Introduction à PhoX : deuxième partie

P. Rozière
Université Paris VII

06/03/2003

TP2 MT3062 2002/2003

1 Introduction.

Cette séance poursuit la formalisation des ordres commencée dans la première séance. A ce propos on complète l'exploration des commandes de preuves par celles qui concernent le quantificateur existentiel.

On donne ensuite un exemple d'utilisation de l'opérateur de description définie. On termine par la preuve d'un petit résultat d'analyse élémentaire qui, tel qu'il est formalisé, n'utilise que des propriétés élémentaires de l'ordre.

2 Axiomatisation des ordres stricts : résultats élémentaires repris de la séance de TP 1.

Remarque. Ce qui suit est un corrigé de la première partie de la première séance de TP1.

2.1 Signature.

```
>PhoX> Sort d.

Cst Infix[5] x "<" y : d -> d -> prop.
```

2.2 Axiomes.

```
>PhoX> claim ordre_transitif /\x,y,z(x < y -> y < z -> x < z).
claim ordre_antireflexif /\z ~ z < z.
```

2.3 Quelques propriétés.

```
>PhoX> prop  ordre_strict_antisymetrique  /\x,y(x< y -> ~ y < x).
      |- /\x,y (x < y -> ~ y < x)
>PhoX> intros.
   H := x < y   |- ~ y < x
>PhoX> intro.
   H0 := y < x   |- False
>PhoX> apply ordre_transitif with H and H0.
   G := x < x
>PhoX> apply ordre_antireflexif with x.
   G0 := ~ x < x
>PhoX> apply G0 with G.
```

```
   G1 := False   |- False
>PhoX> axiom G1.
save.
```

A partir de maintenant on va demander aux prouveurs de détecter automatiquement les axiomes, ce qui allégera les preuves.

```
>PhoX> flag auto_lvl 1.

def Infix[5] x "<=" y = x < y or x = y.

fact ordre_reflexif /\x x <= x.
intro.
      |- x <= x
>PhoX> intro r.
      |- x = x
>PhoX> intro.
save.

prop ordre_large_antisymetrique
/\x,y (x <= y -> y <= x -> x = y).
intros.
   H := x <= y,  H0 := y <= x  |- x = y
>PhoX> left H. (* raisonnement par cas, le cas x = y est traité automatiquement *)
   H := x < y
>PhoX> left H0. (* raisonnement par cas *)
(* cas H0 := y = x *)
  from H0.
(* cas H0 := y < x *)
  apply ordre_strict_antisymetrique with H and H0.
   G := False
>PhoX> elim G.
save.

fact ordre_large_transitif /\x,y,z (x<= y -> y <= z -> x <= z).
intros.
   H := x <= y  H0 := y <= z     |- x <= z
>PhoX> left H. (* raisonnement par cas *)
(* cas H := x = y *)
  rewrite H.
(* cas H := x < y *)
  left H0.
  (* cas H0 := y = z *)
    rewrite_hyp H H0.
    (* H := x < z *)
    intro l.
  (* cas H0 := y < z *)
    intro l.
    (*    |- x < z *)
    elim ordre_transitif with H and H0.
save.
```

3 Ordre total, fonction sup.

3.1 Axiomatisation d'ordre strict total.

```
>PhoX> claim ordre_total /\x,y (x<y or x= y or y < x).
```

Montrez que l'ordre large est un ordre large total :

```
>PhoX> fact ordre_large_total /\x,y(x<= y or y <= x).
....
save.
```

cette varainte est utile pour la dernière proposition

```
>PhoX> lem ordre_total2
/\x,y (x <= y or y < x).
(* -> *)
intros.
apply ordre_total with x and y.
lefts G.
 intro r.
 intro l. intro r.
 intro l. intro l.
(* -> *)
save.
```

3.2 Le sup de deux éléments, la règle d'introduction de l'existentielle.

Une fois que l'on sait que l'ordre total, on peut montrer que deux éléments ont toujours un majorant qui est l'un des deux éléments. Pour cela on a besoin d'utiliser les règles du quantificateur existentiel. Voyons tout d'abord la règle d'introduction. Elle se décompose en deux commandes, `intro` qui fournit une *variable existentielle*, notée `?1`, `?2` ... Ce n'est pas une variable du langage. On peut "remplacer" cette variable existentielle par n'importe quel terme `t` du langage grâce à la commande `instance ?1 t`. C'est ce que montre l'exemple qui suit.

```
>PhoX> prop uval4 /\A /\x(A x -> \/ x A x).
intros.
   H := A x
>PhoX> intro.
      |- A ?1
>PhoX> instance ?1 x.
save.
```

Le fait d'avoir décomposé la règle d'introduction de l'existentielle en dissociant l'introduction et l'instanciation est parfoir utile quand l'instanciation à choisir n'est pas immédiate, et que l'on a besoin de "calculer" sur la variable existentielle pour déterminer sa valeur. On verra plus tard la règle d'élimination. Vous êtes en mesure de prouver la proposition suivante (qui utilise que l'ordre est total !).

```
>PhoX> lem ordre_maj /\x,y \/ z (x <= z & y <= z).
....
save.
```

3.3 La fonction sup.

Pour prouver qu'une relation est fonctionnelle, on montre l'existence et l'unicité de l'image.

```
>PhoX> prop def_sup /\x,y\/!z ((z=x or z=y) & x <= z & y <= z).
....
save.
```

3.4 Opérateur de description définie (approche très partielle).

En logique du premier ordre, si l'on a montré un énoncé de la forme `/\x1, x2 ...xn\/!z P x1...xn z` alors "on ne change pas la théorie" (on ne démontre pas de nouveau théorème dans le langage non étendu) en ajoutant un nouveau symbole de fonction n-aire f vérifiant `P x1 ... xn (f x1...xn)`. C'est ce que permet dans PhoX l'opérateur `Def`. On peut lire `Def \z P x1...xn z` comme "l'unique z vérifiant P x1...xn z".

```
>PhoX> def sup x y = Def \z ((z=x or z=y) & x <= z & y <= z).
```

Les énoncés qui suivent sont des "redites" partielles du résultat `def_sup`. Ne vous attardez pas sur les preuves. Certains résultats peuvent être utiles par la suite. Il faudrait les compléter.

```
>PhoX> fact sup_l /\x,y x <= sup x y.
intros.
apply def_sup with x and y.
apply Def.axiom with G.
lefts G0.
from H0.
save.

fact sup_r /\x,y y <= sup x y.
intros.
apply def_sup with x and y.
apply Def.axiom with G.
lefts G0.
from H1.
save.
```

Il va s'avérer parfois plus lisible d'utiliser ces notations.

```
>PhoX> def Infix[5] x ">" y = y < x.
def Infix[5] x ">=" y = y <= x.
```

4 Un résultat d'analyse élémentaire.

On se propose de démontrer le résultat suivant sur IR : si une fonction est positive sur IR+, si elle a une limite nulle en l'infini et si elle possède un maximum qu'elle atteint sur tout intervalle fermé borné (c'est le cas des fonctions continues), alors elle possède un maximum qu'elle atteint sur IR+. Il s'avère que pour démontrer ce résultat exprimé ainsi on a besoin seulement des propriétés d'ordre total pour IR. dans la suite IR est identifié aux objets de sortes `d`.

4.1 Les définitions utiles.

Pour pouvoir parler de "positif" on introduit la constante 0 :

```
>PhoX> Cst  zero:d.
```

Quelques prédicats sur une fonction `f:d->d`. D'abord, la fonction `f` est positive sur les réels positifs.

```
>PhoX> def positive f = /\ x >=zero (zero <= f(x)).
```

Ce prédicat signifie pour une fonction `f`, dont on sait par ailleurs qu'elle est positive sur les réels positifs, qu'elle a une limite nulle en +infini.

```
>PhoX> def pseudo_lim f = /\ eps > zero \/ N >=zero /\ x > N (f(x) < eps).
```

La fonction `f` a un majorant qu'elle atteint sur tout intervalle fermé o, a.

```
>PhoX> def maj_comp f =
/\ a (zero <= a -> \/ c ((zero<=c & c<= a) & /\x(zero<= x -> x <= a -> f(x) <= f(c)))).
```

4.2 Une conséquence du tiers-exclu.

Le principe du tiers-exclu `/\A (A or ~ A)` se démontre avec le raisonnement par l'absurde. Il est parfois pratique d'utiliser ce principe directement : complétez la preuve du lemme qui suit.

Remarque. Ce n'est pas un énoncé de la logique du premier ordre.

```
>PhoX> lem nulle_ou_nonnulle
/\f(positive f -> /\ x >=zero f(x)=zero or \/ x >= zero zero < f(x)).
intros.
elim excluded_middle with \/x>=zero zero < f x.
(* cas H0 := \/x >= zero  zero < f x *)
....
(* cas H0 := ~ \/x >= zero  zero < f x *)
....
save.
```

4.3 La règle d'élimination de l'existentielle.

La commande `elim` permet d'utiliser un énoncé existentiel.

```
>PhoX> prop uval5 /\C/\A (/\x(A x -> C) -> (\/x A x -> C)).
intros.
(* H := /\x(A x ->) C  H0 := \/x A x   |- C *)
elim H0.
(* H1 := A x *)
elim H with H1.
save.
```

Comme pour la disjonction, la commande `left` peut être aussi utilisée (elle ne conserve pas la formule à laquelle elle est appliquée).

```
>PhoX> prop uval6 /\C/\A (/\x(A x -> C) -> (\/x A x -> C)).
intros.
left H0.
(* H0 := A x *)
elim H with H0.
save.
```

4.4 Le résultat annoncé.

Prouvez maintenant l'énoncé suivant. On peut se servir de la fonction `sup` et des faits `sup_r` et `sup_l`, mais le plus simple est d'utiliser le lemme `ordre_maj`.

Idée d'une preuve – Après avoir éliminé le cas trivial où f est partout nulle, choisir x tel que f(x) non nul et N tel qu'au delà de N f est majorée par f(x) (propriété `pseudo_lim`). Utilisez la propriété (`maj_comp` f) – f atteint son majorant sur un intervalle fermé borné – sur un l'intervalle $0, N$. Remarquez que forcément x <= N (utilisez `ordre_total2` et résoudre le cas x > N par antireflexivité).

Remarque. Ce n'est pas un énoncé de la logique du premier ordre.

```
>PhoX> prop maj_atteinte
 /\f(positive f -> pseudo_lim f -> maj_comp f ->
                         \/ d (zero<=d & /\x(zero <= x -> f(x) <= f(d)))).
intros.
elim nulle_ou_nonnulle with H.
(* cas H2 := /\x >=zero f x = zero *)
....
save.
```