Assistant de preuves pour le calcul propositionnel

Maxime Lesourd

TIPE dirigé par Damien Pous

1 Introduction

Le but de ce TIPE est l'étude d'une formalisation du raisonnement mathématique : la déduction naturelle et la conception d'un programme interactif en OCaml permettant de prouver des formules du calcul propositionel. Les définitions sont tirées de **Karim Nour, René David, Christophe Raffalli** *Introduction à la logique : Théorie de la démonstration - Cours et exercices corrigés*, ed. Dunod.

Table des matières

1 Introduction **1**

2 Calcul propositionel **2**
2.1 Formules . . . 2
2.2 Règles . . . 2
2.3 Déduction naturelle . . . 3
2.4 Preuves . . . 4
2.5 Dérivations de règles . . . 7
2.6 Un système alternatif . . . 9
2.7 Équivalence des systèmes . . . 10

3 Principe du programme **11**
3.1 Construction des preuves . . . 11

4 Manuel d'utilisation **13**
4.1 Syntaxe des formules et séquents . . . 13
4.2 Liste des commandes . . . 13
4.3 Construction des tactiques . . . 14
4.4 Exemples de preuves . . . 15

2 Calcul propositionel

2.1 Formules

Les formules du calcul propositionel sont construites à partir de variables, du symbole $\bot$ et des connecteurs $\wedge, \vee, \rightarrow$ et $\neg$. Plus formellement si $\mathcal{V}$ désigne un ensemble de variables, on peut définir l'ensemble des formules du calcul propositionel sur $\mathcal{V}$ comme le plus petit ensemble contenant $\mathcal{V} \cup \{\bot\}$ et stable par application de $\wedge, \vee, \rightarrow$ et $\neg$.

Notation : Formules

On note $\mathcal{F} := \mathcal{V} \mid \bot \mid \neg\mathcal{F} \mid \mathcal{F} \wedge \mathcal{F} \mid \mathcal{F} \vee \mathcal{F}$

Exemple :

P, $\neg P$, $P \vee Q$, $P \vee Q \rightarrow \neg R$, $P \wedge \neg P$ sont des formules sur $\mathcal{V} = \{P; Q; R\}$

Remarque :

Pour donner du sens à la formule $P \vee Q \rightarrow \neg R$ il faut définir un ordre de priorité pour les connecteurs. Dans les formules mathématiques c'est le symbole $\times$ qui a une plus grande priorité que $+$, ici on a par ordre croissant de priorité :

1. $\rightarrow$
2. $\vee$
3. $\wedge$
4. $\neg$

La formule $P \vee Q \rightarrow \neg R$ se lit donc $(P \vee Q) \rightarrow (\neg R)$

Remarque :

On aura souvent à considérer des ensembles de formules, si Γ est un ensemble de formules et A une formule on notera $\Gamma, A := \Gamma \cup \{A\}$

2.2 Règles

Un théorème est composé de deux parties : des hypothèses et une conclusion. Cette structure se retrouve dans la notion de séquent.

Notation : Séquent

Soit C une formule, Γ un ensemble de formule on note $\Gamma \vdash C$ le séquent qui se lit "Γ prouve C" ou "Γ thèse C".
Γ représente donc les hypothèses et C la conclusion.

La construction d'une preuve se fait selon des règles.

Notation : Règle

Une règle est composée d'un ensemble de séquents : les prémisses $S_1, \ldots, S_n$ et d'un séquent conclusion : C. Elle est notée $\dfrac{S_1 \quad \ldots \quad S_n}{C}$ et se lit "Pour prouver C il suffit de prouver $S_1, \ldots, S_n$" ou "Si on a prouvé $S_1, \ldots, S_n$ alors on a prouvé C".

Exemple :

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge_i$$

Pour utiliser une règle on remplace toutes les occurences d'une lettre par une même formule.

Exemple :

En utilisant la règle précédente avec $\Gamma = \emptyset, A = A, B = \neg A$ on obtient

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash \neg A}{\Gamma \vdash A \wedge \neg A} \wedge_i$$

2.3 Déduction naturelle

La déduction naturelle est composée des règles suivantes :

Introduction des connecteurs

Dans ces règles le connecteur apparait dans la conclusion.

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to_i \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge_i \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee_i^g \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee_i^d \qquad \frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A} \neg_i$$

Élimination des connecteurs

Dans ces règles le connecteur apparait dans les prémisses.

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash A \to B}{\Gamma \vdash B} \to_e \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge_e^g \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge_e^d$$

$$\frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C} \vee_e \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash \neg A}{\Gamma \vdash \bot} \neg_e$$

Règles structurelles

$$\frac{}{\Gamma, A \vdash A} axiome \qquad \frac{\Gamma \vdash A}{\Gamma, B \vdash A} aff$$

Remarque :

La règle d'affaiblissement sera souvent utilisée de manière implicite dans les démonstrations. On peut réécrire toute preuve utilisant des affaiblissements sans ces affaiblissements, on dit que cette règle est admissible.

Absurdité classique

Cette règle correspond au raisonnement par l'absurde :

$$\frac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash A} \bot_c$$

2.4 Preuves

Pour prouver un séquent on lui applique une règle puis on prouve les prémisses. On construit ainsi une preuve à partir de la conclusion. On présente ici deux notations des preuves : l'une arborescente, l'autre linéaire.

Exemple :

On prouve le séquent $\vdash \neg\neg A \to A$
Notation arborescente :

$$\dfrac{\dfrac{\dfrac{}{\neg\neg A, \neg A \vdash \neg A}\ axiome \qquad \dfrac{}{\neg\neg A, \neg A \vdash \neg\neg A}\ axiome}{\neg\neg A, \neg A \vdash \bot}\ \neg_{\mathrm{E}}}{\neg\neg A \vdash A}\ \bot_{\mathrm{C}}}{\vdash \neg\neg A \to A}\ \to_{\mathrm{D}}$$

On écrit le séquent à prouver puis on superpose les règles appliquées jusqu'aux axiomes.

Notation linéaire :

$\vdash \neg\neg A \to A$	$\to_{\mathrm{i}}$
$\neg\neg A \vdash A$	$\bot_{\mathrm{c}}$
$\neg\neg A, \neg A \vdash \bot$	$\neg_{\mathrm{e}}(1)(2)$
$(1)\ \neg\neg A, \neg A \vdash \neg A$	*axiome*
$(2)\ \neg\neg A, \neg A \vdash \neg\neg A$	*axiome*

On écrit le séquent à prouver à gauche et la règle appliquée à droite, si la règle a une seule prémisse on l'écrit dessous, sinon on numérote les prémisses créées et on les écrit dessous.

La notation arborescente est plus lisible pour des preuves courtes mais son écriture de bas en haut est peu adaptée à la recherche de preuves à la main. Les exemples suivants seront donc présentés en notation linéaire.

Prouvabilité

Définition : Séquent prouvable

On dit d'un séquent $\Gamma \vdash F$ qu'il est prouvable s'il peut être obtenu par application d'un nombre fini de règles. Par extension une formule F est prouvable si $\vdash F$ l'est.

Notation : $\vdash$

On note $\Gamma \vdash F$ la phrase *"Le séquent* $\Gamma \vdash \mathrm{F}$ *est prouvable"*.

Remarque :

Cette notation est ambiguë car $\Gamma \vdash F$ peut représenter soit un séquent, soit l'affirmation que ce séquent est prouvable. Si le contexte ne permet pas de déterminer la signification du symbole $\vdash$ il faut la préciser.

Cette notion est purement symbolique, on n'a jamais parlé de vrai ou faux.

Définition : Distribution de valeur de vérité

Une distribution de valeur de vérité est une fonction $d : \mathcal{V} \to \{0, 1\}$

Définition : Évaluation d'une formule

Soit d une distribution de valeur de vérité, on peut la prolonger à $\mathcal{F}$ en posant :
- $d(\bot) = 0$
- $d(\neg F) = 1 \; ssi \; d(F) = 0$
- $d(F_1 \wedge F_2) = 1 \; ssi \; d(F_1) = 1$ et $d(F_2) = 1$
- $d(F_1 \vee F_2) = 1 \; ssi \; d(F_1) = 1$ ou $d(F_2) = 1$
- $d(F_1 \to F_2) = 1 \; ssi \; d(F_1) = 0$ ou $d(F_2) = 1$

Définition : Tautologie

Une formule F est une tautologie si pour toute distribution de valeur de vérité d on a $d(F) = 1$.

Le théorème de complétude du calcul propositionel garantit que dans le système de la déduction naturelle on a : $\vdash F \leftrightarrow F$ est une tautologie.

Exemples de preuves

On montre ici les lois de De Morgan :

$$\neg(A \vee B) \leftrightarrow \neg A \wedge \neg B$$
$$\neg(A \wedge B) \leftrightarrow \neg A \vee \neg B$$

Exemple :

$$\begin{array}{ll}
\vdash \neg(A \vee B) \to \neg A \wedge \neg B & \to_{\mathrm{i}} \\
\neg(A \vee B) \vdash \neg A \wedge \neg B & \wedge_{\mathrm{i}}(1)(2) \\
(1)\, \neg(A \vee B) \vdash \neg A & \neg_{\mathrm{i}} \\
\quad \neg(A \vee B), A \vdash \bot & \neg_{\mathrm{e}}(a)(b) \\
\quad (\mathrm{a})\, \neg(A \vee B), A \vdash (A \vee B) & \vee_{\mathrm{i}}^{\mathrm{g}} \\
\quad\quad \neg(A \vee B), A \vdash A & \mathit{axiome} \\
\quad (\mathrm{b})\, \neg(A \vee B), A \vdash \neg(A \vee B) & \mathit{axiome} \\
(2)\, \neg(A \vee B) \vdash \neg B & \neg_{\mathrm{i}} \\
\quad \neg(A \vee B), B \vdash \bot & \neg_{\mathrm{e}}(c)(d) \\
\quad (\mathrm{c})\, \neg(A \vee B), B \vdash (A \vee B) & \vee_{\mathrm{i}}^{\mathrm{d}} \\
\quad\quad \neg(A \vee B), B \vdash B & \mathit{axiome} \\
\quad (\mathrm{d})\, \neg(A \vee B), B \vdash \neg(A \vee B) & \mathit{axiome}
\end{array}$$

$$\begin{array}{ll}
\vdash \neg A \wedge \neg B \to \neg(A \vee B) & \to_{\mathrm{i}} \\
\neg A \wedge \neg B \vdash \neg(A \vee B) & \neg_{\mathrm{i}} \\
\Gamma := \neg A \wedge \neg B, (A \vee B) \vdash \bot & \vee_{\mathrm{e}}(1)(2)(3) \\
(1)\, \Gamma \vdash (A \vee B) & \mathit{axiome} \\
(2)\, \Gamma, A \vdash \bot & \neg_{\mathrm{e}}(a)(b) \\
\quad (\mathrm{a})\, \Gamma, A \vdash A & \mathit{axiome} \\
\quad (\mathrm{b})\, \Gamma, A \vdash \neg A & \wedge_{\mathrm{e}}^{\mathrm{g}} \\
\quad\quad \Gamma, A \vdash \neg A \wedge \neg B & \mathit{axiome} \\
(3)\, \Gamma, B \vdash \bot & \neg_{\mathrm{e}}(c)(d) \\
\quad (\mathrm{c})\, \Gamma, B \vdash B & \mathit{axiome} \\
\quad (\mathrm{d})\, \Gamma, B \vdash \neg B & \wedge_{\mathrm{e}}^{\mathrm{d}} \\
\quad\quad \Gamma, B \vdash \neg A \wedge \neg B & \mathit{axiome}
\end{array}$$

Exemple :

$\vdash \neg(A \wedge B) \to \neg A \vee \neg B$ $\quad \to_{\mathrm{i}}$
$\neg(A \wedge B) \vdash \neg A \vee \neg B$ $\quad \bot_{\mathrm{c}}$
$\Gamma := \neg(A \wedge B), \neg(\neg A \vee \neg B) \vdash \bot$ $\quad \neg_{\mathrm{e}}(I)(II)$
(I) $\Gamma \vdash \neg A \vee \neg B$ $\quad \vee_{\mathrm{i}}^{\mathrm{g}}$
$\Gamma \vdash \neg A$ $\quad \neg_{\mathrm{i}}$
$\Gamma, A \vdash \bot$ $\quad \neg_{\mathrm{e}}(1)(2)$
(1) $\Gamma, A \vdash \neg A \vee \neg B$ $\quad \vee_{\mathrm{i}}^{\mathrm{d}}$
$\Gamma, A \vdash \neg B$ $\quad \neg_{\mathrm{i}}$
$\Gamma, A, B \vdash \bot$ $\quad \neg_{\mathrm{e}}(a)(b)$
(a) $\Gamma, A, B \vdash A \wedge B$ $\quad \wedge_{\mathrm{i}}(i)(ii)$
(i) $\Gamma, A, B \vdash A$ $\quad axiome$
(ii) $\Gamma, A, B \vdash B$ $\quad axiome$
(b) $\Gamma, A, B \vdash \neg(A \wedge B)$ $\quad axiome$
(2) $\Gamma, A \vdash \neg(\neg A \vee \neg B)$ $\quad axiome$
(II) $\Gamma \vdash \neg(\neg A \vee \neg B)$ $\quad axiome$

$\vdash \neg A \vee \neg B \to \neg(A \wedge B)$ $\quad \to_{\mathrm{i}}$
$\neg A \vee \neg B \vdash \neg(A \wedge B)$ $\quad \neg_{\mathrm{i}}$
$\Gamma := \neg A \vee \neg B, A \wedge B \vdash \bot$ $\quad \vee_{\mathrm{e}}(1)(2)(3)$
(1) $\Gamma \vdash \neg A \vee \neg B$ $\quad axiome$
(2) $\Gamma, \neg A \vdash \bot$ $\quad \neg_{\mathrm{e}}(a)(b)$
(a) $\Gamma, \neg A \vdash A$ $\quad \wedge_{\mathrm{e}}^{\mathrm{g}}$
$\Gamma, \neg A \vdash A \wedge B$ $\quad axiome$
(b) $\Gamma, \neg A \vdash \neg A$ $\quad axiome$
(3) $\Gamma, \neg B \vdash \bot$ $\quad \neg_{\mathrm{e}}(c)(d)$
(c) $\Gamma, \neg B \vdash B$ $\quad \wedge_{\mathrm{e}}^{\mathrm{d}}$
$\Gamma, \neg B \vdash A \wedge B$ $\quad axiome$
(d) $\Gamma, \neg B \vdash \neg B$ $\quad axiome$

2.5 Dérivations de règles

Pour simplifier les preuves on peut regrouper un enchainement de règles pour former une nouvelle règle. On dit alors que cette règle est dérivable.

Absurdes

Il existe plusieurs formulations équivalentes de la règle correspondant au raisonnement par l'absurde notamment :

$$\frac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash A} \bot_c \qquad\qquad \frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A} \bot'$$

On montre que ces règles sont équivalentes ; c'est à dire qu'en choisissant l'une de ces règles comme règle de l'absurde de la déduction naturelle on peut dériver l'autre.

Preuve : $\bot_c \to \bot'$

$\Gamma \vdash A$	$\bot_c$
$\Gamma, \neg A \vdash \bot$	$\neg_{\text{i}}(1)(2)$
$(1)\,\Gamma, \neg A \vdash \neg A$	$axiome$
$(2)\,\Gamma, \neg A \vdash \neg\neg A$	aff
$\Gamma \vdash \neg\neg A$	Prémisse

Preuve : $\bot' \to \bot_c$

$\Gamma \vdash A$	$\bot'$
$\Gamma \vdash \neg\neg A$	$\neg_{\text{i}}$
$\Gamma, \neg A \vdash \bot$	Prémisse

On peut aussi remplacer $\bot_c$ par $\dfrac{\Gamma \vdash \bot}{\Gamma \vdash A}\bot_i$. On obtient alors un système différent : la logique intuitioniste. Celle ci n'est pas complète : il existe des formules vraies que l'on ne peut pas prouver en logique intuitioniste, par exemple $\neg(A \wedge B) \to \neg A \vee \neg B$. Cette règle étant dérivable à partir de la déduction naturelle, toute formule prouvable en logique intuitioniste l'est aussi en logique classique.

Preuve : $\bot_i$ est dérivable

$\Gamma \vdash A$	$\bot_c$
$\Gamma, \neg A \vdash \bot$	aff
$\Gamma \vdash \bot$	Prémisse

Coupure

Dans une preuve longue on montre rarement le résultat directement, on préfère démontrer un certain nombre de lemmes puis les utiliser afin de conclure. La règle de coupure $\dfrac{\Gamma \vdash A \qquad \Gamma, A \vdash B}{\Gamma \vdash B} cut$ correspond à cette idée.

Preuve : *cut* est dérivable

$\Gamma \vdash B$	$\to_e (1)(2)$
$(1)\,\Gamma \vdash A$	Prémisse
$(2)\,\Gamma \vdash A \to B$	$\to_i$
$\Gamma, A \vdash B$	Prémisse

Règles gauches

Le système de la déduction naturelle ne fournit pas directement de règles permettant de travailler sur les hypothèses, on peut cependant dériver de telles règles :

Proposition : Règles gauches

Les règles suivantes sont dérivables en déduction naturelle :

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C} \wedge_g \qquad \frac{\Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma, A \vee B \vdash C} \vee_g \qquad \frac{\Gamma \vdash A \qquad \Gamma, B \vdash C}{\Gamma, A \to B \vdash C} \to_g \qquad \frac{\Gamma \vdash A}{\Gamma, \neg A \vdash \bot} \neg_g$$

Preuve : $\vee_g$

$\Gamma, A \vee B \vdash C$ — $\vee_e(1)(2)(3)$
(1) $\Gamma, A \vee B \vdash A \vee B$ — Axiome
(2) $\Gamma, A \vdash C$ — Prémisse
(3) $\Gamma, B \vdash C$ — Prémisse

Preuve : $\to_g$

$\Gamma, A \to B \vdash C$ — cut(1)(2)
(1) $\Gamma, A \to B \vdash B$ — $\to_e$ $(a)(b)$
(a) $\Gamma, A \to B \vdash A \to B$ — Axiome
(b) $\Gamma, A \to B \vdash A$ — Aff
$\Gamma \vdash A$ — Prémisse
(2) $\Gamma, B \vdash C$ — Prémisse

2.6 Un système alternatif

On introduit ici un nouveau système de règles construit à partir des règles gauches et de la coupure :

Règles droites

Ces règles sont les règles d'introduction de la déduction naturelle

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to_d \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge_d \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee_d^g \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee_d^d \qquad \frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A} \neg_d$$

Règles gauches

Dans ces règles le connecteur apparait dans les hypothèses.

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C} \wedge_g \qquad \frac{\Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma, A \vee B \vdash C} \vee_g \qquad \frac{\Gamma \vdash A \qquad \Gamma, B \vdash C}{\Gamma, A \to B \vdash C} \to_g \qquad \frac{\Gamma \vdash A}{\Gamma, \neg A \vdash \bot} \neg_g$$

Règles structurelles

$$\frac{}{\Gamma, A \vdash A}axiome \qquad \frac{\Gamma \vdash A}{\Gamma, B \vdash A}aff \qquad \frac{\Gamma \vdash A \qquad \Gamma, A \vdash B}{\Gamma \vdash B}cut$$

Absurdité classique

C'est la même qu'en déduction naturelle.

$$\frac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash A}\bot_c$$

2.7 Équivalence des systèmes

Notation : $\vdash_{alt}$

On notera $\Gamma \vdash_{alt} F$ (respectivement $\vdash_{alt} F$) un séquent (respectivement une formule) prouvable dans le système alternatif.

On a déjà vu que les règles du système alternatif sont dérivables de la déduction naturelle.

Proposition :

Les règles de la déduction naturelle sont dérivables dans ce système alternatif.

Théorème : Équivalence des systèmes

Soit $\Gamma \vdash F$ un séquent on a : $\Gamma \vdash F \leftrightarrow \Gamma \vdash_{alt} F$

On peut donc utiliser ce système pour prouver toute formule prouvable en déduction naturelle.

3 Principe du programme

Le programme est une boucle interactive, l'utilisateur commence par entrer un séquent à prouver qui devient le premier but. S'ensuit un dialogue : le programme présente l'un des buts à prouver et l'utilisateur choisit une règle à appliquer, si la règle est appliquable le but est remplacé par un ou plusieurs buts. Quand il n'y a plus de but l'utilisateur demande à terminer la preuve qui est alors affichée.

Le principal objectif est la fiabilité, il faut donc pouvoir vérifier que toute preuve produite par le programme est valide. Le code est séparé en modules, chaque module remplissant une fonction. La figure 3 est un graphe de dépendances, une flèche $A \to B$ signifie que le module B utilise le module A. Pour plus de lisibilité certains modules sont groupés [1] .

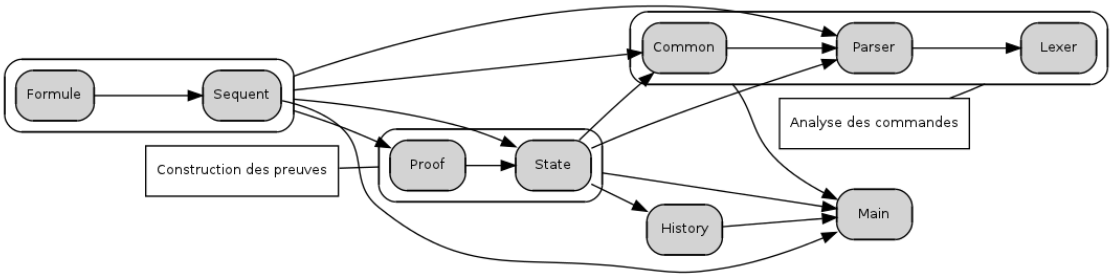

FIGURE 1 – Dépendances des modules

3.1 Construction des preuves

Formules et séquents

Les modules `Formule` et `Séquent` définissent respectivement les types :

- `formule = Bottom | Atom of char | Or of formule*formule | And of formule*formule | Imply of formule*formule | Not of formule`
- `sequent = (string*formule) list * formule`

Dans les séquents les hypothèses sont stockées dans une liste de couples (identifiant,formule) pour pouvoir spécifier quelle formule utiliser dans les règles à gauche sans recopier toute la formule.

Le module Proof

Le module `Proof` définit un type : `proof = sequent*info`. Le premier champ est la formule prouvée, le champ de type `info` ne sert qu'à afficher les étapes de la preuve une fois qu'elle est terminée. Ces types sont protégés, c'est à dire que seules les fonctions du module `Proof` peuvent créer et modifier un objet de type `proof`. Le module définit aussi une fonction par règle qui, étant donné une preuve de chaque prémisse, vérifie que la règle peut être appliquée et renvoie la conclusion de la règle. La définition en tant que fonction des règles permet de vérifier simplement qu'étant donnée une preuve correcte de chaque prémisse la preuve renvoyée sera correcte. On montre ainsi que toute preuve produite par ce module est conforme aux règles de notre système.

L'inconvénient est que pour construire une preuve il faut appliquer successivement les fonctions du module pour arriver à la bonne formule. Cela correspond à écrire les feuilles de la preuve en notation arborescente puis à appliquer les règles jusqu'à obtention de la racine. Cette démarche est réalisable mais ne correspond pas à la recherche de preuve où l'on part de la conclusion pour chercher une preuve. On peut cependant contourner ce problème en construisant la preuve en partant de la conclusion et en appliquant les fonctions du module `Proof` au dernier moment.

1. Un module qui dépend d'un groupe de modules dépend de chacun des modules du groupe.

Le module State

Le module `State` définit un type `state = sequent list * (proof list -> proof)`. Un objet de type `state` correspond à une preuve non terminée, le premier champ est la liste des buts à prouver, le second est une fonction dite partielle qui prend en argument une liste de preuves des buts courants et renvoie une preuve du séquent à prouver. Comme le type `proof` est protégé la fonction qui construit la preuve est construite à partir des fonctions du module `Proof`, elle produit donc une preuve correcte. Pour terminer une preuve il faut arriver à un état dont la liste des buts est vide et appliquer la fonction partielle à la liste vide.

Le module fournit aussi les tactiques, ce sont les fonctions qui permettent d'appliquer les règles au but courant [2] L'implémentation de ces règles dépend fortement du module `Proof` mais même si l'implémentation des tactiques est incorrecte les preuves créées seront correctes : dans le pire des cas la preuve termine et la conclusion est différente, en effet les preuves sont construites des prémisses vers la conclusion.

2. Le but courant est le premier sur la liste.

4 Manuel d'utilisation

4.1 Syntaxe des formules et séquents

Formules

Les formules sont :
- [A – Z] les variables propositionelles
- (*formule*)
- *formule* $\oplus$ *formule* avec $\oplus \in$ { ET, OU, ->}
- !*formule* pour la négation
- _ qui représente $\bot$

Remarque :
Les connecteurs classés par priorité croissante sont : ->, OU, ET, !

Séquents

À l'intérieur d'un séquent les hypothèses sont repérées par un identifieurs formé de caractères alphanumériques et de `.` , par exemple `a.1.1` ou `hyp` sont des identifieurs valides. Si plusieurs hypothèses portent le même identifieur seule la dernière à avoir été ajoutée sera reconnue.

Il y a deux manières d'entrer les séquents :
- i_1`:`f_1 , ... , i_n`:`f_n `|-` *formule* les i_k étant des identifieurs et les f_k des formules.
- f_1 , ... , f_n `|-` *formule* les f_k étant des formules, les identifieurs sont générés automatiquement.

4.2 Liste des commandes

`/load` *fichier.proof* `;;`	interprète le contenu de *fichier.proof* dans la session en cours
`/save` *fichier.proof* `;;`	sauvegarde les commandes entrées depuis le début de la session dans *fichier.proof*
`/undo ;;`	annule la dernière commande
`/exit ;;`	quitte le programme
`/start` *séquent* `;;`	initialise la preuve du séquent
`/apply` *tactique* `;;`	applique la tactique au but sélectionné
`/focus` n `;;`	sélectionne le but n
`/qed ;;`	termine une preuve

4.3 Construction des tactiques

Tactiques élémentaires

Les tactiques correspondent à l'application d'une règle, elles sont présentées sous la forme

$$\texttt{Nom}\ \textit{paramètres} : \frac{\text{But créé} \qquad \text{But créé} \qquad \ldots}{\text{But actuel}}$$

L'application fructueuse d'une tactique remplace le but actuel par un certain nombre de buts créés. Certains paramètres sont facultatifs, ils se trouvent entre [ ].

`Axiom` : $\dfrac{}{\Gamma, i : A \vdash A}$

`Aff` i : $\dfrac{\Gamma \vdash C}{\Gamma, i : A \vdash C}$

`Assume` : $\dfrac{}{\Gamma \vdash A}$ ($\Gamma \vdash A$ est ajoutée aux prémisses)

`Skip` : $\dfrac{\Gamma \vdash A}{\Gamma \vdash A}$

`And_l` i_{ab} [i_a i_b] : $\dfrac{\Gamma, i_a : A, i_b : B \vdash C}{\Gamma, i_{ab} : A \wedge B \vdash C}$

`And_r` : $\dfrac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$

`Or_l` i_{ab} [i_a i_b] : $\dfrac{\Gamma, i_a : A \vdash C \qquad \Gamma, i_b : B \vdash C}{\Gamma, i_{ab} : A \vee B \vdash C}$

`Or_r1` : $\dfrac{\Gamma \vdash A}{\Gamma \vdash A \vee B}$

`Or_r2` : $\dfrac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$

`Imply_l` i_{ab} [i_b] : $\dfrac{\Gamma \vdash A \qquad \Gamma, i_b : B \vdash C}{\Gamma, i_{ab} : A \to B \vdash C}$

`Imply_r` i_a : $\dfrac{\Gamma, i_a : A \vdash B}{\Gamma \vdash A \to B}$

`Neg_l` i : $\dfrac{\Gamma \vdash A}{\Gamma, i : \neg A \vdash \bot}$

`Neg_r` i : $\dfrac{\Gamma, i : A \vdash \bot}{\Gamma \vdash \neg A}$

`Abs` i : $\dfrac{\Gamma, i : \neg A \vdash \bot}{\Gamma \vdash A}$

`Cut` i F : $\dfrac{\Gamma \vdash F \qquad \Gamma, i : F \vdash A}{\Gamma \vdash A}$

Composition

Les tactiques peuvent être composées grace à la construction *tactique*$_1$ **;** *tactique*$_2$ qui applique *tactique*$_1$ au but actuel puis *tactique*$_2$ à chacun des buts créés.

Alternative

Le comportement par défaut lors de l'application d'une tactique invalide est d'annuler l'application et de revenir à l'invite de commande, la structure **try** *tactique*$_1$ [**with** *tactique*$_2$] permet de remplacer *tactique*$_1$ par *tactique*$_2$ en cas d'échec de la première. En l'absence de *tactique*$_2$ `Skip` est utilisée.

4.4 Exemples de preuves

$$\frac{\vdash \neg B \to \neg A}{\vdash A \to B}\ \textsc{Contraposition}$$

```
/start |- A ->B;;
/apply Imply_r a;;
/apply Abs b;;
/apply Cut i !B->!A;;
/apply Assume;;
/apply Imply_l i;;
/apply Axiom;;
/apply Neg_l i;;
/apply Axiom;;
/qed ;;
```

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A, \neg B \vdash \neg B \to \neg A}\ \textsc{Prém} \qquad \dfrac{\dfrac{}{A, \neg B \vdash \neg B}\ \textsc{Ax} \quad \dfrac{\dfrac{}{A, \neg B \vdash A}\ \textsc{Ax}}{A, \neg B, \neg A \vdash \bot}\ \neg_{\text{G}}}{A, \neg B, \neg B \to \neg A \vdash \bot}\ \to_{\text{G}}}{A, \neg B \vdash \bot}\ \textsc{Cut}}{A \vdash B}\ \bot_{\text{C}}}{\vdash A \to B}\ \to_{\text{D}}$$