Structure des preuves
Comprendre le programme construisant des preuves en déduction naturelle

Michel Lévy

27 février 2023

Table des matières

Résumé

Ce document explique la structure des preuves, essentiellement la gestion des hypothèses. Cela peut servir à comprendre comment sont rédigées les preuves mathématiques, mais plus modestement, ce document cherche à expliquer le fonctionnement du programme construisant des preuves en déduction naturelle.

Remerciements

Les prouveurs pour la déduction naturelle, que j'ai écrits et qui sont disponibles à l'adresse teachinglogic, viennent essentiellement du travail de deux personnes.

Le premier est Robert Franz Stärk, qui fut assistant professeur, dans les années 2000, à l'ETH de Zürich. Il avait écrit un prouveur en *Prolog*, qui produisait des preuves

analogues à celles que j'ai adoptées, avec des lignes assume, pour introduire des hypothèses, et des lignes therefore, pour éliminer les hypothèses introduites. C'est aussi Robert Stârk, qui a défini l'interface, écrite en *php*, avec le prouveur. J'ai repris cette interface dans quelques logiciels que j'ai écrits. Cette présentation des preuves est une présentation naturelle, utilisée de façon intuitive et non formalisée par les mathématiciens. Certains logiciens l'ont formalisée depuis longtemps, comme Stanaslaw Jaskowski en 1934 et Frederic Brenton Fitch en 1952, ainsi que le rappelle l'historique de Francis Jeffrey Pelletier [3] .

La règle d'élimination de la disjonction, qui est utilisé dans notre livre [1] et dans le logiciel de preuve DN

$$\frac{A \vee B \quad A \Rightarrow C \quad B \Rightarrow C}{C} \vee E$$

est d'ailleurs celle qui est utilisée dans le style Fitch.

Le second est Roy Dickhoff, qui fut professeur à l'université de St Andrews en Écosse. J'ai rencontré Roy, il y a de nombreuses années au CIRM, Centre Iternational de Rencontre Mathématiques à Marseilles. Il m'a montré les nombreux logiciels de preuves qu'il avait écrit en Prolog pour les ordinateurs Apple et cela m'a donné le goût de suivre son travail et de rédiger à mon tour des démonstrateurs. Mais nous n'avons pas fait que travailler ensemble. Nous sommes allés nous baigner dans les calanques car Roy, en plus d'être un grand logicien, était un bon marcheur et nageur. Il est décédé trop tôt, en août 2018. C'est lui qui a écrit les bases du prouveur intuitioniste dans l'article [2]. C'est ce prouveur (sans contraction) qui est implémenté par la fonction preuve du module preuve.ml. J'ai cru comprendre que la tactique tauto de l'assistant de preuves coq est inspiré du même article.

1 Preuve avec assume et therefore

Nous présentons ce qu'est une preuve avec des explications reprenant en partie celles du livre [1], mais qui permettent aussi de mieux comprendre mon logiciel produisant et vérifiant les preuves.

Les formules ont la syntaxe usuelle indiquée dans ce livre et sur le site `http://teachinglogic.liglab.fr/DN/syntax.php`. On rappelle simplement que la disjonction est notée +, la conjonction &, et la négation par -.

Dans la définition des preuves que nous présentons, nous utilisons, pour les listes et les couples, les notations du langage Ocaml.

Une preuve est une liste de lignes de l'une des formes

assume formule
formule
therefore formule

Pour vérifier qu'une liste de lignes est bien une preuve, on construit pour chaque ligne i de la preuve une liste de listes llu_i de couples (j,a) où a est une formule figurant sur la ligne j. Cette liste est construite de sorte que les formules utilisables à la ligne i pour en dériver des formules, sont les formules a telles que le couple (j,a) est élément d'une des listes de llu_i.

Pour faciliter la vérification de la correction d'une preuve, on définit

— la liste lfu_i comme la liste des couples (j,a) élément d'une des listes de llu_i. Ainsi la formule a est utilisable sur la ligne i si et seulement si il y a un entier j tel que (j,a), où a, la formule de la ligne j, est élément d'une liste de llu_i ou élément de la liste lfu_i.

— la liste $cont_i$ comme la liste des couples (j,a) dernier élément d'une des listes de llu_i.

— la liste $contexte_i$ comme la liste obtenue en remplaçant chaque couple (i,a) de la liste $cont_i$ par la formule a.
Le contexte de la ligne i est la liste des hypothèses utilisables sur cette ligne, et ce contexte vérifie la propriété 1.

Une preuve de la formule a est une liste de lignes de l'une des formes, assume b, therefore b, b où b est une formule, telle que, n étant le nombre de lignes de la preuve, nous avons

1. $llu_0 = [[]]$ et $llu_n = [[n,a]]$, autrement dit au début de la preuve, aucune formule n'est utilisable et à la fin de la preuve, la seule formule utilisable sur la dernière ligne de la preuve, est a, la formule prouvée.
   $cont_0 = []$, le contexte initial est vide.
2. Si la ligne i est (assume b) alors $llu_i = [(i,b)] :: llu_{i-1}$. La nouvelle liste ajoutée comporte uniquement la nouvelle hypothèse.
   Nous avons $cont_i = (i,b) :: cont_{i-1}$. Il en résulte que $contexte_i = b :: contexte_{i-1}$.
3. Si la ligne i est (therefore b) alors c'est une utilisation de la règle d'introduction de l'implication
   — la liste llu_{i-1} doit comporter *au moins deux éléments*, autrement dit s'écrire $u_1 :: u_2 :: finllu$ et $llu_i = [i,b :: u_2] :: finllu$
   — la formule b doit être une implication $c => d$ où c est la dernière hypothèse ajoutée, autrement dit il existe j tel que (j,c) est le dernier élément de u_1 (qui est la première liste de llu_{i-1}) et d est une formule utilisable sur la ligne $i-1$, autrement il doit y avoir un couple (k,d) élément de la liste lfu_{i-1}.
   — la liste $cont_i$ est obtenue en enlevant le premier élément de $cont_{i-1}$. Il en résulte de même que la liste $contexte_i$ est obtenue en enlevant le premier élément de $cont_{i-1}$.
   Notons que la justification de la ligne i est alors (=>I j,k)
4. Si la ligne i est seulement la formule (b) alors cette formule doit être la conclusion d'une règle de la déduction naturelle, dont les prémisses sont des formules *utilisables* sur la ligne $i-1$, autrement dit des formules c telles qu'il existe j pour lequel le couple (j,c) est un élément de lfu_{i-1}.
   On utilise la variante ci-dessous de l'élimination de la disjonction, dont les prémisses sont des *formules* :

$$\frac{a+b \quad a => c \quad b => c}{c} \; +ES$$

Propriété 1 *Soit une preuve d'une formule a, conforme à la définition ci-dessus. Ainsi qu'il est prouvé dans notre livre [1], soit* $contexte_i$ *la liste des hypothèses de la ligne i de la preuve et soit b la formule de cette ligne. Nous avons* $contexte_i \vdash b$.

Dans la preuve de la formule a, cette formule est la dernière ligne de la preuve et le contexte est alors vide, donc $\vdash a$.

Exemple 2 *On présente l'exemple de la preuve de la formule* $a+b => b+a$.

numéro	*preuve*	*llu*	*règle*
1	*assume* $a+b$	*[[1,*$a+b$*];[]]*	
2	*assume a*	*[[2,a];[1,*$a+b$*];[]]*	
3	$b+a$	*[[3,*$b+a$*;2,a];[1,*$a+b$*];[]]*	*+I2 2*
4	*therefore* $a => b+a$	*[[4,*$a => b+a$*;1,*$a+b$*];[]]*	*=>I 2,3*
5	*assume b*	*[[5,b];[4,*$a => b+a$*;1,*$a+b$*];[]]*	
6	$b+a$	*[[6,*$b+a$*;5,b];[4,*$a => b+a$*;1,*$a+b$*];[]]*	*+I1 5*
7	*therefore* $b => b+a$	*[[7,*$b => b+a$*;4,*$a => b+a$*;1,*$a+b$*];[]]*	*=>I 5,6*
8	$b+a$	*[[8,*$b+a$*;7,*$b => b+a$*;4,*$a => b+a$*;1,*$a+b$*];[]]*	*+ES 1,4,7*
9	*therefore* $a+b => b+a$	*[[9,*$a+b => b+a$*]]*	

On rappelle que les formules utilisables sur la ligne i sont les formules c telles qu'il existe un entier j *où* (j,c) *est un élément des éléments de la liste llu de la ligne i. Ainsi sur la ligne 8, sont utilisables pour en déduire* $b+a$*, les formules des lignes 1,4 et 7.*

On présente la même preuve en indiquant le contexte de chaque ligne de la preuve.

Ainsi que l'indique la propriété 1, pour toute ligne de contexte Γ *et de formule a, on a :* $\Gamma \vdash a$*. Par exemple, sur la ligne 6, nous avons* $[b; a+b] \vdash b+a$.

numéro	*preuve*	*contexte*	*règle*
1	*assume* $a+b$	*[*$a+b$*]*	
2	*assume a*	*[a;*$a+b$*]*	
3	$b+a$	*[a;*$a+b$*]*	*+I2 2*
4	*therefore* $a => b+a$	*[*$a+b$*]*	*=>I 2,3*
5	*assume b*	*[b;*$a+b$*]*	
6	$b+a$	*[b;*$a+b$*]*	*+I1 5*
7	*therefore* $b => b+a$	*[*$a+b$*]*	*=>I 5,6*
8	$b+a$	*[*$a+b$*]*	*+ES 1,4,7*
9	*therefore* $a+b => b+a$	*[]*	

2 Preuve avec assume, therefore et end

Nous rappelons la règle usuelle de la disjonction :

$$\frac{a+b \quad a \vdash c \quad b \vdash c}{c} +E$$

Cette règle a comme prémisses la formule $a+b$ et les deux *conditions* $a \vdash c$ et $b \vdash c$ qui, elles aussi, doivent être utilisables au moment où cette règle est appliquée. Pour cela il y a deux changements dans les preuves.

On ajoute la nouvelle ligne (end b) où b est une formule, qui a pour effet d'enlever la dernière hypothèse, disons c, et d'ajouter à l'hypothèse précédente la condition $c \vdash b$.

Ce qui était llu_i, la liste des listes dont les éléments étaient des couples (j,b) où j est la ligne où figure la formule b , devient maintenant une liste des listes dont les éléments sont des couples (j,b), où b est la formule de la ligne j, qui est utilisable en ligne i, et des quadruplets (j,b,k,c), où b est la formule de la ligne j, c est la formule de la ligne k et la condition $b \vdash c$ est utilisable sur la ligne i.

Remarque 3 *Puisque notre programme est écrit dans le langage Ocaml et que dans ce langage, les élément d'une liste sont tous de même type, les éléments d'une liste llu_i sont, dans notre programme, du type fc (formule et condition) où :*

*type fc = | F of int*formula | C of int*formula*int*formula.*

Mais dans notre description informelle des listes llu_i, nous admettrons que les listes peuvent avoir comme élément des couples et des quadruplets.

L'effet des lignes assume b, therefore b où b est une formule, est celui indiqué au paragraphe précédent.

Si la ligne i est (end b) alors

— la liste llu_{i-1} doit comporter *au moins deux éléments*, donc s'écrire $u_1 :: u_2 :: finllu$ et nous avons $llu_i = [(j,c,i,b) :: u_2] :: finllu$ où j,c est le dernier élément de la liste u_1, autrement dit c est l'hypothèse enlevée par la ligne end.

— La liste $cont_i$ est obtenue en enlevant le premier élément de la liste $cont_{i-1}$ et de même la liste de formule $contexte_i$ est obtenue en enlevant le premier élément de la liste $contexte_{i-1}$

— la formule b doit être la conclusion d'une règle de la déduction naturelle des formules ou des conditions utilisables sur la ligne $i-1$.
Parmi ces règles, l'élimination de la disjonction est effectuée par la règle $+E$ que nous avons rappelé ci-dessus.

De la liste llu_i, la liste des listes dont les éléments sont des couples (j,a) et des quadruplets (j,a,k,b), on extrait deux listes lfu_i et lcu_i ainsi définies :

— lfu_i est la liste des couples (j,b) éléments d'une des listes de llu_i, autrement dit lfu_i est la liste des couples (j,b), où b est la formule de la ligne j et b est utilisable sur la ligne i.

— lcu_i est la liste des quadruplets (j,b,k,c) éléments d'une des listes de llu_i, autrement dit lcu_i est la liste des quadruplets (j,b,k,c) où b est la formule de la ligne j, c est la formule de la ligne k et $b \vdash c$ est une condition utilisable sur la ligne i.

Si la ligne i est seulement la formule (b) alors cette formule doit être la conclusion d'une règle de la déduction naturelle, dont les prémisses sont des formules ou des conditions *utilisables* sur la ligne $i-1$.

Pour pouvoir appliquer la règle $+E$, rappelée ci-dessus, à la ligne i, il faut que la formule $b+c$ soit utilisable sur la ligne $i-1$, donc qu'il existe j tel que $(j,b+c)$ soit un élément de lfu_{i-1} et que les conditions $b \vdash d$ et $c \vdash d$ soient utilisables sur la ligne $i-1$, donc qu'il existe des entiers k,l,m,n tels que (k,b,l,d) et (m,c,n,d) soient éléments de la liste lcu_{i-1}.

La justification de cette règle appliquée à la ligne i est alors (+E j, k-l, m-n)

Exemple 4 *On présente la preuve de la formule $a+b => b+a$ avec l'élimination de la disjonction par la règle $+E$.*

numéro	*preuve*	*llu*	*règle*
1	*assume* $a+b$	*[[1,*$a+b$*];[]]*	
2	*assume* a	*[[2,*a*];[1,*$a+b$*];[]]*	
3	*end* $b+a$	*[[(2,*a*,3,*$b+a$*);1,*$a+b$*];[]]*	*+I2 2*
4	*assume* b	*[[4,*b*];[(2,*a*,3,*$b+a$*);1,*$a+b$*];[]]*	
5	*end* $b+a$	*[[(4,*b*,5,*$b+a$*);(2,*a*,3,*$a+b$*);1,*$a+b$*];[]]*	*+I1 4*
6	$b+a$	*[[6,*$b+a$*;(4,*b*,5,*$b+a$*);(2,*a*,3,*$a+b$*);1,*$a+b$*];[]]*	*+E 1,2-3,4-5*
7	*therefore* $a+b => b+a$	*[[7,*$a+b => b+a$*]]*	

On présente la même preuve mais pour chaque ligne i de la preuve, on écrit les listes lfu_i *des formules utilisables et* lcu_i *des conditions utilisables en ligne i extraites des listes* llu_i*.*

numéro	*preuve*	*lfu*	*lcu*	*règle*
1	*assume* $a+b$	*[1,*$a+b$*]*	*[]*	
2	*assume* a	*[2,*a*;1,*$a+b$*]*	*[]*	
3	*end* $b+a$	*[2,*a*;1,*$a+b$*]*	*[(2,*a*,3,*$b+a$*)]*	*+I2 2*
4	*assume* b	*[4,*b*; 1,*$a+b$*]*	*[(2,*a*,3,*$b+a$*)]*	
5	*end* $b+a$	*[1,*$a+b$*]*	*[(4,*b*,5,*$b+a$*);(2,*a*,3,*$b+a$*)]*	*+I1 4*
6	$b+a$	*[6,*$b+a$*;1,*$a+b$*]*	*[(4,*b*,5,*$b+a$*);(2,*a*,3,*$b+a$*)]*	*+E 1,2-3,4-5*
7	*therefore* $a+b => b+a$	*[7,*$a+b => b+a$*]*		*=>I 1,6*

À cause de la ligne end, la propriété 1 doit être modifiée

Propriété 5 *Soit une preuve d'une formule conforme à la définition de ce paragraphe. Soit* $contexte_i$ *la liste des hypothèses de la ligne i.*

- *Si la ligne i est une des lignes, (assume b), (therefore b), b où b est une formule alors* $contexte_i \vdash b$
- *Si la ligne i est la ligne (end b) alors* $contexte_{i-1} \vdash b$

La preuve de cette propriété est laissée au lecteur.

Exemple 6 *Nous présentons une nouvelle fois, la preuve de la formule* $a+b => b+a$ *avec le contexte de chaque ligne et la conséquence de ce contexte. On constate sur cet exemple, que la propriété 5 est vérifiée.*

numéro	*preuve*	*contexte*	*conséquence*	*règle*
1	*assume* $a+b$	*[*$a+b$*]*	$[a+b] \vdash a+b$	
2	*assume* a	*[*a*;*$a+b$*]*	$[a;a+b] \vdash a$	
3	*end* $b+a$	*[*$a+b$*]*	$[a;a+b] \vdash b+a$	*+I2 2*
4	*assume* b	*[*b*;*$a+b$*]*	$[b;a+b \vdash b$	
5	*end* $b+a$	*[*$a+b$*]*	$[b;a+b] \vdash b+a$	*+I1 4*
6	$b+a$	*[*$a+b$*]*	$[a+b] \vdash b+a$	*+E 1,2-3,4-5*
7	*therefore* $a+b => b+a$	*[7]*	$[] \vdash a+b => b+a$	*=>I 1,6*

3 Les règles de la négation et de l'équivalence

Comme dans notre livre [1], les formules et les preuves sont écrites avec *toutes* les connectives usuelles mais une preuve est correcte si et seulement la preuve obtenue en en remplaçant (-a) par (a => F) et les (a <=> b) par ((a =>b)& (b => a)) est correcte avec les règles de la déduction naturelle sans les règles pour la négation et l'équivalence

Dans le programme `http://teachinglogic.liglab.fr/DN/`, c'est cette méthode qui est adoptée. Ainsi pour éliminer l'équivalence, on utilise la règle à gauche en omettant la règle de copie, qui est implicite. Sur la droite du tableau, on a explicité cette copie.

règle	règle avec copie
$\dfrac{i,(a <=> b)}{(a => b)}$ $\&E1\ i$	$\dfrac{\dfrac{i,(a <=> b)}{i,(a => b)\&(b => a)}\ Copie\ i}{(a => b)}\ \&E1\ i$

L'avantage de cette méthode est de diminuer le nombre de règles qu'il faut programmer et de raccourcir les preuves. L'inconvénient est la difficulté de lire les preuves : il faut identifier une formule et la formule obtenue en remplaçant (-a) par (a => F) et (a <=> b) par ((a=>b) & (b =>a).

Exemple 7 *Considérons la preuve suivante de la formule (a <=> - -a).*

numéro	*preuve*	*contexte*	*règle*
1	*assume a*	*[a]*	
2	*assume -a*	*[-a ;a]*	
3	*F*	*[-a ;a]*	*=>E 2,1*
4	*therefore - -a*	*[a]*	*=>I 2,3*
5	*therefore a => - -a*	*[]*	*=>I 1,4*
6	*assume - -a*	*[- -a]*	
7	*a*	*[- -a]*	*Raa 6*
8	*therefore - -a => a*	*[]*	*=>I 6,7*
9	*a <=> - -a*	*[]*	*&I 4,8*

On voit ci-dessous que les règles sont appliquées aux formules obtenues en éliminant la négation et l'équivalence. Par exemple à la ligne 3, on déduit F des lignes 2 et 1, par la règle =>E appliquée aux formules (-a) et (a) car la formule (-a) est, à une règle de copie près, égale à (a => F).

numéro	*preuve*	*contexte*	*règle*
1	*assume a*	*[a]*	
2	*assume a => F*	*[a => F ;a]*	
3	*F*	*[a => F ;a]*	*=>E 2,1*
4	*therefore (a=>F)=>F*	*[a]*	*=>I 2,3*
5	*therefore a => ((a=>F)=>F)*	*[]*	*=>I 1,4*
6	*assume (a =>F)=>F*	*[(a=>F)=>F]*	
7	*a*	*[(a=>F)=>F]*	*Raa 6*
8	*therefore ((a=>F)=>F) => a*	*[]*	*=>I 6,7*
9	*(a => ((a=>F)=>F)) & (((a=>F)=>F) => a)*	*[]*	*&I 4,8*

Pour éviter ces difficultés dans la lecture des preuves, nous avons choisi, pour le programme `http://teachinglogic.liglab.fr/DN2` d'ajouter les règles de la déduction naturelle pour l'équivalence et la négation

introduction	élimination <=>E1	élimination <=>E2
$\frac{(a => b) \quad (b => a)}{(a <=> b)} <=> I$	$\frac{(a <=> b)}{(a => b)} <=> E1$	$\frac{(a <=> b)}{(b => a)} <=> E2$
$\frac{(a \vdash F)}{(-a)} -I$	$\frac{(-a) \quad (a)}{b} -E$	

La règle d'introduction de la négation est traitée avec la ligne therefore. La ligne therefore -a est correcte si, en cette ligne, la formule a (de la ligne i) est la dernière hypothèse active (le premier élément du contexte) et si F (de la ligne j) est utilisable. Dans ce cas l'hypothèse a est enlevée, et la justification de la règle est -I i,j.

4 Les programmes DN, DN1, DN2

Nous avons trois versions du prouveur et vérificateur de preuves. Dans les trois versions, appelées DN, DN1 et DN2, une preuve liste est, en Ocaml, une liste de type *line list* et, informellement, une preuve est une liste de lignes qui sont des formules ou des formules précédées des mots assume, therefore ou end.

Dans DN, comme dans DN1, il n'y a pas de règles pour la négation et l'équivalence, la formule $-A$ est une abréviation de $A \Rightarrow F$ et la formule $A \Leftrightarrow B$ est une abréviation de $(A \Rightarrow B)\&(B \Rightarrow A)$. À cause de la simplicité de la structure des preuves et parce notre livre [1] utilise les mêmes règles et les mêmes abréviations, *nous recommandons cette version pour un enseignement de la déduction naturelle*.

4.1 Le programme DN

Le programme Ocaml est divisé en trois modules

1. Le module Interface. Ce module contient la définition des structures utilisées, dont les structures formula, line, preuve_arbre ainsi qu'une fonction string_of_formula pour transformer une formule en chaîne.
2. Le module Assistant. Ce module est celui du programme principal assistant. Ce programme est appelé ainsi : assistant option fichier.

(a) avec l'option -p, le fichier doit comporter une formule suivie d'un point. Le programme assistant lit la formule du fichier , fait la preuve de cette formule et affiche cette preuve sans justification.

(b) avec les options -a,-b,-v le fichier doit comporter une formule suivie d'une preuve. Le programme lit la formule et la preuve.

i. avec l'option -a, le programme affiche la preuve identée avec la justification des règles utilisées

ii. avec l'option -b, le programme affiche la preuve avec la justification des règles utilisées mais sans indenter la preuve, ce qui convient pour lire une preuve sur un petit écran.

iii. avec l'option -v, le programme vérifie seulement la correction de la preuve sans l'afficher

3. Le module Preuve. Ce module comporte les principales fonctions suivantes

(a) la fonction preuve recherche une preuve intuitioniste d'une formule en suivant la méthode de Roy Dickoff [2]

(b) la fonction preuvec recherche une preuve classique d'une formule en suivant la méthode de notre livre [1]

Avec l'option -a, le programme assistant exécute l'expression (dnpreuve f), où f est le fichier comportant la formule à prouver. Cette fonction recherche d'abord une preuve *intuitioniste* en exécutant l'expression (preuve a) où a est la formule à prouver. Si la formule à prouver n'a pas de preuve intuitioniste, le programme recherche une preuve classique. Soit p la preuve obtenue de la formule a. Avant d'afficher une preuve de a, la fonction dnpreuve effectue divers traitements pour enlever de la preuve p toutes les lignes répétées ou inutiles de la preuve.

4.1.1 Eviter la répétition des preuves

Considérons la preuve suivante de la formule (p => p + q) & (p => p+q) produite par le programme DN1, *qui ne comporte pas ce traitement* consistant à supprimer les preuves inutilement répétées

numéro	preuve	règle
1	assume p	
2	p+q	+I1 1
3	therefore p => p+q	=>I 1, 2
4	assume p	
5	p+q	+I1 4
6	therefore p => p+q	=>I 4, 5
7	(p => p+q) & (p => p+q)	&I 3,6

La formule (p => p+q) ayant été prouvée à la ligne 3, il est inutile de faire une autre preuve de cette même formule. Donc il faut enlever les lignes 4,5,6 pour obtenir la preuve plus courte suivante :

numéro	preuve	règle
1	assume p	
2	p+q	+I1 1
3	therefore p => p+q	=>I 1, 2
7	(p => p+q) & (p => p+q)	&I 3,3

Cette suppression des preuves répétées est obtenue ainsi dans la fonction dnpreuve. La preuve p de type line list est transformée en une preuve pla de type preuve_arbre list. Sur l'exemple ci-dessus, la preuve 4.1.1 est transformée en la preuve, qui est une liste d'éléments de type preuve_arbre.

[Bloc(p,p+q,p=>p+q) ;Bloc(p,p+q,p+q) ;(p=>p+q)&(p=>p+q)]

C'est la fonction preuve_arbre_liste_de_preuve_ligne_liste (du module Preuve) qui transforme la liste p en la liste pla. Puis la fonction compacter_preuve_arbre (du module Preuve) transforme la liste pla en une liste pc de type line list, tout en utilisant la structure des blocs, pour supprimer les répétitions de preuves de la même formule.

4.1.2 Supprimer les lignes inutiles

Une preuve d'une formule a peut comporter des lignes inutiles, qui ne sont pas utilisées dans la preuve de a. Pour trouver ces lignes inutiles, la fonction dnpreuve transforme la preuve pc de type line list en un tableau tpc avec les justifications de chaque ligne. Cette transformation est effectuée par la fonction preuve_tableau_de_preuve_liste (du module Assistant). Grâce à ces justifications, la fonction preuve_liste_de_preuve_tableau peut marquer les lignes utilisées dans la preuve de a, puis retransformer le tableau en une liste ne comportant que les lignes utiles dans la preuve de a.

Exemple 8 *Considerons la preuve suivante de la formule (p & q => p), comportant les quatre lignes :*

assume (p & q)
p
q
therefore p & q => p

Cette preuve est transformée par la fonction preuve_tableau_de_preuve_liste en le tableau ci-dessous :

numéro	*preuve*	*régle*
1	*assume (p & q)*	
2	*p*	*&E1 1*
3	*q*	*&E2 1*
4	*therefore (p & q) => p*	*=>I 1, 2*

La fonction preuve_liste_de_preuve_tableau marque les ancêtres de la ligne 4. Ce marquage montre que la ligne 3 est inutile, et cette fonction transforme le tableau en la preuve suivant qui n'a plus que trois lignes

assume (p & q)
p
therefore p & q => p

4.1.3 Calcul des justifications des règles

Comme nous l'avons déjà indiqué, le calcul de la justification des règles appliquées est effectué par la fonction preuve_tableau_de_preuve_liste du module Assistant. Cette fonction suit les principes indiquées dans la section 1. La liste llu_i, liste des listes des formules, accompagnées de leur ligne, *utilisables* à la ligne i est dans cette fonction représentée par la référence lu. Le contexte de chaque ligne, notée dans cette même section par $cont_i$ est dans le programme Ocaml représenté par la référence lh.

4.2 Le programme DN1

Nous ne notons que les différences avec le programme DN dans le calcul de la justification des règles. Ainsi qu'il a été remarqué dans la section 2, il n'y a pas que les formules qui sont utilisables. Il y a aussi les conditions $a \vdash b$ utilisées dans l'application de la règle d'élimination de la disjonction. La liste des listes llu_i est une liste des listes d'éléments de type fc, formule et condition. La liste llu_i est représentée en Ocaml par la référence llu, la liste lfu_i des formules utilisables à la ligne i est représentée par la variable lfu et la liste lcu_i des conditions utilisables à la ligne i est représentée par la variable lcu. Le contexte $cont_i$ des formules avec l'indication de leur ligne, est représenté par la référence $cont$.

À cause de la structure plus complexe des preuves, due à la présence des lignes end, la fonction dnpreuve n'effectue pas la suppression des preuves répétées mais elle effectue la suppression des lignes inutiles.

4.3 Le programme DN2

Ce programme utilise les mêmes variables et les mêmes structures que DN1. La seule différence, avec le programme DN1, est l'absence des fonctions de dépliage des formules. Il y a des règles pour la négation et l'équivalence, ce qui peut augmenter la longueur des preuves.

4.4 La fonction verifier_preuve

Lors de l'execution du programme assistant avec l'option -a, le programme exécute la fonction verifier_preuve du module Assistant. Cette fonction a comme argument un fichier qui doit comporter une formule et une preuve. Sa fonction essentielle est de calculer les justifications des règles appliquées, grâce à la fonction preuve_tableau_de_preuve_liste (du module Assistant) et d'afficher la preuve avec ses justifications.

Références

[1] Stéphane DEVISMES, Pascal LAFOURCADE et Michel LEVY. *Logique et démonstration automatique*. Ellipses, 2012.

[2] Roy DYCKHOFF. *Contraction-Free Sequent Calculi for Intuitionistic Logic*. URL : https://scholar.google.com/citations?view_op=list_works&hl=fr&hl=fr&user=x5zdy1cAAAAJ (visité le 05/02/2023).

[3] Francis Jeffry PELLETIER. URL : http://www.sfu.ca/~jeffpell/papers/NDHistory.pdf (visité le 25/02/2023).