec i l'indice de H_0 dans la liste des hypothèses. Or, ce i ne veut rien dire. Si l'hypothèse H_0 avait été à un autre endroit, on aurait quand même appelé H_0. Il acquiert du sens seulement si, dans le chemin que l'on fait dans l'arbre de décision pour arriver jusqu'à la réponse `apply H0`, on pose une question sur une hypothèse et que c'est H_0 qui y répond. On représentera alors `apply H0` par `("apply %", [j])` avec j la profondeur de la question où H_0 a répondu. Or il peut y avoir plusieurs hypothèses répondant à la question, on peut se rendre compte plus tard que l'une d'elles n'y répond pas, on peut ne pas encore avoir trouvé de question où H_0 répond...

Grand nombre de réponses *A priori*, n'importe quelle fonction ou constante du langage (y compris celles définies par l'utilisateur) peuvent être des réponses. Pour ne pas surcharger l'arbre, dans le cas où seulement certaines réponses sont intéressantes, on a créé la réponse `$$OTHER$$` dans laquelle on met les réponses faisant perdre le moins d'entropie.

Questions non fixes Le fait d'interroger une question permet ensuite d'interroger ses fils.

Types complexes Les types des objets sont souvent grands, en témoigne le type `(galinaRoot * int * (id * answer) list * (nodeAddress * ((galinaRoot * ((id * (int * galinaTerm)) list)) list)) list) list` manipulé à plusieurs endroits du programme. Sachant que `galinaRoot`, `galinaTerm`, `nodeAddress` answer sont encore des types composés.

Concepts proches Il est souvent ardu de s'y retrouver dans les programmes car il y a beaucoup de structures se ressemblant mais qu'il ne faut pas confondre. Par exemple, on peut considérer une hypothèse comme un terme de galina, un *token* de galina ou une adresse à laquelle on peut poser une question. Il y a les arbres de décision et les arbres syntaxiques. Les réponses à un environnement, les réponses aux questions que l'on pose sur l'environnement. Les tactiques « élémentaires » qui sont les réponses dans les exemples d'apprentissage et la tactique qui résultera de l'arbre de décision.

4.6 Pipeau ?

L'apprentissage automatique est souvent accusé d'être pipeau. Il y a effectivement une part non négligeable d'approximations et de suppositions dans cet algorithme. Nous allons les lister ici et voir dans quelle mesure elles sont raisonnables.

Supposition de distribution représentative

L'algorithme n'a de sens que si la distribution des tactiques qu'il faut utiliser en fonction des réponses aux questions que l'on peut poser sur l'environnement est la même dans les exemples d'apprentissage et dans les exemples où va être utilisé l'arbre de décision.

Les exemples utilisés pour inférer l'arbre et les exemples où l'arbre sera utilisé sont pris dans le même domaine. Donc, d'après la loi des grands nombres, pour un nombre d'exemple suffisamment grand, la distribution dans les exemples sera aussi proche de la distribution réelle que l'on veut. Le nombre de questions est relativement petit, ce qui accélère la convergence. Il y a 2 questions au départ, et dans la pratique ce nombre augmente peu. Chaque question posée supprime cette question (sauf les questions sur les hypothèses) et rajoute autant de questions qu'il y a de fils. Les arités les plus courantes sont 1 et 2 donc le nombre de questions possibles stagne ou augmente de 1 la plupart du temps. De plus, plus le domaine dans lequel nous générons la tactique est restreint, plus cette convergence est accélérée.

L'arbre n'est pas minimal

Pour déterminer les tactiques en minimisant la taille de l'arbre, on utilise une heuristique. Considérons un grand nombre d'exemples. Les buts sont toujours `false`. Dans la moitié des exemples, les hypothèses contiennent `H:True` $\rightarrow$ `False`, `H':0=0` et des hypothèses aléatoires et la tactique utilisée est `apply H`. Dans l'autre moitié, les hypothèses contiennent `H:True` $\rightarrow$ `True`, `H':0=1` et des hypothèses aléatoires, et la tactique utilisée est `inversion H'`. Demander si il y a une hypothèse dont la racine est un `=` ou un $\rightarrow$ n'apportera aucune information car tous les exemples ont de telles hypothèses. Donc l'algorithme ne posera pas ces questions-là. Donc il n'interrogera jamais les fils droits de H et H' qui sont pourtant les seules informations réellement pertinentes.

Nous n'avons donc aucune borne d'approximation pour cette heuristique. La seule confiance que nous pouvons lui faire est l'intuition (notre exemple est quand même tiré par les cheveux, l'heuristique semble marcher) et l'expérience.

Choix des questions

Il arrive que le raisonnement que fait le mathématicien ne puisse pas être traduit dans les termes de l'arbre de décision. Considérons le problème de l'égalité de deux conjonctions : $a\&(b\&c)\&(d\&e) = (b\&a)\&(d\&c)\&e$. Le mathématicien va d'abord vérifier que les listes de variables sont les mêmes des deux côtés. Cette question a un caractère global que l'on ne peut pas capturer avec nos questions `Equal` et `What`.

Ca semble la limite la plus sévère à notre algorithme. Il faudrait rajouter des questions. Oui mais lesquelles ? Si l'on autorise un trop grand nombre de questions par rapport au nombre d'exemples, il y a de fortes probabilités pour qu'une question soit considérée comme intéressante alors qu'elle ne l'est pas.

4.7 Résultats

Nous avons rempli nos objectifs de base : l'extraction de données, algorithme d'apprentissage et algorithme de parcours d'un arbre de décision à partir d'un problème. De plus, nous avons fait deux des bonus qu'on avait prévus : la compréhension des hypothèses ainsi que la transformation en une tactique.

Les tactiques générées semblent satisfaisantes sur de petits ensembles. Malheureusement, nous n'avons pas eu le temps de l'essayer à grande échelle et sur des problèmes complexes.

4.8 Perspectives

Compréhension des termes De même que l'on a réussi à comprendre H dans `apply H` comme un argument correspondant à une hypothèse, il faudrait réussir à comprendre `cut (z=0)`. Mais ce n'est pas aussi simple. *A priori* `z=0` ne se retrouve nulle part. Ou pas entièrement. On pourrait voir, dans les arbres syntaxiques interrogés, lesquels ont le plus grand sous-arbre commun avec `z=0`.

Connexion avec l'IDE Il serait bien que, depuis l'IDE, l'utilisateur puisse dire quelles preuves il veut rajouter dans les bases d'exemples pour quels domaines. Les tactiques serait réactualisées régulièrement pour correspondre aux nouveaux exemples.

Bases d'exemples sur Internet Plus il y a de preuves, meilleures seront les tactiques. On peut imaginer de grandes bases de données sur Internet. Chaque utilisateur de Coquille les utiliserait et les alimenterait.

Update à la volée Actuellement, si on veut rajouter des exemples, il faut refaire entièrement l'arbre. On pourrait chercher un moyen pour actualiser l'arbre de décision avec les nouveaux exemples même si on a effacé les exemples d'avant.

Chapitre 5

Le WP communication

5.1 Objectifs

L'objectif de ce groupe de travail est de faciliter la coordination des autres groupes de travail et de promouvoir le projet et ses résultats auprès de potentiels utilisateurs.

5.2 Matériels et méthodes

Pour atteindre ces objectifs, ce groupe de travail a réalisé :
- un site Internet ;
- une présentation du projet, notamment de nos résultats les plus intéressants ;
- une page de contacts et la documentation de Coquille pour faciliter son utilisation.

Ce groupe a par ailleurs réalisé la rédaction des différents rapports hebdomadaires et coordonné la rédaction du rapport de mi-projet. Enfin, des documents de communications internes et des formations HTML et LaTeX ont aussi été réalisées.

5.3 Résultats

Les principales réalisations de ce groupe de travail sont le site Internet et le présent rapport. La documentation est disponible sur le site Internet, pour tenir compte de l'avancée des WP. Ce groupe de travail a réalisé les diapositives de la démonstration du projet.

Bibliographie

[1] Yohji Akama, Stefano Berardi, Susumu Hayashi, and Ulrich Kohlenbach. An Arithmetical Hierarchy of the Law of Excluded Middle and Related Principles. *Logic in Computer Science, Symposium on*, 2004.

[2] Alberto Ciaffaglione and Pietro Di Gianantonio. A Tour with Constructive Real Numbers. 2000.

[3] F. Dechesne. Archives automath. `http://www.win.tue.nl/automath/`.

[4] Hugo Herbelin. Proposed naming conventions for the Coq standard library. Technical report, INRIA, 2009.

[5] Cezary Kaliszyk and Russell O'Connor. Computing with Classical Real Numbers. *Journal of Formalized Reasoning*, 2008.

Chapitre 6

Annexe 1 : documentation de l'IDE