Computational Logic

Master IP Paris – Cyber-Physical Systems

Implementing a propositional prover

Émilie THOMÉ

December 2020

1 Introduction

Le but de ce TD était de mettre en place un prouveur de logique propositionnelle qui permet donc de vérifier des preuves et de les élaborer facilement.

Je n'ai pas mis tout le code du TD dans un seul fichier `prove.ml` car je ne trouvais pas cela très pratique. En effet, j'ai dans un premier temps réalisé des tests conduisant intensionnellement à des erreurs. De plus, nous avons deux preuves interactives différentes (avec souvent des noms de fonctions similaires). Donc ce TD est séparé en trois fichiers de code :

- `prover_tests.ml`, représentant le code jusqu'à 4.1 avec des tests mais sans le prouveur intéractif, plutôt adapté à une visualisation sur Emacs,
- `prover_notests.ml`, représentant le code jusqu'à 4.1 avec le prouveur intéractif mais sans les tests (les deux fichiers ont été comparé et en effet il n'y a que cette différence),
- `prover_dependent_type.ml`, représentant la partie sur les types dépendants.

Ce TD a été majoritairement fait pendant les vacances de la Toussaint. C'est pour cela que je ne me rappelle plus exactement de ce qui a été fait. Je ne crois pas avoir été au courant d'un rapport à la Toussaint donc je n'avais pas pris de notes sur ma progression.

2 Contenu

J'ai réalisé l'entièreté du TD mis à part les questions optionnelles suivantes (ce qui est proche de la totalité des questions optionnelles) :

- 4.2 Optional: Declarations
- 4.3 Optional: Other inductive types
- 5.15 Optional: inductive types
- 5.16 Optional: interactive prover
- 5.17 Optional: universes
- 5.18 Optional: better handling of indices

Je crois même que la question optionnelle 4.1 n'a pas été entièrement traitée (en tout cas elle n'a pas été amalgamée dans l'ensemble du projet).

3 Fonctionnement

Ce TD est conçus pour vous permettre de ne pas avoir à écrire à la main dans le terminal.

Il y a donc un `Makefile` permettant de compiler les deux fichiers `prover_notests.ml prover_dependent_type.ml` mais aussi de montrer les preuves déjà réalisées. Le program dans `prover_tests.ml` n'est pas fait pour être compiler dans le terminal et être utilisé directement car il lève des erreur (volontairement, afin de voir que le code est correctement faux).

Un dossier `src/` contient les trois fichiers de code.

Les deux prouveurs automatiques enregistrent les preuves dans un fichier `k.proof` que l'on peut renommer une fois la preuve terminée. Il sera enregistrer dans le dossier `proof/`. J'ai modifié le code des types dépendants pour que le fichier `interactive.proof` ne soit plus créé et que la preuve soit directement mise avec les autres preuves dans le fichier `k.proof` que l'on peut renommer.

Dans le prouveur avec les types dépendants, le context de départ (nommé `env`) est modifié afin d'avoir directement les preuves réalisées. Ces dernières ne sont présentent que dans l'environnement car je trouvais plus simple de les coder directement dans le fichier code plutôt qu'avec le prouveur automatique (si je me trompe je peux modifier simplement).

```
let env = ref [
  ("com_mult", (infer [] com_mult, Some com_mult)) ;
  ("mult_1", (infer [] mult_1, Some mult_1)) ;
  ("dev_mult_add", (infer [] dev_mult_add, Some dev_mult_add)) ;
  ("multz", (infer [] multz, Some multz)) ;
  ("zmult", (infer [] zmult, Some zmult)) ;
  ("ass_mult", (infer [] ass_mult, Some ass_mult)) ;
  ("dev_add_mult", (infer [] dev_add_mult, Some dev_add_mult)) ;
  ("mult", (infer [] mult, Some mult)) ;
  ("com_add", (infer [] com_add, Some com_add)) ;
  ("com_1", (infer [] com_1, Some com_1)) ;
  ("ass_add", (infer [] ass_add, Some ass_add)) ;
  ("Cong", (infer [] cong, Some cong)) ;
  ("Sym", (infer [] sym, Some sym)) ;
  ("Trans", (infer [] trans, Some trans)) ;
  ("addz", (infer [] addz, Some addz)) ;
  ("zadd", (infer [] zadd, Some zadd)) ;
  ("pred", (infer [] pred, Some pred)) ;
  ("Seq", (infer [] seq, Some seq)) ;
  ("add", (infer [] add, Some add))] in [...]
```

Les preuves de l'environnement sont préalablement vérifiée.

4 Difficultés

La partie la plus difficile était celle des preuves avec les types dépendants. Non pas que ce soit particulièrement complexe, mais c'est beaucoup plus laborieux d'écrire la preuve avec notre prouveur que sur papier. Sur papier, la preuve prend quelques lignes mais avec la sémantique du prouveur réalisé cela prend des lignes et des lignes.

De plus je me suis rendue compte qu'il fallait faire attention aux noms de variables que l'on utilise. Parce que lors de la vérification :

```
let b_s = conv context infer_s (Pi ("n", Nat, Pi ("m",
                                 App (p, Var "n"),
                                 App (p, S (Var "n")))))
```

La variable "n" était déjà utilisée dans p, sauf qu'on ne peut plus rien faire pour cette erreur lorsque qu'elle est commise dans le code. Donc j'ai remplacé automatiquement "n" et "m" par de nouvelles variables. C'est maintenant évident mais je me suis vue passer deux heures à chercher une erreur dans mes centaines de lignes de preuves alors que l'erreur était simplement là. Donc je voulais souligner que le debuggage n'est pas évident avec ce prouveur automatique car laborieux.

5 Améliorations

Si je n'avais pas arrêté le TD il y a plus d'un mois de cela, j'aurais pû :

- faire les options proposées
- rentrer les preuves automatiques enregistrer dans le contexte de départ
- vérifier plus en profondeur le code et le rendre plus efficace et lisible

6 Conclusion

C'était un TD en effet assez long mais très intéressant. Il était très satisafaisant de passer plusieurs heures pour trouver une petite erreur. J'espère que les modifications apportées ne vous dérangeront pas. Merci pour nous avoir proposer ce travail.