# Preuves taillées en biseau

Martin Clochard

# Preuves taillées en biseau

Martin Clochard

*Lab. de Recherche en Informatique, Univ. Paris-Sud, CNRS, Orsay, F-91405 &*
*Inria, Université Paris-Saclay, Palaiseau F-91893*

**Résumé**

Nous présentons une extension que nous avons introduite dans la logique de l'outil Why3, dans le but de décrire les étapes de raisonnement à effectuer pour décharger une obligation de preuve. Cette extension prend la forme de deux nouveaux connecteurs, que nous appelons des indicateurs de coupure. Nous montrons que l'ajout de ces deux connecteurs permet d'encoder les étapes de raisonnement d'une preuve à l'intérieur d'une formule. En particulier, cet ajout donne la possibilité d'utiliser directement la syntaxe des formules comme langage de preuve. Cette approche apporte un mécanisme léger de preuve déclarative dans un environnement où les prouveurs sont utilisés comme des boîtes noires. De plus, ce mécanisme permet une restriction de la portée des lemmes intermédiaires, évitant ainsi la saturation des prouveurs automatiques.

## 1. Introduction

Le travail que nous présentons a été réalisé dans le contexte de l'outil de vérification déductive de programmes Why3 [4]. Celui-ci repose sur l'utilisation de prouveurs externes, principalement automatiques, pour obtenir les preuves des lemmes auxiliaires ainsi que des conditions de vérification. Nous répondons ici à un besoin de guider les prouveurs lorsque les preuves sont trop complexes pour être obtenues de manière complètement automatique, sans pour autant faire appel à un prouveur interactif comme Coq. Notons que cette dernière solution est possible, mais elle n'est généralement pas très satisfaisante en pratique à cause de la traduction de la logique et de la nécessité de basculer entre les outils.

Une autre solution possible est d'ajouter au contexte logique des lemmes intermédiaires représentant les diverses étapes de la preuve. Nous avons cependant constaté que cela avait tendance à rapidement saturer les prouveurs automatiques plus qu'à les aider. Cela vient principalement du fait que les lemmes intermédiaires utilisés pour décrire une étape locale de la preuve sont a priori utilisables globalement.

Nous présentons la notion d'*indicateurs de coupure* comme une solution à ce problème. L'idée générale est d'utiliser un connecteur, appelé `by`, pour attacher localement les étapes intermédiaires à la formule qu'elles servent à prouver. Nous introduisons également le connecteur dual `so` pour attacher un corollaire à une formule. Ces connecteurs permettent notamment de traduire des preuves données en langage naturel sans pour autant apporter de changement majeur à Why3 comme l'ajout d'un langage de preuve dédié. Nous donnons une interprétation formelle des indicateurs de coupure par des règles de déduction spécifiques.

Pour traiter les indicateurs de coupure en pratique, nous présentons également un algorithme d'élimination. Celui-ci transforme une formule en un ensemble de buts correspondant aux différentes étapes de preuves induites par les indicateurs. Cet algorithme a été implémenté dans l'outil Why3, et est disponible comme une transformation dans la version courante. De manière pratique, les indicateurs de coupure ont été utilisés avec succès au sein de développements de plusieurs milliers de lignes de Why3.

## 2. By et So par l'exemple

Dans cette section, nous montrons à l'aide d'un exemple comment traduire des preuves données en langage naturel en utilisant les connecteurs `by` et `so`. Cette traduction est basée sur l'équivalence

intuitive de ces deux connecteurs avec les conjonctions de coordinations “car” et “donc” en langage naturel. Nous employons donc le connecteur `by` pour exprimer la justification, tout comme “car” en langage naturel. De manière similaire, nous traduisons la notion de conséquence exprimée par “donc” en utilisant le connecteur `so`.

Nous prenons comme exemple la preuve de l'irrationalité de la racine carrée de deux, exprimée de la manière suivante : pour tout rationnel $\frac{p}{q}$, $(\frac{p}{q})^2 \neq 2$. Autrement dit, pour toute paire d'entiers $p, q$ avec $q \neq 0$, $p^2 \neq 2q^2$. Cette proposition est représentée en Why3 comme suit :

```
lemma racine_2_irrationelle : forall p q. q <> 0 → p * p <> 2 * q * q
```

Rappelons tout d'abord une manière standard de prouver cette proposition.

— Soit $p, q$ deux entiers tels que $q \neq 0$. On cherche à montrer que $p^2 \neq 2q^2$.
— Par contradiction, on suppose $p^2 = 2q^2$.
— Soit $d$ le plus grand diviseur commun de $p$ et $q$.
— Il existe donc $a$ et $b$ tels que $da = p$ et $db = q$.
— Donc $a, b$ sont premiers entre eux.
— En divisant par $d^2$ dans l'équation initiale, $a^2 = 2b^2$.
— Donc $a$ est pair
— Donc il existe $c$ tel que $a = 2c$.
— Donc $b^2 = 2c^2$.
— Donc $b$ est également pair, ce qui contredit $a, b$ premiers entre eux. □

En utilisant les principes mentionnés au début de cette section, cette preuve se traduit comme suit (les connecteurs `by` et `so` sont associatifs à droite) :

```
lemma racine_2_irrationelle : forall p q. q <> 0 → p * p <> 2 * q * q
  by (p * p = 2 * q * q → false                (* Par contradiction *)
    by let d = gcd p q in                      (* Soit d ... *)
       exists a b. p = d * a ∧ q = d * b       (* Il existe a,b ... *)
    so gcd a b = 1                             (* a, b sont ... *)
    so a * a = 2 * b * b                       (* a^2 = 2b^2 *)
    so divides 2 a                             (* a pair *)
    so exists c. a = 2 * c                     (* Il existe c ... *)
    so b * b = 2 * c * c                       (* b^2 = 2c^2 *)
    so divides 2 b)                            (* 2 divise b *)
```

Nous retrouvons bien la même structure dans les énoncés des deux preuves. En effet, si l'on lit la formule en remplaçant `by` par le mot “car” et `so` par le mot “donc”, on retrouve pratiquement le texte initial. Cependant, la preuve présentée jusqu'ici est insuffisamment détaillée, au sens que les prouveurs automatiques utilisés par Why3 n'arrivent pas à justifier certaines étapes de raisonnement. Il faut donc ajouter des sous-preuves pour ces étapes :

— Justifier que $\gcd(a, b) = 1$. Cela vient du fait que $d = \gcd(da, db) = d \cdot \gcd(a, b)$, comme de plus $d \neq 0$ on peut simplifier.
— Justifier que $a^2 = 2b^2$. Cette équation est valide car $d^2(a^2 - 2b^2) = p^2 - 2q^2 = 0$ et $d \neq 0$
— Justifier que pour n'importe quel $a, b$ tels que $a^2 = 2b^2$, 2 divise $a$. Dans le cas contraire, $a$ est impair de la forme $2k+1$, d'où une contradiction immédiate en développant $a$ dans l'équation $a^2 = 2b^2$. De manière plus concise, c'est vrai car $a$ ne peut pas être de la forme $2k + 1$.

L'ajout de ces sous-preuves peut se traduire par :

```
lemma racine_2_irrationelle : forall p q. q <> 0 → p * p <> 2 * q * q
  by (p * p = 2 * q * q → false
    by let d = gcd p q in
      exists a b. p = d * a ∧ q = d * b
    so (gcd a b = 1 by d * gcd a b = d so d * (gcd a b - 1) = 0 ∧ d <> 0)
    so (a * a = 2 * b * b by (d*d) * (a*a-2*b*b) = 0 ∧ d <> 0)
    so (divides 2 a by not exists k. a = 2*k+1)
    so exists c. a = 2 * c
    so b * b = 2 * c * c
    so (divides 2 b by not exists k. b = 2*k+1))
```

Exprimée ainsi, nous pouvons vérifier la preuve en appliquant la transformation `split_goal_wp` de Why3, qui a été étendue dans le cadre de ce travail pour traiter les connecteurs `by` et `so` (voir section 4). La transformation produit 16 obligations de preuves, correspondant précisément aux étapes de raisonnement attendues. Nous utilisons ensuite une combinaison de prouveurs automatiques pour prouver chacun des buts, en l'occurrence Alt-Ergo [2], CVC4 [1] et E [6].

Notons que cet exemple simple aurait pu être prouvé sans les indicateurs de coupure, en énonçant une cascade de lemmes intermédiaires. Cependant, ces lemmes intermédiaires seraient restés visible par la suite, ce qui conduit à la saturation des prouveurs des exemples plus larges.

# 3. Interprétation et règles de raisonnement

Les connecteurs `by` et `so` diffèrent des connecteurs logiques habituels en ce qu'ils ne construisent pas véritablement de nouvelles formules. Nous définissons en effet l'interprétation des formules $A \,\texttt{by}\, B$ et $A \,\texttt{so}\, B$ comme étant la même que $A$. Nous avons fait ce choix car utiliser le connecteur `by` pour introduire la preuve d'une formule ne doit pas changer le sens de la formule en question, et il en va de même pour `so` par dualité. Autrement dit, ces deux connecteurs sont des indicateurs qui servent uniquement à attacher syntaxiquement une formule à une autre, et nous pouvons les effacer sans changer le sens de la formule.

Par contre, la présence d'un indicateur de coupure altère l'utilisation de la formule dans une preuve. Nous souhaitons que la preuve d'une formule $A \,\texttt{by}\, B$ se fasse en prouvant d'abord $B$, puis ensuite $A$ avec $B$ dans le contexte. $B$ sert donc d'assertion intermédiaire dans la preuve de $A$. De manière duale, la preuve d'une formule avec $A \,\texttt{so}\, B$ parmi ses hypothèses doit commencer par dériver $B$ comme hypothèse supplémentaire à partir de $A$. Nous disons donc que $B$ sert d'hypothèse compagnon à $A$. Ces deux comportements correspondent tous les deux à l'introduction d'une *coupure* sur la formule $B$ dans le raisonnement. En nous plaçant dans le calcul des séquents **LK** [5], cela revient à indiquer l'utilisation d'instances spécifiques de la règle de coupure :

$$\textsc{Cut}\ \frac{\Gamma \vdash B, \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma \vdash \Delta}$$

Nous formalisons donc le comportement de ces nouveaux connecteurs par l'ajout des règles suivantes :

$$\textsc{By-r}\ \frac{\Gamma \vdash B, A, \Delta \qquad \Gamma, B \vdash A, \Delta}{\Gamma \vdash A \,\texttt{by}\, B, \Delta} \qquad \textsc{By-l}\ \frac{\Gamma, A \vdash \Delta}{\Gamma, A \,\texttt{by}\, B \vdash \Delta}$$

$$\textsc{So-l}\ \frac{\Gamma, A \vdash B, \Delta \qquad \Gamma, A, B \vdash \Delta}{\Gamma, A \,\texttt{so}\, B \vdash \Delta} \qquad \textsc{So-r}\ \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A \,\texttt{so}\, B, \Delta}$$

Les règles By-r et So-l sont les instances attendues de la règle de coupure, tandis que les autres règles correspondent à l'interprétation triviale des connecteurs. Pour se convaincre de la correction de ces règles, il est suffisant de montrer que si un séquent est prouvable alors sa version "effacée", où l'on a éliminé les indicateurs de coupure, l'est également. C'est immédiat car l'effacement rend toutes les règles supplémentaires admissibles dans **LK**. Remarquons par ailleurs qu'en employant la règle d'affaiblissement, nous pouvons également montrer l'équiprouvabilité du séquent et de sa version effacée.

Notons cependant que la règle By-r ne correspond pas tout à fait à l'intuition donnée pour le connecteur `by`. La règle correspondant à l'intuition est la suivante, dérivée immédiatement par affaiblissement de By-r :

$$\textsc{By-r'}\ \frac{\Gamma \vdash B, \Delta \qquad \Gamma, B \vdash A, \Delta}{\Gamma \vdash A \,\texttt{by}\, B, \Delta}$$

La formule supplémentaire $A$ dans la règle By-r permet de conserver la possibilité de prouver directement $A$ au lieu de l'assertion intermédiaire $B$, ce qui est nécessaire pour obtenir l'équiprouvabilité entre les formules $A \,\texttt{by}\, B$ et $A$. Nous pouvons également voir la règle By-r comme une preuve par contradiction puisque c'est équivalent à supposer $\neg A$ pour prouver $B$.

Nous pourrions de manière similaire définir une règle duale So-L' qui correspondrait à l'oubli de l'hypothèse $A$ dans la prémisse $\Gamma, A, B \vdash \Delta$ de la règle So-L, mais elle n'a pas d'intérêt pratique.

Remarquons que toute preuve en calcul des séquents **LK** standard peut être complètement encodée dans les formules via le connecteur `by`. Nous définissons cet encodage des preuves par :

$$T\left(\frac{(\pi_i)_{i\in I}}{\Gamma \vdash \Delta}\right) = \left(\forall(x)_{x\in fv(\Gamma\cup\Delta)}.\ \bigwedge \Gamma \to \bigvee \Delta\right) \texttt{by} \bigwedge_{i\in I} T(\pi_i)$$

De par l'interprétation donnée à `by`, il est immédiat que la formule résultante est équivalente à la conclusion de la preuve encodée. De plus, l'application répétée de la règle By-R' et de la règle de conjonction à droite sur le séquent $\vdash T(\pi)$ construit un arbre de preuve dont les feuilles correspondent précisément aux noeuds de $\pi$. En particulier, il est possible d'utiliser les indicateurs de coupure pour donner toute la structure d'une preuve si nécessaire.

# 4. Algorithme d'élimination des indicateurs de coupure

## 4.1. Algorithme théorique

Dans cette section, nous présentons un algorithme de traitement des indicateurs de coupure. L'objectif est d'éliminer les connecteurs `by` et `so` d'une formule arbitraire sans perdre l'information donnée par les indicateurs. Il faut bien évidemment que la formule initiale soit une conséquence logique de la formule obtenue par l'algorithme d'élimination, de façon à pouvoir réduire la preuve de la formule initiale à celle après élimination. L'implication réciproque n'est pas nécessaire, et ne sera pas vraie en général.

Pour représenter fidèlement les informations données par les indicateurs de coupure, la formule obtenue doit regrouper de manière conjonctive les feuilles de l'arbre de preuve obtenu par l'application des règles de **LK** étendu. Par exemple, pour la formule $A \,\texttt{by}(B \,\texttt{so}\, C)$ on souhaite obtenir une formule isomorphe à $B \wedge (B \to C) \wedge (B \wedge C \to A)$, qui correspond à la structure de preuve suivante :

$$\dfrac{\dfrac{\dfrac{\vdash B}{\vdash B \,\texttt{so}\, C}}{\vdash B \,\texttt{so}\, C, A} \qquad \dfrac{\dfrac{B \vdash C}{B \vdash C, A} \qquad B, C \vdash A}{B \,\texttt{so}\, C \vdash A}}{\vdash A \,\texttt{by}(B \,\texttt{so}\, C)}$$

Remarquons que l'arbre de preuve donné ci-dessus contient deux affaiblissements sur la partie droite, dont l'un correspondant à la règle dérivée By-R'. Nous avons fait le choix d'ajouter ces affaiblissements pour que les formules obtenues par notre algorithme soient plus naturelles. En effet, laisser plusieurs formules à droite du séquent reviendrait à systématiquement donner la possibilité de faire une preuve par contradiction sur la formule que l'on cherche à prouver. Nous pouvons par ailleurs nous permettre de faire de tels affaiblissements car nous voulons seulement que la formule finale implique la formule de départ. Dans le cas particulier où l'on souhaite faire une preuve par contradiction, nous avons toujours la possibilité de le faire explicitement via la construction $\neg A \to (\bot \,\texttt{by}\, B)$.

Les formules traitées par l'algorithme d'élimination sont décrites par la syntaxe suivante :

$$\begin{array}{rcl} \varphi & ::= & A \\ & | & \top \mid \bot \\ & | & \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \to \varphi \\ & | & \neg\varphi \\ & | & \varphi \,\texttt{by}\, \varphi \mid \varphi \,\texttt{so}\, \varphi \\ & | & \forall x.\varphi \mid \exists x.\varphi \end{array}$$

L'algorithme d'élimination se compose de quatre fonctions auxiliaires mutuellement récursives $e^+, e^-, c^+, c^-$ respectant les propriétés ci-dessous :
— si $c^+(\varphi)$ est vérifiée, alors $e^+(\varphi)$ implique $\varphi$.
— si $c^-(\varphi)$ est vérifiée, alors $\varphi$ implique $e^-(\varphi)$.

L'algorithme final consiste alors simplement à calculer $e^+(\varphi) \wedge c^+(\varphi)$. L'idée intuitive est la suivante : $e^+$ remplace les sous-formules $A \,\texttt{by}\, B$ qui apparaissent en position positive par $B$. Cela revient à remplacer le but $A$ par l'assertion intermédiaire $B$. De manière duale, $e^-$ remplace $A \,\texttt{so}\, B$ en position négative par $A \wedge B$, ajoutant ainsi l'hypothèse compagnon $B$ au contexte. Les deux fonctions $c^+$ et $c^-$ collectent les conditions sous lesquelles ces remplacements sont possibles, respectivement $B \to A$ et $A \to B$ dans les cas ci-dessus. On appelle ces conditions "conditions de bord" dans ce qui suit. Ces conditions correspondent précisément à la conjonction des étapes de raisonnement induites par les indicateurs de coupure dans **LK** étendu.

Les quatre fonctions récursives sont définies comme suit :

$$
\begin{array}{lcllcll}
e^+(\varphi) & = & \varphi & e^-(\varphi) & = & \varphi & \varphi \in \{A, \top, \bot\} \\
e^+(\varphi_1 \,\texttt{op}\, \varphi_2) & = & e^+(\varphi_1) \,\texttt{op}\, e^+(\varphi_2) & e^-(\varphi_1 \,\texttt{op}\, \varphi_2) & = & e^-(\varphi_1) \,\texttt{op}\, e^-(\varphi_2) & \texttt{op} \in \{\wedge, \vee\} \\
e^+(\varphi_1 \to \varphi_2) & = & e^-(\varphi_1) \to e^+(\varphi_2) & e^-(\varphi_1 \to \varphi_2) & = & e^+(\varphi_1) \to e^-(\varphi_2) & \\
e^+(\neg\varphi) & = & \neg e^-(\varphi) & e^-(\neg\varphi) & = & \neg e^+(\varphi) & \\
e^+(\varphi_1 \,\texttt{by}\, \varphi_2) & = & e^+(\varphi_2) & e^-(\varphi_1 \,\texttt{by}\, \varphi_2) & = & e^-(\varphi_1) & \\
e^+(\varphi_1 \,\texttt{so}\, \varphi_2) & = & e^+(\varphi_1) & e^-(\varphi_1 \,\texttt{so}\, \varphi_2) & = & e^-(\varphi_1) \wedge e^-(\varphi_2) & \\
e^+(Qx.\varphi) & = & Qx.e^+(\varphi) & e^-(Qx.\varphi) & = & Qx.e^-(\varphi) & Q \in \{\forall, \exists\}
\end{array}
$$

$$
\begin{array}{lcll}
c^i(\varphi) & = & \top & \varphi \in \{A, \top, \bot\} \\
c^i(\varphi_1 \,\texttt{op}\, \varphi_2) & = & c^i(\varphi_1) \wedge c^i(\varphi_2) & \texttt{op} \in \{\wedge, \vee\} \\
c^+(\neg\varphi) & = & c^-(\varphi) & \\
c^-(\neg\varphi) & = & c^+(\varphi) & \\
c^i(Qx.\varphi) & = & \forall x.c^i(\varphi) & Q \in \{\forall, \exists\} \\
c^+(\varphi_1 \to \varphi_2) & = & c^-(\varphi_1) \wedge (e^-(\varphi_1) \to c^+(\varphi_2)) & \\
c^-(\varphi_1 \to \varphi_2) & = & c^+(\varphi_1) \wedge c^-(\varphi_1) \wedge (e^-(\varphi_1) \to c^-(\varphi_2)) & \\
c^+(\varphi_1 \,\texttt{by}\, \varphi_2) & = & c^+(\varphi_2) \wedge c^-(\varphi_2) \wedge (e^-(\varphi_2) \to e^+(\varphi_1) \wedge c^+(\varphi_1)) & \\
c^-(\varphi_1 \,\texttt{by}\, \varphi_2) & = & c^-(\varphi_1) & \\
c^+(\varphi_1 \,\texttt{so}\, \varphi_2) & = & c^+(\varphi_1) & \\
c^-(\varphi_1 \,\texttt{so}\, \varphi_2) & = & c^-(\varphi_1) \wedge (e^-(\varphi_1) \to e^+(\varphi_2) \wedge c^+(\varphi_2) \wedge c^-(\varphi_2)) &
\end{array}
$$

Nous pouvons aisément vérifier les deux propriétés de correction par récurrence sur la formule. Par exemple, considérons le traitement d'une formule existentielle $\exists x.\varphi$ en position négative. Nous devons donc démontrer qu'il existe un témoin $x_1$ pour $e^-(\varphi)$, sachant que $\forall x.c^-(\varphi)$ est vraie et qu'il existe un témoin $x_0$ pour $\varphi$. Par hypothèse de récurrence, comme $c^-(\varphi)$ est vraie pour $x = x_0$, $x_0$ est le témoin voulu. Remarquons qu'il est nécessaire de transformer la quantification existentielle en quantification universelle dans les conditions de bord pour pouvoir effectuer ce raisonnement. De manière générale, les conditions de bord s'accumulent de manière conjonctive.

Notons que l'algorithme que nous avons décrit traite les opérateurs binaires $\wedge$ et $\vee$ de manière différente de l'implication pour les conditions de bord. En effet, l'implication préserve $e^-(\varphi_1)$ en tant que contexte quand c'est possible pour les conditions de bord, tandis que les autres opérateurs ne préservent pas de contexte. Nous avons fait ce choix pour préserver la nature symétrique de $\wedge$ et $\vee$. Cependant, si l'on considère que $\wedge$ et $\vee$ sont asymétriques alors d'autres définitions sont possibles, comme par exemple :

$$
\begin{array}{lcl}
c^+(\varphi_1 \wedge \varphi_2) & = & c^+(\varphi_1) \wedge c^-(\varphi_1) \wedge (e^-(\varphi_1) \to c^+(\varphi_2)) \\
c^-(\varphi_1 \wedge \varphi_2) & = & c^-(\varphi_1) \wedge (e^-(\varphi_1) \to c^-(\varphi_2)) \\
c^+(\varphi_1 \vee \varphi_2) & = & c^+(\varphi_1) \wedge (e^+(\varphi_1) \vee c^+(\varphi_2)) \\
c^-(\varphi_1 \vee \varphi_2) & = & c^-(\varphi_1) \wedge c^+(\varphi_1) \wedge (e^+(\varphi_1) \vee c^-(\varphi_2))
\end{array}
$$

Dans ce contexte, le traitement de l'implication n'est alors qu'un cas particulier de traitement asymétrique de la disjonction.

## 4.2. Algorithme implémenté dans Why3

Pour traiter les indicateurs de coupure dans Why3, nous avons intégré un algorithme d'élimination des indicateurs de coupure au sein de la transformation `split_goal_wp`. Cette transformation est utilisée pour transformer un but à vérifier en une collection de sous-buts

indépendants dont la conjonction implique le but initial. Elle consiste de manière effective à éliminer les indicateurs, puis à découper ensuite la structure conjonctive de la formule obtenue pour obtenir les différents sous-buts. L'algorithme utilisé pour éliminer les indicateurs diffère cependant légèrement de l'algorithme théorique pour des raisons de complexité, le calcul des conditions de bord pouvant sinon créer des formules de taille exponentielle.

Le phénomène se produit à cause de la duplication des conditions de bord dans le traitement des connecteurs binaires asymétriques, comme l'implication. En effet, pour une formule $\varphi_1 \to \varphi_2$ les conditions de bord négatives de $\varphi_1$ sont propagées à la fois dans les conditions de bord positive et négative de $\varphi_1 \to \varphi_2$. Par ailleurs, comme le connecteur `by` donne un contexte différent aux conditions de bord négatives et positives, il n'est pas possible de traiter le problème simplement par regroupement des branches conjonctives identiques. Une famille de formules exhibant un comportement pathologique est par exemple :

$$\varphi_0 = A, \varphi_{n+1} = (\varphi_n \to U_n) \,\mathtt{by}\, T_n$$

où $(U_n, T_n)_{n\in\mathbb{N}}$ sont des atomes distincts. On peut montrer que le nombre de branches conjonctives dans les conditions de bord majore la suite de Fibonacci.

Pour pallier au problème, nous avons renforcé les conditions de bord du connecteur `by` de manière à toujours donner un contexte identique aux conditions de bord positives et négatives originaire d'une même formule :

$$\begin{array}{lcll} c^+(\varphi_1 \,\mathtt{by}\, \varphi_2) & = & c^+(\varphi_2) \wedge c^-(\varphi_2) \wedge (e^-(\varphi_2) \to e^+(\varphi_1) \wedge c^+(\varphi_1)) & \text{(avant renforcement)} \\ c^+(\varphi_1 \,\mathtt{by}\, \varphi_2) & = & c^+(\varphi_2) \wedge c^-(\varphi_2) \wedge c^+(\varphi_1) \wedge (e^-(\varphi_2) \to e^+(\varphi_1)) & \end{array}$$

De cette façon, nous pouvons traiter les conditions de bord de manière uniforme. Nous calculons donc directement $c^{all}(\varphi) = c^+(\varphi) \wedge c^-(\varphi)$ en simplifiant les duplications quand elles apparaissent, ce qui rend les formules générées de taille polynomiales. En calculant simultanément $c^{all}, e^+$ et $e^-$ on obtient un algorithme d'élimination en temps polynomial. Le résultat final de l'élimination est alors donné par $c^{all}(\varphi) \wedge e^+(\varphi)$.

Pour terminer la description de l'algorithme, il nous reste à préciser si nous utilisons les règles de calcul symétriques ou asymétriques pour la conjonction et la disjonction. Comme Why3 supporte les deux variantes pour chacun de ces connecteurs, le choix se fait selon leur nature.

Les restrictions présentent néanmoins deux défauts. Premièrement, comme on ne peut plus distinguer les conditions de bord positives et négatives, on est obligé de toutes les inclure au résultat final. Ce n'est cependant pas un problème en pratique car les formules sur lesquelles l'élimination est utilisée ont normalement des condition de bord négatives triviales. Deuxièmement, comme on a restreint les conditions de bord générées pour le connecteur `by`, il y a des cas assez rares où des éléments de contexte désirables disparaissent. Le premier exemple pratique que nous ayons rencontré est de la forme $((A \,\mathtt{by}\, B) \vee (C \,\mathtt{by}\, D)) \,\mathtt{by}\, E$, où l'hypothèse $E$ n'est visible que pour prouver $B \vee D$.

# 5. Conclusion

Les indicateurs de coupure que nous avons présenté permettent de pallier le manque de possibilités de décrire la structure d'une preuve dans l'outil Why3. Nous avons ainsi obtenu un mécanisme de preuve déclaratif, permettant de représenter des preuves données en langage naturel. Le sujet de l'expression de preuves formelles en langage plus ou moins naturel est très vaste. Le langage de preuve déclaratif Isar [8] a été proposé au-dessus de l'assistant de preuve Isabelle. La bibliothèque ssreflect pour l'assistant de preuve Coq offre un langage de tactique permettant de formuler des preuves de manière plus déclarative que les tactiques standard. Corbineau a proposé [3] un langage de preuve déclaratif au-dessus de Coq. Le langage ForTheL [7] permet d'écrire des textes de preuves très proches de l'anglais, tout en restant vérifiables par ordinateur.

Notre travail est clairement moins ambitieux et plus limité, mais a l'avantage de la simplicité puisque nous n'avons pas eu besoin de définir un langage complet dédié à la preuve.

La prochaine étape de ce travail est d'éliminer les restrictions imposées pour obtenir un algorithme d'élimination efficace. Une piste possible est d'utiliser des variables booléennes pour coder la structure des conditions de bord.

**Remerciements** Nous remercions Claude Marché et Andrei Paskevich pour leur conseils et leur suggestions.

# Références

[1] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *Proceedings of the 23rd international conference on Computer aided verification*, CAV'11, pages 171–177, Berlin, Heidelberg, 2011. Springer-Verlag.

[2] François Bobot, Sylvain Conchon, Évelyne Contejean, Mohamed Iguernelala, Stéphane Lescuyer, and Alain Mebsout. The Alt-Ergo automated theorem prover, 2008. `http://alt-ergo.lri.fr/`.

[3] Pierre Corbineau. A declarative language for the Coq proof assistant. In Marino Miculan, Ivan Scagnetto, and Furio Honsell, editors, *Types for Proofs and Programs, International Conference, TYPES 2007, Cividale del Friuli, Italy, May 2-5, 2007, Revised Selected Papers*, volume 4941 of *Lecture Notes in Computer Science*, pages 69–84. Springer, 2008.

[4] Jean-Christophe Filliâtre and Andrei Paskevich. Why3 — where programs meet provers. In Matthias Felleisen and Philippa Gardner, editors, *Proceedings of the 22nd European Symposium on Programming*, volume 7792 of *Lecture Notes in Computer Science*, pages 125–128. Springer, March 2013.

[5] G. Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, (39) :176–210, 405–431, 1934.

[6] Stephan Schulz. System description : E 0.81. In David A. Basin and Michaël Rusinowitch, editors, *Second International Joint Conference on Automated Reasoning*, volume 3097 of *Lecture Notes in Computer Science*, pages 223–228. Springer, 2004.

[7] Konstantin Verchinine, Alexander V. Lyaletski, Andrey Paskevich, and Anatoly V. Anisimov. On correctness of mathematical texts from a logical and practical point of view. In Serge Autexier, John A. Campbell, Julio Rubio, Volker Sorge, Masakazu Suzuki, and Freek Wiedijk, editors, *Intelligent Computer Mathematics, 9th International Conference, AISC 2008, 15th Symposium, Calculemus 2008, 7th International Conference, MKM 2008, Birmingham, UK, July 28 - August 1, 2008. Proceedings*, volume 5144 of *Lecture Notes in Computer Science*, pages 583–598. Springer, 2008.

[8] Markus M. Wenzel. *Isabelle/Isar — a versatile environment for human-readable formal proof documents.* PhD thesis, Institut fur Informatik, TU München, 2002.

# A. Appendice : exemple Why3 complet

```
module Sqrt2

  use import int.Int
  use import number.Parity
  use import number.Divisibility
  use import number.Gcd

  lemma anneau_integre : forall a b. a * b = 0 → a = 0 \/ b = 0

  lemma racine_2_irrationelle : forall p q. q <> 0 → p * p <> 2 * q * q
      by (p * p = 2 * q * q → false
        by let d = gcd p q in
          exists a b. p = d * a ∧ q = d * b
        so (gcd a b = 1 by d * gcd a b = d so d * (gcd a b - 1) = 0 ∧ d <> 0)
        so (a * a = 2 * b * b by (d*d) * (a*a-2*b*b) = 0 ∧ d <> 0)
        so (divides 2 a by not exists k. a = 2*k+1)
        so exists c. a = 2 * c
        so b * b = 2 * c * c
        so (divides 2 b by not exists k. b = 2*k+1))

end
```