Une preuve sous *Raffalli's* `AF2`

P. Manoury

6 janvier 1999

On se propose de démonter en utilisant l'implémentation due à C. Raffalli du système d'aide à la preuve `af2` un petit résultat du calcul des propositions:

On se place dans le fragment du calcul des propositions ne comprenant que le seul connecteur $\leftrightarrow$ que nous appellerons «fragment ssi». On a alors que

une proposition du fragment ssi est une tautologie si et seulement si chaque variable propositionnelle y a un nombre pair d'occurences.

Nous ne présentons, pour l'instant que la preuve que la parité du nombre d'occurence implique d'être une tautologie.

1 Formalisation

1.1 Introduction

Nous allons formaliser notre problème dans le calcul des prédicats du second ordre AF2, puis mener la preuve dans l'implantation qu'a donné C. Raffalli de ce système formel.

Entiers et booléens

Nous supposons que nous disposons d'une théorie déjà définie des *entiers naturels* ainsi que des *booléens*. Nous disposons donc de deux symboles unaires de prédicats $\mathbb{N}$ et $\mathbb{B}$.
Les formules

$$\forall x(\mathbb{N}(x) \Rightarrow \Phi) \text{ et } \forall x(\mathbb{B}(x) \Rightarrow \Phi)$$

seront abrégées en

$$\forall x : \mathbb{N}.\Phi \text{ et } \forall x : \mathbb{B}.\Phi$$

Les constantes entières seront notées usuellement 0, 1, 2, etc... les constantes booléennes, TT et FF.

Le fragment *ssi*

Les *propositions* du fragment ssi seront codées dans le calcul des prédicats comme des *termes* d'individus. On introduit deux symboles de fonctions *var* et *ssi*. Le symbole *var* est unaire, il servira à coder les variables propositionnelles en utilisant les entiers: $var(0)$, $var(1)$, $var(2)$, etc... Le symbole *ssi* est binaire, il représentera le connecteur propositionnel d'équivalence $\Leftrightarrow$ Ainsi, la proposition usuellement notée $p \Leftrightarrow q$ devient, dans notre formalisation, $ssi(p,q)$.

Par abus de langage, si n est un entier, nous nous permettrons d'en parler comme d'une variable propositionnelle.

Plus, nous ferons des propostions du fragment ssi un *type de données* au même titre que les booléens et les entiers. Nous disposerons donc d'un symbole de prédicat unaire SSI et pourrons écrire $\forall x : SSI.\Phi$. En faisant des propositions du fragment ssi un type de données, nous obtenons le droit de raisonner par *récurrence* sur ces propositions.

Notations AF2

Pour pouvoir utiliser l'éditeur de preuve AF2, il nous faut passer de la notation usuelle des formules à la syntaxe comprise par le système. Nous donnons ici quelques éléments de cette syntaxe.

Le quantificateur $\forall$ s'écrit `/\`

Le quantificateur $\exists$ s'écrit `\/`

Le connecteur $\Rightarrow$ s'écrit `->`

L'application d'un symbole de fonction à ses arguments s'écrit *à la LISP*: $var(n)$ s'écrit `(var n)`; $ssi(p_1, p_2)$ s'écrit `(ssi p1 p2)`.

De même, si P est un symbole (constante ou variable) de prédicat, $P(t)$ s'écrit `(P t)`, ou plus simplement `P t`, par juxtaposition.

Dans la suite, nous donnerons, pour chaque formule, sa notation usuelle et sa notation AF2.

1.2 L'énoncé

On définit le fragment ssi comme le type de données SSI à deux constructeurs `var` et `ssi`. Les entiers naturels serviront à contruire les variables propositionnelles. On utilise pour cela, une formule du second ordre. On introduit donc le symbole SSI et on *définit* la formule $SSI(p)$ à une variable libre p, comme une abréviation de

$$\forall X(\forall n:\mathbb{N}.X(var(n)) \Rightarrow \forall p_1 \forall p_2(X(p_1) \Rightarrow X(p_2) \Rightarrow X(ssi(p_1,p_2))) \Rightarrow X(p))$$

Soit, dans la syntaxe AF2

```
def SSI x = /\X(/\n:N(X (var n)) ->
                /\p1/\p2(X p1 -> X p2 -> X (ssi p1 p2))
                -> X x).
```

La distribution de valeur de vérité sur les variables est formalisée comme une fonction des entiers dans les booléens. La valeur de vérité d'une proposition est alors définie comme le résultat de la fonction booléenne `eval` dont l'un des arguments est une distribution de valeur de vérité sur les variables.

Remarque

Pour définir l'évaluation d'une proposition, nous avons besoin de pouvoir parler de *toute distribution de valeur de vérité*. En d'autres termes, nous avons besoin de *variables de fonction* que nous pouvons *quantifier*. Nous nous autorisons donc cette extension du langage. Ainsi, l'expression *pour toute fonction v de $\mathbb{N}$ dans $\mathbb{B}$* pourra être notée $\forall v : \mathbb{N} \to \mathbb{B}$, ou encore, dans la syntaxe AF2, `/\ v:(N --> B)`

La fonction *eval* qui prend en argument une distribution de valeurs de vérité v et une proposition est définie, récursivement, par les équations

$$\begin{array}{lcll} eval(v, var(n)) & = & v(n) & \\ eval(v, ssi(p_1,p_2)) & = & eval(v,p_2) & \text{si } eval(v,p_2) = TT \\ eval(v, ssi(p_1,p_2)) & = & not(eval(v,p_2)) & \text{si } eval(v,p_2) = FF \end{array}$$

où la fonction *not* est la négation booléenne.

En syntaxe AF2, nous utiliserons la construction fonctionnelle ternaire `if then else` pour exprimer en une seule nos deux dernières équations; la négation booléenne est notée `~~` Nous posons donc les deux axiomes (`claim`) équationnels nommés `eval_var` et `eval_ssi`

```
claim eval_var /\v:(N --> B) /\n:N (eval v (var n)) = (v n).
claim eval_ssi /\v:(N --> B) /\p1,p2:SSI
       (eval v (ssi p1 p2)) = if (eval v p1) then (eval v p2) else ~~(eval v p2).
```

Une proposition p est une tautologie si et seulement si sa valeur de vérité est TT pour toute distribution v de valeur de vérité. On écrira donc $tauto(p)$ comme abrv́iation de la formule à une variable libre p

$$\forall v : \mathbb{N} \rightarrow \mathbb{B}.eval(v, p) = TT$$

En syntaxe AF2, on pose :

```
def tauto p = /\v:(N --> B)(eval v p = TT).
```

Pour pouvoir parler du nombre d'occurence d'une variable dans une proposition, on définit simplement la fonction chargée de ce calcul. Soit *nbocc* un symbole de fonction binaire. Si n est un entier et p une roposition, $nbocc(n, p)$ reourne le nombre d'occurence de $var(n)$ dans p. pose

$$\begin{array}{lcll} nbocc(n, var(n)) & = & 1 & \\ nbocc(n, var(m)) & = & 0 & \text{si } n \neq m \\ nbocc(n, ssi(p_1, p_2)) & = & nbocc(n, p_1) + nbocc(n, p_2) & \end{array}$$

Malheureusement, la syntaxe d'AF2 ne supporte pas la notation décimale des entiers. On note donc N0 pour 0 et (S N0) pour 1. Le symbole S dénotant la fonction successeur des entiers naturels. Le symbole == dénote le test booléen d'égalité.

```
claim nbocc_var /\n,m:N
       (nbocc n (var m)) = if (n == m) then (S N0) else N0.
claim nbocc_ssi /\n:N /\p1,p2:SSI
       (nbocc n (ssi p1 p2)) = (nbocc n p1)+(nbocc n p2).
```

Un entier n est pair s'il existe un entier m tel que $n = 2 \times m$. Donc chaque variable a un nombre pair d'occurrence dans une proposition p si et seulement si

$$\forall n : \mathbb{N}.\exists m : \mathbb{N}.nbocc(n, p) = 2 \times m$$

En sytaxe AF2, on pose la définition de la propriété even :

```
def even p = /\n:N \/m:N (nbocc n p) = N2*m.
```

où * est l'opérateur de multiplication et N2 l'entier 2.

Compte tenu de l'ensemble de ces définitions, 'énoncé à démontrer devient :

$$\forall p : SSI.even(p) \Rightarrow tauto(p)$$

C'est à dire, dans la syntaxe AF2

```
goal /\p:SSI (even p -> tauto p).
```

1.3 Quelqu'autres définitions utiles

On aura besoin de raisonner par induction généralisée sur la taille des propositions. On définit une fonction *size* de mesure sur les propositions :

$$\begin{array}{ll} size(var(n)) & = 1 \\ size(ssi(p_1, p_2)) & = 1 + size(p_1) + size(p_2) \end{array}$$

En AF2, on pose les deux axiomes équationnels

```
claim size_var /\n:N (size (var n) = N1).
claim size_ssi /\p1,p2:SSI (size (ssi p1 p2) = (size p1)+(size p2)).
```

où N1 représente l'entier 1.

Nous raisonnerons par induction généralisée sur la taille des propositions. Au second ordre, on exprime ce principe d'induction par la formule

$$\forall X(\forall p:SSI.(\forall p':SSI.(size(p') < size(p) \Rightarrow X(p'))) \Rightarrow X(p))$$

Soit l'axiome AF2 suivant :

```
claim SSI.ind
 /\X ((/\p:SSI (/\p':SSI(size p' < size p -> X p') -> X p))
      -> /\p:SSI (X p)).
```

Indépendament de son compte, on aura besoin de parler de l'occurence d'une variable dans une proposition. Une variable a une occurence dans une proposition lorsque son nombre d'occurence (calculé par la fonction *nbocc*) dans cette proposition est non nul. On définit donc la formule d'appartenance d'une variable n à une proposition p, notée $mem(n,p)$ par

$$nbocc(n,p) \neq 0$$

En syntaxe AF2, on pose

```
def mem n p = nbocc n p != N0.
```

Pour mener à bien notre preuve, nous utiliserons de façon primordiale une relation que l'on peut appeler de *permutation* (notée *perm*) entre propositions. Une proposition p_1 est permutation d'une proposition p_2 lorsqu'elles ont excatement le même nombre d'occurence de chacune des variables propositionnelles. Ce qu'exprime la formule à deux variables libres p_1 et p_2

$$\forall n:N.nbocc(n,p_1) = nbocc(n,p_2)$$

En syntaxe AF2, on pose

```
def perm p1 p2 = /\n:N (nbocc n p1 = nbocc n p2).
```

Parmi toutes les propositions, certaines ont la propriété remarquable autant qu'utile d'être *plates*. On définit récursivement par les deux clauses

1. toute variable propositionnelle est plate.
2. si x est une variable propositionnelle et p est une proposition plate alors $ssi(x,p)$ est une proposition plate.

Plus précisément, dans la seconde clause, on utilise le fait que les variables sont données par le symbole *var* pour poser

2. pour tout entier n, si p est une proposition plate alors $ssi(var(n),p)$ est une proposition plate.

Au second ordre, on définit la formule $flat(p)$ comme une abréviation de la formule

$$\forall X(\forall n:I\!N.X(var(n)) \Rightarrow (\forall n:I\!N\forall p':SSI.(X(p') \Rightarrow X(ssi(var(n),p')))) \Rightarrow X(p))$$

En AF2, on pose la définition

```
def flat p =
 /\X (/\n:N X (var n)
      -> /\n:N/\p:SSI(X p -> X (ssi (var n) p))
      -> X p).
```

Les propositions comptent un nombre fini de variables. Ne voulant pas nous lancer dans une définition telle que « *l'ensemble des variables d'une proposition* » et de sa « *finitude* », nous utilisons le fait que nos variables ne sont rien d'autre que des entiers pour définir la notion de *variable maximale*[1]. On utilise la relation d'ordre sur les entiers comme relation d'ordre sur les variables. Une variable n est alors maximale dans une proposition lorsqu'aucune variable m telle que $m > n$ n'a d'occurrence dans p. Ce qui se formule

$$\forall m : I\!N(n < m \Rightarrow nbocc(m,p) = 0$$

Ce qui se définit en AF2 par :

```
def max_var n p = /\m:N (n < m -> nbocc m p = N0).
```

Enfin, nous aurons besoin d'itérer des sommes. Si f est une fonction de $I\!N$ dans $I\!N$, on note usuellement

$$\sum_{i=0}^{n} f(i)$$

ce que l'on peut définir par induction sur n par

$$\sum_{i=0}^{0} = f(0)$$

$$\sum_{i=0}^{n+1} = f(n+1) + \sum_{i=0}^{n}$$

Cette définition ne dépend que de f et n, on peut donc se donner un opérateur *sum* qui prend en argument la fonction f et l'entier n pour poser les équations

$$\begin{array}{lcl} sum(0,f) & = & f(0) \\ sum(n+1,f) & = & f(n+1) + sum(n,f) \end{array}$$

Soit, selon la syntaxe AF2, les axiomes équationnels

```
claim sum_0 /\f:(N --> N) (sum N0 f = f N0).
claim sum_S /\n:N /\f:(N --> N) (sum (S n) f = f (S n) + sum n f).
```

2 L'idée de la preuve

Si une proposition contient un nombre pair d'occurences de chaque variable[2], nous dirons que cette proposition *est paire*, ce que l'on noté $even(p)$. Si deux propositions sont dans la relation de permutation, nous dirons que l'une *est une permutation* de l'autre, ce que l'on a noté $perm(p_1, p_2)$.

1. L'existence d'une variable maximale pour chaque proposition nous donne bien qu'on ne peut avoir une infinité de variable ayant au moins une occurence dans une proposition.

2. ce qui inclut l'absence d'occurence

2.1 Par cas sur les propositions paires

Les propositions paires ont cette propriété remarquable que

ou bien elle sont de la forme $x \leftrightarrow x$ pour une variable propositionnelle x donnée,

ou bien il existe une variable propositinnelle x est une proposition p', elle même paire, telles que $(x \leftrightarrow x) \leftrightarrow p$ en est une permutation.

Ceci nous permettra de raisonner par cas sur les propositions paires. Il faut, bien entendu établir le bien fonder de cette dichotomie. Nous montrons donc un lemme affirmant que si une proposition p est paire alors ou bien il existe une variable n telle que $p = ssi(var(n), var(n))$, ou bien il existe une variable n et une proposition paire p' telle que p est une permutation de $ssi(ssi(var(n), var(n)), p')$. Ce qui se formule

$$\forall p : SSI.(even(p) \Rightarrow \exists n : I\!N.p = var(n) \vee \exists n : I\!N \exists p' : SSI.perm(p, ssi(ssi(var(n), var(n)), p')))$$

Soit, selon la syntaxe `AF2`

```
goal /\p:SSI
 (even p
  -> \/n:N p = ssi (var n) (var n)
      or \/n:N \/p':SSI (perm p (ssi (ssi (var n) (var n)) p')
                         & even p')).
```

Nous appelerons `even_case` ce lemme.

2.2 De l'invariance par permutation

Les propositions en relation de permutation conservent deux propriétés : leur taille et leur valeur de vérité.

L'invariance pour la taille est établie par le lemme `perm_size_equal` :

$$\forall p_1 : SSI.\forall p_2 : SSI.(perm(p_1, p_2) \Rightarrow size(p_1) = size(p_2)$$

C'est à dire, en `AF2`

```
goal /\p1,p2:SSI (perm p1 p2 -> (size p1 = size p2)).
```

L'invariance de la valeur de vérité nous est donnée par le lemme `perm_eval` :

$$\forall p_1 : SSI.\forall p_2 : SSI.(perm(p_1, p_2) \Rightarrow \forall v : I\!N \rightarrow I\!B.eval(v, p_1) = eval(v, p_2)$$

Soit, en `AF2`

```
goal /\p1,p2:SSI
 (perm p1 p2 -> /\v:(N-->B)(eval v p1 = eval v p2)).
```

La taille pour démontrer que la taille des propositions est invariante par permutation ont utilise le fait que la taille d'une proposition est égale à la somme du nombre d'occurence de l'ensemble des variables y occurant. Nous utilisons pour formuler ce lemme la notion de variable maximale d'une proposition et montrons que si n est la variable maximale d'une proposition p alors

$$size(p) = \sum_{i=0}^{n} nbocc(i, p)$$

Pour formuler cette égalité, nous voulons utiliser notre opérateur sum, mais celui-ci n'admet en second argument que de fonctions de $I\!N$ dans $I\!N$. Nous utiliserons ici la λ notation pour le terme $nbocc(m, p)$ en une fonction dont l'argument et m et où p est un paramètre. Ce qui nous donne la formule

$$\forall p : SSI.\forall n : I\!N(max\_var(n, p) \Rightarrow size(p) = sum(n, \lambda m.nbocc(m, p))$$

Soit, en syntaxe `AF2`

```
goal /\p:SSI /\n:N (max_var n p -> size p = sum n \m(nbocc m p)).
```

Nous appellerons `size_sum_nbocc` ce lemme.

La valeur de vérité l'invariance de la valeur de vérité par permutation est démontrée en utilisant la propriété plus faible de l'invariance pour les seules permutations plates jointe à l'existence d'une permutation plate pour chaque proposition. Le lemme `flat_perm_eval` exprime cette invariance particulière, il s'énonce :

$$\forall p_1 : SSI.\forall p_2 : SSI.(flat(p_2) \Rightarrow perm(p_1, p_2) \Rightarrow \forall v : \mathbb{N} \to \mathbb{B}(eval(v, p_1) = eval(v, p_2))$$

Soit, selon la syntaxe `AF2`

```
goal /\p1:SSI /\p2:SSI (flat p2 -> perm p1 p2
       -> /\v:(N --> B) eval v p1 = eval v p2).
```

2.3 Le résultat principal

Soit donc à montrer que si une proposition p est paire alors p est une tautologie. On procède par induction généralisée sur la taille de p, ce qui nous donne comme hypothèse d'induction : pour toute proposition p' de taille strictment inférieure à p, si p' est paire alors p' est une tautologie.
Raisonnons maintenant par cas sur p que nous supposons paire.

a) si p est égale à $x \leftrightarrow x$, pour une certaine variable x, le résultat est immédiat.

b) s'il existe une variable x et une proposition paire p' telle que p est une permutation de $(x \leftrightarrow x) \leftrightarrow p'$, on obtient le résultat par hypothèse d'induction, la taille de p' étant bien strictement inférieure à celle de p.