

Memento du démonstrateur interactif B

Version 3.5

Didier Bert

17 février 2000

1 Généralités

1.1 Structure d'une obligation de preuve dans l'atelier B

Au niveau du démonstrateur interactif, une "obligation de preuve" (OP) est composée de trois ensembles de formules :

- les *hypothèses courantes* **HYP**
- les *hypothèses locales* **LOC**
- le *but* **G**

Les hypothèses courantes sont en fait constituées des hypothèses *contextuelles* et des hypothèses *dérivées*. Les premières sont produites par le générateur d'obligations de preuves; les secondes sont produites à partir des commandes de preuve interactive. Les hypothèses locales proviennent de la traduction des substitutions gardées dans l'OP initiale, ou, en cours de preuve interactive, apparaissent lorsqu'un nouveau sous-but est une implication (voir les explications des commandes). Dans la suite, on visualise donc une preuve par :

$$\mathbf{HYP} \vdash \mathbf{LOC} \Rightarrow \mathbf{G}$$

Dans la partie droite de l'écran du démonstrateur interactif, la liste **HYP** apparaît dans la fenêtre du haut, le but courant **LOC** \Rightarrow **G** est dans la fenêtre centrale, alors que la fenêtre du bas, contient les commandes tapées par l'utilisateur et les réponses du système. Le système répète souvent le but courant dans la fenêtre du bas.

1.2 Types de commandes du démonstrateur interactif

On trouve cinq types de commandes dans le démonstrateur interactif (les numéros sont les numéros des paragraphes) :

- (2) calculs sur les hypothèses ou le but pour construire une preuve;
- (3) recherche d'information pour mener la preuve;
- (4) navigation dans les preuves et dans l'espace des OP;
- (5) sauvegarde des scripts de preuve;
- (6) modification des théories, validation de lemmes.

2 Construction des preuves

Il s'agit de commandes qui permettent de construire une preuve interactivement. Elles font appel à la créativité de l'utilisateur et à ses connaissances en matière de mécanismes de preuve. Pour plus de détails, voir [Atelier B, Prouveur interactif, Manuel Utilisateur et Manuel de Référence, version 3.5]

2.1 Appels des démonstrateurs

2.1.1 Appel du prouveur principal (pr)

Le prouveur principal de l'Atelier **B** est basé sur l'inférence de règles. Il cherche s'il existe un *enchaînement* de règles qui élimine tous les buts. Un but est éliminé (déchargé) s'il se trouve déjà en hypothèse. La recherche des règles de cet enchaînement peut se faire par des tactiques "Backward" (en arrière, c'est-à-dire du but vers les hypothèses) ou "Forward" (sens inverse). En mode interactif, on peut lancer le "cœur du prouveur" sur un but. C'est même la première chose à faire dès que le but a été modifié par une commande manuelle (quitte à revenir en arrière, si cette commande ne donne rien de bon).

Formes de la commande (les mots-clés sont en gras, les métavariabes sont en italiques) :

pr	Appel du cœur du prouveur
pr(Red)	Appel du prouveur réduit (pas de preuve par cas)
pr(Tac(<i>t</i>))	Appel du prouveur avec tactique <i>t</i>
pr(<i>r.b.h, f, s</i>)	Appel de prouveur avec traces
pr(Tac(<i>t</i>), <i>r.b.h, f, s</i>)	Appel avec tactique et traces
mp	MiniProof: même chose que pr(Red)

Une tactique fait référence à des "théories" qui font partie de la bibliothèque prédéfinie ou à des tactiques utilisateurs. Une tactique *t* est de la forme *B, F* où *B* est une tactique "Backward" et *F* une tactique "Forward". Pour plus de détails sur les tactiques, voir le manuel de référence, page 69. Les métavariabes de l'appel avec trace signifient :

<i>r</i>	None Ru	affichage du nom des règles affichage du nom et du corps des règles
<i>b</i>	Goal Stop	affichage des buts affichage et arrêt sur chaque but
<i>h</i>	None Gen Full	aucun affichage des hypothèses affichage des hypothèses générées affichage des hypothèses montées dans la pile
<i>f</i>	File NoFile	génération du fichier des traces visualisable par la commande Show Proof Tree pas de génération (option par défaut)
<i>s</i>	Simpl NoSimpl	affichage des simplifications pas d'affichage des simplifications (option par défaut)

Si le prouveur n'a pu rien pu faire, il affiche le message :

Prover call has not done anything

Dans ce cas, il faut tenter autre chose. Sinon, cela peut vouloir dire que la commande n'a pas abouti par manque de temps. On peut donc recommencer à appeler "**pr...**" pour aller plus loin dans la preuve.

La commande de **ff** (Force) permet de changer la force du prouveur et de relancer la ligne de commande courante avec la nouvelle force.

ff (n)	Reprise de la preuve avec la force n
-------------------	--

Le paramètre n peut être : **Rapide**, 0, 1, 2 ou 3.

2.1.2 Appel du prouveur de prédicats (**pp**)

Le prouveur de prédicats fonctionne suivant le mode de la "résolution". Il est bien adapté à la preuve dans la théorie des ensembles, mais pas encore pour les preuves arithmétiques. De plus, il n'est pas efficace s'il y a beaucoup d'hypothèses. La façon habituelle de l'utiliser est de réduire le nombre des hypothèses qu'il devra prendre en compte pour la preuve. Formes d'utilisation :

pp	Appel du prouveur de prédicats (si peu d'hypothèses)
pp (rp.i)	Appel du prouveur avec hypothèses réduites
pp (t)	Appel du prouveur limité à la durée t (en secondes)
pp (rp.i t)	combinaison des deux cas précédents

Lorsqu'il n'y a pas de durée indiquée, le prouveur est arrêté automatiquement au bout de 60s. La commande **rp** signifie :

rp.0	but sans hypothèses
rp.1	but et hypothèses qui ont une variable libre en commun avec le but
rp.2	itère le calcul par rapport à l'ensemble rp.1
etc.	

La commande **rp** utilisée seule affiche l'obligation de preuve courante avec hypothèses réduites (cf. section 3.4).

2.1.3 Appel du prouveur arithmétique (**ap**)

Le prouveur arithmétique recherche les preuves par contradiction dans des ensembles d'égalités et de relations d'ordre entre valeurs numériques. Il est bien adapté s'il y a beaucoup d'inégalités dans les hypothèses et pour résoudre des buts de la forme : $a \leq b$ ou $a = b$.

ap	Appel du prouveur arithmétique
ap (n)	Appel du prouveur arithmétique avec limitation n

Par défaut, la limitation est $n = 400$.

2.1.4 Simplificateur d'expressions ensemblistes

La commande **ss** (simplify set) permet de simplifier un but composé d'expressions d'ensembles.

ss	Appel du simplificateur d'ensembles sur le but
-----------	--

La commande appelle plusieurs outils. Il y a un simplificateur d'expressions littérales qui traite les expressions d'ensemble, les intervalles et les valeurs booléennes. Le simplificateur d'expressions quelconques ne traite que les expressions avec opérateurs ensemblistes

2.1.5 Preuve par tentative

Cette commande **tp** (proof by attempts) essaie de modifier le but courant en fonction des règles applicables :

tp (m)	Essai de preuve suivant le mode m
tp (m, n)	Essai n tentatives de preuve en mode m

Le paramètre n est un entier indiquant le nombre maximum de tentatives (par défaut 20). Le paramètre m peut prendre deux valeurs :

m	Goal	le prouveur génère des hypothèses d'après le but (voir aussi ch)
	Hyp	le prouveur essaie à partir des hypothèses.

Dans sa nouvelle version, cette commande traite des cas de preuve par contradiction (incohérence des hypothèses). Autre possibilité, voir les commandes **ct** et **cts**, paragraphe 2.3.

2.2 Application d'une règle ou d'une tactique

Les règles de la base sont organisées en théories (ensembles de règles qui forment un tout logique et qui peuvent être utilisées avec la même tactique avant ou arrière). L'ensemble des règles (et des théories) du démonstrateur est visible en cliquant sur le bouton "Display Rules Database" du menu "Display/Print" de la fenêtre INTERACTIVE PROOF du démonstrateur interactif. On peut repérer quelles sont les règles potentiellement applicables à une formule en cours de preuve par une commande de recherche du démonstrateur (il s'agit de la commande **sr** au paragraphe 3).

La commande **ar** (Apply Rule) est l'application d'une règle ou d'une théorie, ou encore d'une tactique à l'OP courante. Elle prend une des formes suivantes :

ar (r, M)	Application d'une règle r avec le mode M
ar (T)	Application de la théorie ou de la tactique T

Le mode est de plusieurs formes suivant que la règle est une règle de réécriture ou pas:

Type de règle	M	Signification
règle backward	Once Multi	la règle est appliquée une seule fois (backward) application itérée tant que la règle s'applique
règle forward	Fwd Fwd(P)	application forward: (voir ci-dessous) application forward contrainte par P
règle de réécriture	Goal AllHyp Hyp(h)	réécriture du but réécriture de toutes les hypothèses réécriture des l'hypothèses qui coïncident avec h

Appliquer une règle $r = h_1 \wedge \dots \wedge h_n \Rightarrow g$ en forward sur une OP signifie: pour tout ensemble d'hypothèses $H_1 \wedge \dots \wedge H_n$ de **HYP** qui a la même forme par σ que les antécédents de r^1 , on génère la formule $\sigma(g)$ dans les hypothèses courantes (c'est là qu'il y a un risque de bouclage). Dans le cas où l'on utilise **ar($r, \mathbf{Fwd}(P)$)**, cela signifie que l'une des hypothèses h_i doit coïncider avec P .

Dans les cas de règle de réécriture, les modes peuvent être suivis de “**.Part(f)**” où f est un filtre. La réécriture est alors restreinte aux sous-formules des formules sélectionnées qui satisfont f . Cela est utile lorsqu'il y a plusieurs sous-formules qui peuvent être réécrites par la règle, alors que l'intention est de ne réécrire qu'une seule de ces sous-formules.

Une application de théorie/tactique Backward est de la forme:

$$\mathbf{ar}(\mathbf{tb1};\mathbf{tb2};\dots\mathbf{tbn})$$

Si l'on utilise des théories Backward et Forward, on écrira:

$$\mathbf{ar}((\mathbf{tb1};\mathbf{tb2};\dots\mathbf{tbn};\mathbf{DED}),(\mathbf{tf1};\dots\mathbf{tfm}))$$

Dans ce cas, la théorie **DED** est obligatoire².

Les hypothèses engendrées par les applications Backward de la commande **ar** se retrouvent en hypothèses courantes. Les hypothèses engendrées par les applications Forward sont en hypothèses locales.

2.3 Preuve par contradiction

Il y a deux cas d'utilisation. On peut essayer de montrer que la négation du but est fausse (valide à cause du tiers exclu) ou bien que le but est vrai parce que les hypothèses sont contradictoires.

1. Voir le paragraphe 3.1.

2. Ce point qui est lié à la “remontée des hypothèses” n'est pas parfaitement clair.

2.3.1 Contradiction

Il y a deux formes de commandes :

ct	Contradiction avec montée d'hypothèse
cts	Contradiction spéciale

Pour les commandes qui modifient syntaxiquement la forme d'une OP, on note leur effet par une transformation où la forme avant application est au dessus de la barre et la forme après transformation est au-dessous. L'effet des commandes **ct** et **cts** peut s'exprimer par :

$$\text{ct} \quad \frac{\mathbf{HYP} \vdash \mathbf{G}}{\mathbf{HYP}, \neg \mathbf{G} \vdash \mathbf{bfalse}} \quad \text{cts} \quad \frac{\mathbf{HYP} \vdash \mathbf{G}}{\mathbf{HYP} \vdash \neg \mathbf{G} \Rightarrow \mathbf{bfalse}}$$

On rappelle que **bfalse** est le prédicat constant *Faux* (et **true** le prédicat constant *Vrai*).

2.3.2 Hypothèses contradictoires

La commande **fh** (False Hypothesis) s'écrit :

fh (h)	Hypothèse h contradictoire
-------------------	------------------------------

Dans ce cas, $h \in \mathbf{HYP}$. L'effet de cette commande est de remplacer le but courant par la négation de l'hypothèse contradictoire. Si le nouveau but réussi, alors l'obligation de preuve initiale est démontrée. Plus symboliquement :

$$\mathbf{fh}(h) \quad \frac{\mathbf{H}_i, h, \mathbf{H}_j \vdash \mathbf{G}}{\mathbf{H}_i, h, \mathbf{H}_j \vdash \neg h}$$

2.4 Preuve par cas

La commande "Do Cases" peut prendre trois formes :

dc (h)	Décomposition sur un prédicat
dc (v, E)	Décomposition sur une énumération
dcs ($A \vee B$)	Décomposition spéciale sur une disjonction

La commande **dc** engendre deux sous-buts dans le premier cas et $p + 1$ sous-buts dans le deuxième cas, si l'ensemble E contient p éléments ($p \leq 50$). La commande **dcs** engendre $n + 1$ sous-buts si le prédicat a n disjonctions, Note : lorsqu'on indique qu'un but est de la forme : $\mathbf{HYP} \vdash \mathbf{G} \wedge \mathbf{F}$, cela se traduit dans l'atelier par deux sous-buts : $\mathbf{HYP} \vdash \mathbf{G}$ et $\mathbf{HYP} \vdash \mathbf{F}$. Effet de la commande **dc** :

$$\mathbf{dc}(h) \quad \frac{\mathbf{HYP} \vdash \mathbf{G}}{\mathbf{HYP} \vdash (h \Rightarrow \mathbf{G}) \wedge (\neg h \Rightarrow \mathbf{G})}$$

Pour le deuxième cas, l'ensemble d'énumération peut être énuméré simple, ou produit cartésien de deux formes, d'où trois possibilités d'extensions. Dans les règles, on a $E = \{e_1, \dots, e_n\}$:

$$\mathbf{dc}(v, E) \frac{\mathbf{HYP} \vdash \mathbf{G}}{\mathbf{HYP} \vdash (v \in E) \wedge (v = e_1 \Rightarrow \mathbf{G}) \wedge \dots \wedge (v = e_n \Rightarrow \mathbf{G})}$$

$$\mathbf{dc}(v, E \times \{f\}) \frac{\mathbf{HYP} \vdash \mathbf{G}}{\mathbf{HYP} \vdash (v \in E \times \{f\}) \wedge (v = \{e_1 \mapsto f\} \Rightarrow \mathbf{G}) \wedge \dots \wedge (v = \{e_n \mapsto f\} \Rightarrow \mathbf{G})}$$

$$\mathbf{dc}(v, E \times F) \frac{\mathbf{HYP} \vdash \mathbf{G}}{\mathbf{HYP} \vdash (v \in E \times F) \wedge (v = (\{e_1\} \times F) \Rightarrow \mathbf{G}) \wedge \dots \wedge (v = (\{e_n\} \times F) \Rightarrow \mathbf{G})}$$

Avec la commande de décomposition spéciale **dc**s, on cherche à prouver le but sous les hypothèses où l'un des sous-prédicats de la disjonction est vrai et les autres sont faux. Cela fait donc n cas différents. De plus, il faut que l'hypothèse donnée (la disjonction) soit aussi vérifiée. D'où la règle (pour la disjonction de deux prédicats):

$$\mathbf{dc}s(A \vee B) \frac{\mathbf{HYP} \vdash \mathbf{G}}{\mathbf{HYP} \vdash (A \vee B) \wedge (A \wedge \neg B \Rightarrow \mathbf{G}) \wedge (\neg A \wedge B \Rightarrow \mathbf{G})}$$

On peut noter que le premier cas est un cas particulier de la décomposition spéciale avec **dc**(h) identique à **dc**s($h \vee \neg h$).

2.5 Utilisation explicite de la réécriture

La commande **eh** (use Equality in Hypothesis) s'écrit :

eh (a, A, f)	Utilisation de l'égalité $a = A$ dans f
-------------------------	---

La formule $a = A$ ou $A = a$ doit se trouver dans les hypothèses de l'OP courante. L'expression A peut être remplacée par le mot-clé **_h**, ce qui signifie : la première expression A dans les hypothèses (en partant des dernières) telle que $a = A$ ou $A = a$. La métavariante f est mise pour :

f	Goal	réécriture dans le but
	AllHyp	réécriture dans toutes les hypothèses
	Hyp (h)	réécriture dans l'hypothèse h

2.6 Opérations sur les quantificateurs

2.6.1 Particularisation d'un "pour tout" en hypothèse

Lorsqu'un quantificateur \forall se trouve en hypothèse, on peut le remplacer par une de ces instances. Si la preuve réussit avec une simple instance de cette hypothèse, alors la proposition initiale est démontrée. La commande **ph** (Particularize Hypothesis) a la forme suivante :

ph (v_1, \dots, v_n, h)	Particularise l'hypothèse h avec les valeurs v_1, \dots, v_n
------------------------------------	--

La formule h doit être dans les hypothèses. Elle a la forme :

$$h = \forall w_1, \dots, w_n \cdot (P(w_1, \dots, w_n) \Rightarrow Q(w_1, \dots, w_n))$$

Les valeurs v_1, \dots, v_n (même nombre que le nombre de variables liées) sont soit des expressions, soit le mot-clé **h**. Dans ce cas, la variable correspondante reste une variable quantifiée universellement. La commande a l'effet suivant³ :

$$\mathbf{ph}(v_1, \dots, v_n, h) \frac{\mathbf{H}_i, h, \mathbf{H}_j \vdash \mathbf{G}}{\mathbf{H}_i, h, \mathbf{H}_j \vdash P(v_1, \dots, v_n) \wedge (Q(v_1, \dots, v_n) \Rightarrow \mathbf{G})}$$

2.6.2 Suggestion pour un “il existe” dans le but

La commande **se** (Suggest for Exist) s'applique à un but quantifié existentiellement de la forme “ $\exists w_1, \dots, w_n \cdot (P(w_1, \dots, w_n))$ ”. Elle propose des valeurs v_1, \dots, v_n qui sont des instances valides des variables liées (ou le mot-clé **h**, si l'on ne veut pas instancier la variable à cette position).

se (v_1, \dots, v_n)	Suggestion de preuve d'un but existentiel avec v_1, \dots, v_n
---------------------------------	--

L'effet de la commande est le suivant :

$$\mathbf{se}(v_1, \dots, v_n) \frac{\mathbf{HYP} \vdash \exists w_1, \dots, w_n \cdot (P(w_1, \dots, w_n))}{\mathbf{HYP} \vdash P(v_1, \dots, v_n)}$$

Dans le cas d'un paramètre non lié à la position i :

$$\mathbf{se}(v_1, \dots, \mathbf{h}, \dots, v_n) \frac{\mathbf{HYP} \vdash \exists w_1, \dots, w_n \cdot (P(w_1, \dots, w_n))}{\mathbf{HYP} \vdash \exists w_i \cdot (P(v_1, \dots, w_i, \dots, v_n))}$$

2.7 Opérations sur les hypothèses

2.7.1 Déduction directe

La commande **dd** (Direct Deduction) permet de faire passer les hypothèses locales dans les hypothèses courantes, sans passer par **pr** qui normalise les formules. Le paramètre entier ($i = 0, 1, 2$ ou 3) indique la force du traitement⁴.

dd	Déduction par “remontée” des hypothèses
dd (i)	Déduction avec une certaine force des traitements

Habituellement, on utilise cette commande après **ah**. Du point de vue forme de la preuve, on a :

$$\mathbf{dd} \frac{\mathbf{HYP} \vdash \mathbf{LOC} \Rightarrow \mathbf{G}}{\mathbf{HYP}, \mathbf{LOC} \vdash \mathbf{G}}$$

3. S'il y a des variables qui restent quantifiées, l'effet n'est pas spécifié dans les documents.

4. Ce n'est pas tellement plus expliqué que ça.

2.7.2 Création d'hypothèses

Cette commande **ch** (Create Hypothesis) construit des hypothèses qui sont déduites de la forme du but. L'explicitation de ces hypothèses permet quelquefois au prouveur d'avancer à partir des règles de la base.

ch	Création d'hypothèses d'après la forme du but
-----------	---

2.7.3 Modus ponens sur hypothèse

Cette commande **mh** (Modus ponens on Hypothesis) permet de traiter les hypothèses de la forme $P \Rightarrow Q$ et P dans les hypothèses (sans passer par le prouveur qui simplifierait Q).

mh (h)	Modus ponens sur l'hypothèse h
-------------------	----------------------------------

Transformation de la preuve par la commande :

$$\mathbf{mh}(P \Rightarrow Q) \quad \frac{\mathbf{H}_i, P \Rightarrow Q, \mathbf{H}_j, P, \mathbf{H}_k \vdash \mathbf{G}}{\mathbf{H}_i, P \Rightarrow Q, \mathbf{H}_j, P, \mathbf{H}_k \vdash Q \Rightarrow \mathbf{G}}$$

2.7.4 Ajout d'hypothèses

La commande **ah** (Add Hypothesis) permet d'ajouter une hypothèse si elle est déductible des hypothèses courantes :

ah (h)	Ajout de l'hypothèse h
-------------------	--------------------------

Il y a deux effets de la commande suivant que h est déjà en hypothèse ou non. Si h n'est pas en hypothèse, l'effet de la commande est :

$$\mathbf{ah}(h) \quad \frac{\mathbf{HYP} \vdash \mathbf{G}, \quad h \notin \mathbf{HYP}}{\mathbf{HYP} \vdash h \wedge (h \Rightarrow \mathbf{G})}$$

Si h est en hypothèse, seul le deuxième sous-but est produit. Cela a l'effet de faire "descendre" une hypothèse dans le but (comparer avec **dd**).

$$\mathbf{ah}(h) \quad \frac{\mathbf{H}_i, h, \mathbf{H}_j \vdash \mathbf{G}}{\mathbf{H}_i, h, \mathbf{H}_j \vdash h \Rightarrow \mathbf{G}}$$

À l'usage, cette commande est très utile, car elle permet d'ajouter graduellement des lemmes prouvés dans les hypothèses en vue de construire des étapes intermédiaires pour attendre le but ou pour le réécrire (si l'hypothèse ajoutée est une égalité). L'idée du lemme à ajouter peut venir de l'examen de la base de règles du démonstrateur. Les règles qui n'ont pas été appliquées automatiquement (en particulier les réécritures) peuvent quelquefois être démontrées comme des lemmes et ensuite utilisées explicitement avec **eh**. Lorsqu'on ajoute une hypothèse, la suite habituelle des commandes est de la forme :

Commande	But courant	
	P	
ah (h)	h	
pr ou pp (...)	$h \Rightarrow P$	Si la preuve de h a réussi
dd	P	h est dans les hypothèses. Ici, il peut y avoir eh pour une réécriture de P à l'aide de h , ou pr pour retenter de démontrer P .

3 Recherche d'information

3.1 Principes de la recherche à partir de filtres

Une recherche de règle ou plus généralement de formules se fait à l'aide de la notion de *filtre*. En **B**, le filtre le plus général est le *joker* qui est une variable composée d'une seule lettre (majuscule ou minuscule). Un joker filtre n'importe quelle formule.

Plus formellement, on dit que : une formule F "est filtrée par" G , ou F "a la même forme que" G , ou encore F "coïncide avec" G , ou enfin F "est un modèle de" G , s'il existe une substitution σ des jokers de G telle que $F = \sigma(G)$. Un filtre est donc une formule générale dont la formule filtrée est une instance. Si f est un filtre, on peut définir les *modèles* de f par :

$$\mathcal{M}(f) = \{F \mid F = \sigma(f)\}$$

On a également besoin de définir des filtrages de sous-formules. Pour cela, on définit les *positions* dans une formule, qui sont des indices attachés de manière biunivoque aux nœuds de l'arbre abstrait de la formule. Pour une formule G , $pos(G)$ est l'ensemble de ses positions. D'autre part, si $p \in pos(G)$ alors on note $G|_p$ la sous-formule de G en position p , et $G[p \mapsto F]$ est le remplacement de la sous-formule $G|_p$ par F dans G . L'ensemble des sous-formules d'une formule F est donné par :

$$\mathcal{F}(F) = \{F|_p \mid p \in pos(F)\}$$

L'AtelierB propose un langage des filtres qui permet une recherche fine. Si f et g sont des filtres, alors les expressions suivantes sont des filtres :

Filtre	Modèles
f _and g	$\mathcal{M}(f) \cap \mathcal{M}(g)$
f _or g	$\mathcal{M}(f) \cup \mathcal{M}(g)$
_not (f)	$\overline{\mathcal{M}(f)}$

Notons qu'on ne peut pas utiliser les opérateurs de filtres sous un **_not**. Il semble également que les jokers de chaque sous-formule soient indépendants, ce qui justifie la définition des modèles (à vérifier). Les filtres *absolus* s'appliquent sur la formule entière, alors que les filtres *relatifs* s'appliquent sur les sous-formules.

3.2 Recherche d'hypothèses dans l'OP courante

La commande **sh** (Search Hypothesis) permet de chercher des hypothèses d'une OP qui sont modèles d'un certain filtre. La forme de la commande est :

sh (P, A)	Recherche avec filtre relatif et filtre absolu
sh (P)	Recherche avec filtre relatif seul

Dans la commande, P représente un filtre relatif et A un filtre absolu. La forme simplifiée est équivalente à “**sh**(P, a)” où a est un joker. La commande **sh** sélectionne les hypothèses H d'une OP : $HYP \vdash G$ définies par :

$$H = \{ h \mid h \in HYP \wedge \mathcal{F}(h) \cap \mathcal{M}(P) \neq \emptyset \wedge h \in \mathcal{M}(A) \}$$

3.3 Recherche de règles dans la base de règles du prouveur

Les règles en bibliothèque sont groupées dans des théories. Elles ont la forme $Ant \Rightarrow Cons$ où Ant est l'*antécédent* et $Cons$ est le *conséquent* (cf. paragraphe 2.2). La commande **sr** (Search Rule) prend les formes suivantes :

sr (T, CO, AN)	Recherche avec filtres sur conséquents et antécédents
sr (T, CO)	Recherche avec filtre sur les conséquents
sr	Forme simplifiée équivalente à sr (Back.Rewr, Goal)

Les paramètres de **sr** sont :

T	Paramètre de théorie. Ce paramètre est de la forme : $t_1.t_2.t_3$ où chaque t_i est soit un nom de théorie, soit un des mots clés : All : toutes les théories de la base de règles Rewr : les théories de règles de réécriture Back : les théories de règles en tactique “en arrière” Fwd : les théories de règles en tactique “en avant”
CO	critères de sélection des règles d'après le conséquent : Goal : le conséquent doit filtrer le but courant abs (C, D) : filtre de recherche du conséquent
AN	critères de sélection des règles d'après les antécédents : abs (A, B) : filtre de recherche des antécédents

Pour une règle $R : Ant \Rightarrow Cons$, satisfaire une contrainte sur le conséquent exprimée par **abs**(C, D) signifie que C est un filtre absolu de $Cons$ que D est un filtre relatif, c'est-à-dire :

$$Cons \in \mathcal{M}(C) \wedge \mathcal{F}(Cons) \cap \mathcal{M}(D) \neq \emptyset$$

De même, satisfaire une contrainte **abs**(A, B) signifie qu'un des antécédents de R est de la forme A et qu'il contient des sous-formules de la forme B , c'est-à-dire :

$$\exists h \in Ant, h \in \mathcal{M}(A) \wedge \mathcal{F}(h) \cap \mathcal{M}(B) \neq \emptyset$$

Un exemple de filtre de recherche est :

$$\mathbf{abs}(a = b), ((a \leftarrow b) \mathbf{_and} (a \setminus b) \mathbf{_and} \mathbf{_not}(\{x \mid Q\}))$$

qui signifie : rechercher les règles dont le conséquent (ou dont un antécédent) est de la forme *absolue* “ $a = b$ ” et qui contient une surcharge de fonction (forme “ $a \leftarrow b$ ”), une union d’ensembles (forme “ $a \vee b$ ”) et ne contient pas d’ensemble en compréhension (forme “ $\{x \mid Q\}$ ”).

Une partie B ou D est facultative s’il n’y a pas de filtre relatif à appliquer, ou bien on peut utiliser le mot-clé **No** pour indiquer l’absence de filtre. Exemple :

sr(Rewr, abs(No), a \leftarrow b, abs(a = b))

Cette commande signifie que l’on cherche les règles de réécriture dont le conséquent contient une surcharge de relation et dont l’un des antécédents est une égalité.

Le résultat de la commande **sr** fournit la liste des règles trouvées avec leur nom et leur définition. Rappelons que le nom d’une règle est toujours $t.i$ avec t nom de théorie et i numéro de règle dans la théorie.

3.4 Visualisation des OP

La commande **gs** (Global Situation) donne la liste des OP d’un composant avec pour chaque OP l’information **Proved** ou **Unproved** et le but à prouver (uniquement la partie **G**).

gs	Situation globale des OP d’un composant
-----------	---

La commande **rp** (show reduced PO) permet d’afficher le but et les hypothèses qui sont en relation plus ou moins directe avec le but.

rp (i)	Hypothèses reliées au but avec profondeur i
rp	Equivalent à rp (1)

L’entier i est strictement positif. Si $i = 1$, on a l’ensemble H_1 des hypothèses qui ont un identificateur en commun avec le but; avec $i = 2$, on a H_2 , l’ensemble des hypothèses qui ont un symbole en commun avec H_1 (donc $H_1 \subseteq H_2$), etc.

La commande **lp** (show literal PO) visualise une OP telle quelle est générée, sans les simplifications effectuées par le prouveur. En fait le résultat n’est pas très lisible et n’apporte pas grand chose.

lp ($f.n$)	Visualisation de la forme littérale de l’OP $f.n$
---------------------	---

Le paramètre indique l’obligation de preuve à visualiser : f est un nom d’opération du composant; n est un numéro valide d’OP de l’opération.

4 Navigation

4.1 Retour arrière dans la preuve

La commande **ba** (Back) permet de revenir en arrière, (défaire un certain nombre de pas de preuve qui n'ont pas abouti).

ba	Retour en arrière d'un pas
ba (n)	Retour en arrière dépendant de n

La variable n peut prendre deux types de valeurs. Dans le premier cas (**Node**), il s'agit de revenir au nœud précédent de même niveau dans l'arbre de la preuve tel qu'il apparaît dans la fenêtre en bas à gauche de l'écran du démonstrateur interactif. Dans le second cas, il s'agit d'un entier qui donne le nombre de pas de retour arrière :

n	Node < entier >	revient au même niveau d'indentation précédent recale de n pas de preuve
-----	---------------------------	---

La commande **re** (Reset PO) supprime toutes les commandes de preuve pour l'obligation de preuve courante. L'OP revient à son état initial avec le statut "unproved".

re	Retour au début de la preuve
-----------	------------------------------

4.2 Répétition de commandes

La commande **rr** (Repeat) permet de répéter la dernière commande faite dans la preuve interactive. Rappelons qu'une ligne de commande peut comporter plusieurs commandes simples séparées par le symbole "&".

rr	Répétition de la dernière ligne de commande
-----------	---

La commande **bb** (Loop) permet de lancer une ligne de commandes itérativement sur une série de sous-buts de décomposition. En effet, lorsqu'on a utilisé la commande **dc** (cas de décomposition sur un ensemble), il y a génération de n sous-buts qui ont en général la même forme. Plutôt que de répéter la même suite de commandes, on peut utiliser **bb**.

bb (l)	Répétition de l sur les cas de décomposition de la preuve
-------------------	---

Le paramètre l est de la forme suivante :

$$c_1 \& c_2 \& \dots \& c_n \& ll((d_1 \& \dots \& d_j), \dots, (z_1 \& \dots \& z_k))$$

Les c_i sont des commandes qui sont répétées. Si elles réussissent dans la preuve, le démonstrateur va à la branche suivante, sinon, il tente les d_i , et ainsi de suite, jusqu'à ce que la preuve réussisse et il va à la branche suivante, ou bien jusqu'à ce que les différentes alternatives du ll soient épuisées. La branche $ll(\dots)$

doit arriver en dernier de la liste de commandes.

La commande **te** (Try Everywhere) Permet de lancer une ligne de commandes sur un ensemble d'OP d'un composant.

te (<i>l, m.n.p</i>)	Essai des commandes <i>l</i> avec informations d'application explicites
te (<i>l</i>)	Essai des commandes <i>l</i> avec paramètres par défaut

La signification des paramètres est :

<i>l</i>	une ligne de commandes séparées par & ou bien le nom d'une OP dont on veut réappliquer la preuve	
<i>m</i>	Append Prepend Replace	tente <i>l</i> après les commandes déjà faites tente <i>l</i> avant les commandes déjà faites les commandes <i>l</i> remplacent les commandes déjà faites
<i>n</i>	Loc Gen	applique <i>l</i> aux OP de l'opération en cours applique <i>l</i> aux OP du composant
<i>p</i>	All Unproved	traite toutes les OP même déjà prouvées ne traite que les OP non prouvées

L'identification d'une OP est détaillée au paragraphe 4.3. La commande **te**(*l*) est une abréviation de :

te(*l, Replace.Loc.Unproved*)

La commande **st** (Step) permet de rejouer pas à pas une preuve qui a été sauvegardée au cours d'une session antérieure et où l'on vient de se repositionner :

st	Rejouer le pas suivant de la preuve
st (<i>n</i>)	Rejouer un certain nombre de pas suivants

Le paramètre *n* a les valeurs suivantes :

<i>n</i>	< entier >	rejoue les <i>n</i> pas de preuve suivants
	End	joue la preuve jusqu'à la fin

4.3 Navigation dans les obligations de preuve

Ces commandes permettent de passer d'une OP à une autre. Il s'agit de **ne** (Next), **go** (Goto), **gr** (Goto with Reset) et **gw** (Goto Without save) :

ne	Aller a la prochaine OP non prouvée
go (<i>f.n</i>)	Aller a l'OP identifiée par <i>f.n</i>
gr (<i>f.n</i>)	Aller a l'OP <i>f.n</i> en réinitialisant en force 0
gw (<i>f.n</i>)	Aller a l'OP <i>f.n</i> sans sauvegarder la preuve courante

L'identification d'une OP est composée de

f : nom d'une opération (ou **initialization**)
n : numéro de l'OP de l'opération

5 Sauvegarde des preuves et fin de session

Lorsqu'une preuve a été faite (ou même lorsqu'elle est en cours), on peut la sauvegarder. En fait, on sauvegarde le "script" de la preuve⁵. Après sauvegarde, le script apparaît dans la fenêtre la plus basse en bas à gauche.

sw	Sauvegarde sans confirmation
sq	Sauvegarde avec confirmation

Avec **sw** (Save without question) le système range la preuve en écrasant la précédente (s'il y en a une). Avec la commande **sq** (Save with question), le système peut informer l'utilisateur de l'état des sauvegardes pour faire confirmer le rangement.

La commande **qu** permet de quitter le démonstrateur interactif. Un dialogue peut s'engager s'il y a des preuves en cours qui n'ont pas été sauvegardées.

qu	Quitter le démonstrateur interactif
-----------	-------------------------------------

6 Modification des théories

De nouvelles règles d'inférence peuvent être ajoutées par l'utilisateur. Ces règles doivent être écrites dans le langage des théories⁶. Pour un composant (machine, raffinement ou implémentation) **NN**, les règles introduites manuellement sont contenues dans un fichier "**NN.pmm**" (Proof Manual Method) dans le répertoire **.bdp** du projet. Ce fichier est créé et modifié directement par l'utilisateur. Il contient donc des théories séparées par "&" et chaque théorie contient une suite de règles séparées par ";". Exemple de fichier avec une théorie de deux règles (les noms des règles sont calculés automatiquement):

```
THEORY Th_DISTRI3 IS
/* Th_DISTRI3.1      (Rewr) */
  binhyp(a:NATURAL)
  =>
  (a - a == 0)
;
/* Th_DISTRI3.2      (Rewr) */
  binhyp(a:NATURAL) &
  binhyp(b:NATURAL) &
  binhyp(c:NATURAL)
  =>
  (a - (b+c) == a-b-c)
END
```

5. Une étude intéressante serait de construire effectivement le "terme" de preuve d'un théorème à partir du script de la preuve et des informations données par le prouveur.

6. J.-R. Abrial, *The Logic Solver*, AtelierB, Stéria, 1996.

6.1 Chargement des règles utilisateur

Pour pouvoir utiliser un fichier de règles utilisateur, il faut le charger dans une session de preuve interactive. Cela se fait par la commande **pc** (Pmm Compile) qui compile et charge le fichier, après quoi, les règles sont utilisables par la commande **ar**.

pc	Chargement du fichier .pmm de règles utilisateur
-----------	--

Il est nécessaire d'équiper explicitement ces règles du *système de trace*, si l'on veut utiliser ce système lors des démonstrations (cf. **pr**). La méthode pour réaliser cet équipement est indiquée dans le manuel de référence (chapitre 9). Exemple de commande de chargement :

```
PRI > pc
Loading theory Th_DISTRI3
```

6.2 Vérification d'une règle utilisateur

La commande **vr** (Validation of rule) permet de vérifier qu'une règle d'inférence est valide. La vérification est faite par le prouveur de prédicat. Il est préférable de vérifier une règle avant de l'insérer dans le fichier **.pmm**. Formes de la commande :

vr ($b, r \mid i$)	Vérification de la règle r avec temps i en mode b
vr ($r \mid i$)	Vérification de la règle r en mode <i>Backward</i>
vr (b, r)	Autres cas avec paramètres par défaut
vr (r)	

La signification des paramètres est :

b	Back Fwd	règle Backward (valeur par défaut) règle Forward
r		la règle proprement dite
i		temps de coupure en seconde du prouveur de prédicat pour la validation de la règle (par défaut 60s.)

Le langage des théories est restreint à certains opérateurs (voir le manuel de référence du prouveur interactif). Exemple :

```
PRI > vr(Back, (binhyp(a:NATURAL) => (a - a == 0)))
The rule was proved
```

7 Récapitulatif dans l'ordre d'apparition

pr	Appel du cœur du prouveur
pr(Red)	Appel du prouveur réduit (pas de preuve par cas)
pr(Tac(t))	Appel du prouveur avec tactique t
pr($r.b.h, f, s$)	Appel de prouveur avec traces
pr(Tac(t), $r.b.h, f, s$)	Appel avec tactique et traces
mp	MiniProof: même chose que pr(Red)
pp	Appel du prouveur de prédicats (si peu d'hypothèses)
pp(rp.i)	Appel du prouveur avec hypothèses réduites
pp(t)	Appel du prouveur limité à la durée t (en secondes)
pp(rp.i t)	combinaison des deux cas précédents
ap	Appel du prouveur arithmétique
ap(n)	Appel du prouveur arithmétique avec limitation n
ss	Appel du simplificateur d'ensembles sur le but
tp(m)	Essai de preuve suivant le mode m
tp(m, n)	Essai n tentatives de preuve en mode m
ar(r, M)	Application d'une règle r avec le mode M
ar(T)	Application de la théorie ou de la tactique T
ct	Contradiction avec montée d'hypothèse
cts	Contradiction spéciale
fh(h)	Hypothèse h contradictoire
dc(h)	Décomposition sur un prédicat
dc(v, E)	Décomposition sur une énumération
dc($A \vee B$)	Décomposition sur une disjonction
eh(a, A, f)	Utilisation de l'égalité $a = A$ dans f
ph(v_1, \dots, v_n, h)	Particularise l'hypothèse h avec les valeurs v_1, \dots, v_n
se(v_1, \dots, v_n)	Suggestion de preuve d'un but existentiel avec v_1, \dots, v_n
dd	Déduction par "remontée" des hypothèses
dd(i)	Déduction avec une certaine force des traitements
ch	Création d'hypothèses d'après la forme du but
mh(h)	Modus ponens sur l'hypothèse h
ah(h)	Ajout de l'hypothèse h
sh(P, A)	Recherche avec filtre relatif et filtre absolu
sh(P)	Recherche avec filtre relatif seul
sr(T, CO, AN)	Recherche avec filtres sur conséquents et antécédents
sr(T, CO)	Recherche avec filtre sur les conséquents
sr	Forme simplifiée équivalente à sr(Back.Rewr, Goal)
gs	Situation globale des OP d'un composant
rp(i)	Hypothèses reliées au but avec profondeur i
rp	Equivalent à rp(1)
lp(f, n)	Visualisation de la forme littérale de l'OP $f.n$

ba	Retour en arrière d'un pas
ba (n)	Retour en arrière dépendant de n
re	Retour au début de la preuve
rr	Répétition de la dernière ligne de commande
bb (l)	Répétition de l sur les cas de décomposition de la preuve
te ($l, m.n.p$)	Essai des commandes l avec informations d'application explicites
te (l)	Essai des commandes l avec paramètres par défaut
st	Rejouer le pas suivant de la preuve
st (n)	Rejouer un certain nombre de pas suivants
ne	Aller a la prochaine OP non prouvée
go ($f.n$)	Aller a l'OP identifiée par $f.n$
gr ($f.n$)	Aller a l'OP $f.n$ en réinitialisant en force 0
gw ($f.n$)	Aller a l'OP $f.n$ sans sauvegarder la preuve courante
sw	Sauvegarde sans confirmation
sq	Sauvegarde avec confirmation
qu	Quitter le démonstrateur interactif
pc	Chargement du fichier <code>.pmm</code> de règles utilisateur
vr ($b, r i$)	Vérification de la règle r avec temps i en mode b
vr ($r i$)	Vérification de la règle r en mode <i>Backward</i>
vr (b, r)	Autres cas avec paramètres par défaut
vr (r)	

Table des matières

1	Généralités	1
1.1	Structure d’une obligation de preuve dans l’atelier B	1
1.2	Types de commandes du démonstrateur interactif	1
2	Construction des preuves	2
2.1	Appels des démonstrateurs	2
2.1.1	Appel du prouveur principal (pr)	2
2.1.2	Appel du prouveur de prédicats (pp)	3
2.1.3	Appel du prouveur arithmétique (ap)	3
2.1.4	Simplificateur d’expressions ensemblistes	4
2.1.5	Preuve par tentative	4
2.2	Application d’une règle ou d’une tactique	4
2.3	Preuve par contradiction	5
2.3.1	Contradiction	6
2.3.2	Hypothèses contradictoires	6
2.4	Preuve par cas	6
2.5	Utilisation explicite de la réécriture	7
2.6	Opérations sur les quantificateurs	7
2.6.1	Particularisation d’un “pour tout” en hypothèse	7
2.6.2	Suggestion pour un “il existe” dans le but	8
2.7	Opérations sur les hypothèses	8
2.7.1	Déduction directe	8
2.7.2	Création d’hypothèses	9
2.7.3	Modus ponens sur hypothèse	9
2.7.4	Ajout d’hypothèses	9
3	Recherche d’information	10
3.1	Principes de la recherche à partir de filtres	10
3.2	Recherche d’hypothèses dans l’OP courante	11
3.3	Recherche de règles dans la base de règles du prouveur	11
3.4	Visualisation des OP	12
4	Navigation	13
4.1	Retour arrière dans la preuve	13
4.2	Répétition de commandes	13
4.3	Navigation dans les obligations de preuve	14
5	Sauvegarde des preuves et fin de session	15
6	Modification des théories	15
6.1	Chargement des règles utilisateur	16
6.2	Vérification d’une règle utilisateur	16
7	Récapitulatif dans l’ordre d’apparition	17
	Index	20

Index des Commandes

ah : Ajout d'hypothèses	9	st : Rejouer pas à pas	14
ap : Appel prouveur arithmétique	3	sw : Sauvegarder la preuve	15
ar : Application d'une règle	4	te : Essayer sur toute OP	14
ba : Retour arrière	13	tp : Preuve par tentative	4
bb : Répétition sur décomposition	13	vr : Vérifier une règle utilisateur	16
ch : Création d'hypothèses	9		
cts : Contradiction spéciale	6		
ct : Preuve par contradiction	6		
dcs : Décomposition spéciale	6		
dc : Décomposition par cas	6		
dd : Dédution directe	8		
eh : Réécriture d'un terme	7		
ff : Changement de force	3		
fh : Hypothèses contradictoires	6		
go : Aller à une certaine OP	14		
gr : Recommencer une OP	14		
gs : Situation globale	12		
gw : Aller à sans sauvegarde	14		
lp : Visualiser OP	12		
mh : Modus ponens d'hypothèse	9		
mp : Mini Preuve	2		
ne : Aller à l'OP suivante	14		
pc : Charger les règles utilisateur	16		
ph : Particularisation "pour tout"	7		
pp : Appel prouveur de prédicats	3		
pr : Appel prouveur principal	2		
qu : Quitter la session	15		
re : Recommencer la preuve	13		
rp : Visualiser OP réduite	12		
rr : Répéter dernière commande	13		
se : Suggestion pour "il existe"	8		
sh : Recherche d'hypothèses	11		
sq : Sauvegarder la preuve	15		
sr : Recherche de règles	11		
ss : Simplificateur d'ensembles	4		