TP1 MT3062 2003/2004 – 18/02/04

1 Introduction.

Remarque. Quand ce texte fait référence à la documentation, il s'agit du polycopié d'introduction ("Quelques éléments pour débuter avec le vérificateur de preuves PhoX").

Le but de cette séance est de vous familiariser avec les principales commandes du logiciel PhoX. Tout d'abord remarquez que si vous lisez la version `.phx` de ce fichier, le texte que vous êtes en train de lire, est ce que l'on appelle du *commentaire*. Il n'est pas évalué par le logiciel (voir documentation §3 p. 4). On va utiliser comme support la théorie de l'ordre. Au début vous avez juste à évaluer au fur et à mesure les entrées qui vous sont proposées, et à comprendre leur fonctionnement en inspectant la sortie. Ensuite vous êtes invités à utiliser ces commandes pour compléter des preuves. Le logiciel ne vous autorise pas à "passer" des entrées. Les "...." doivent être effacés, et remplacés par une preuve.

2 Première partie.

2.1 Signature.

Nous devons tout d'abord définir la sorte pour les objets de base. Appelons-la `d`. Entrez la commande suivante (lire le § 4 p. 5 de la documentation).

```
>PhoX> Sort d.
```

si la commande est acceptée, le système vous répond : Sort d defined
nous introduisons le symbole < (ordre strict) :

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

Le mot clef `Cst` signifie que l'on introduit un nouveau symbole du langage (`Cst` pour "constante" ce mot étant utilisé en un sens plus large que vu en cours). Après le " :" vous avez la *sorte* du symbole, c'est à dire le fait que c'est un prédicat d'arité 2 sur la sorte `d`. Le mot clef `Infix` signifie que la syntaxe est celle usuelle (`x < y` et non par exemple `< x y`).

2.2 Axiomes.

Pour les axiomes nous n'utiliserons que la quantification universelle `/\x A x` l'implication `A -> B` et la négation `~ A`. Voir la documentation § 3.4 p. 5 pour les conventions d'écriture des formules. Le mot-clef `claim` introduit les axiomes.

Remarque. Les sortes et les formules utilisent la même notation `->`, dans deux sens distincts.

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

Remarquez que l'implication "associe à droite" : $A \to B \to C$ signifie $A \to (B \to C)$. Il s'agit bien de la transitivité, on pourrait utiliser la conjonction : $\forall x, y, z((x < y \wedge x < z) \to x < z)$, mais la première forme s'avère plus maniable dans les preuves.

```
>PhoX> claim ordre_antireflexif /\z ~ z < z.
```

En fait la négation $\neg A$ est définie par "A implique une contradiction", $A \to \bot$. Ceci peut aider à comprendre son maniement dans les preuves.

Maintenant ces axiomes sont connus du système, comme le montre ce qui suit :

```
>PhoX> print ordre_transitif.
print ordre_antireflexif.
```

Vous pourrez utilisez ces noms dans les preuves.

2.3 Quelques preuves avec $\forall$, $\to$, $\neg$.

On veut prouver l'antisymétrie qui pour un ordre strict s'exprime par "quelquesoient x et y, si $x < y$ alors $\neg y < x$".

On va commencer par prouver quelques énoncés très simples du calcul propositionnel et du calcul des prédicats pour illustrer les commandes de preuve utiles. Lire le début du § 4.3.1, et le § 4.3.2 p 8 de la documentation (le détail de l'exemple n'est pas utile pour le moment).

Remarque. ces énoncés ne sont pas du premier ordre !

Voici un premier exemple très simple utilisant essentiellement l'implication. Vous pouvez vérifier dans la documentation le sens des commandes utilisées.

```
>PhoX> prop taut1 /\ A (A -> A).
intro. (* soit A quelconque prouvons A -> A *)
intro. (* Supposons A, prouvons A *)
axiom H. (* c'est "évident" *)
save.    (* cqfd *)
```

un exemple utilisant la négation :

```
>PhoX> prop taut2 /\A(A -> ~ ~ A).
intros. (* Supposons H:=A pour A quelconque, factorise les deux premiers intro,
           on pourrait écrire aussi "intro 2" *)
intro. (* Supposons H0:=~A, prouvons une contradiction,
          cf ci-dessus ~A signifie A -> false *)
apply H0 with H. (* de ~A et A on déduit une contradiction *)
axiom G.         (* c'est ce que nous voulions *)
save.            (* cqfd *)
```

on aurait pu utiliser `elim` plutôt que `apply`, (voir documentation).

```
>PhoX> prop taut3 /\A(A -> ~ ~ A).
intros.
intro.
elim H0 with H.
save.

prop taut4 /\A(A -> ~ ~ A).
intros.
intro.
elim H0.
axiom H.
save.
```

Enfin un exemple utilisant la quantification sur les objets du (premier ordre) et l'axiome de transitivité. Notez la forme : `apply ... with ... and ...`

```
>PhoX> prop transitive3 /\x,y,z,t(x < y -> y < z -> z < t -> x <t).
intros.
apply ordre_transitif with H and H0.
(* terminez la preuve *)
....
save.
```

Vous allez maintenant prouver l'antisymétrie de l'ordre en utilisant ces commandes. Commencez bien-sûr par réfléchir à la preuve que vous feriez sur papier, n'hésitez pas à l'écrire auparavant.

Dans cette preuve vous aurez besoin d'utiliser les axiomes, les commandes `apply` et `elim` le permettent avec la même syntaxe que ci-dessus, par exemple `apply ordre_transitif with ...` , `elim ordre_antireflexif.`

```
>PhoX> prop  ordre_strict_antisymetrique  /\x,y(x< y -> ~ y < x).
....
save.
```

On peut demander au prouveur de détecter automatiquement les axiomes, ce qui allège certaines preuves. C'est ce que fait la commande suivante. Elle prend effet pour toutes les preuves qui suivent, jusqu'à une nouvelle commande `flag auto_lvl ...`

```
>PhoX> flag auto_lvl 1.
```

recopiez le script de la preuve précédente et faites disparaître les commandes devenues inutiles.

2.4 Des preuves utilisant $\vee$ et $=$

On veut maintenant montrer que les axiomes d'ordre strict pour "inférieur" entraînent les axiomes d'ordre large pour "inférieur ou égal". Introduisons tout d'abord une notation pour "inférieur ou égal".

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

les exemples propositionnels suivants illustrent comment l'on démontre une disjonction

```
>PhoX> prop taut_orl /\A,B(A -> (A or B)).
intros. (* Soient A et B, supposons A, montrons A ou B *)
intro l. (* A ou B se déduit de A, "l" est une abréviation de "left" *)
save.

prop taut_orr /\A,B(B -> (A or B)).
intros. (* Soient A et B, supposons B, montrons A ou B *)
intro r. (* A ou B se déduit de B "r" est une abréviation de "right" *)
save.
```

Vous aurez aussi besoin de propriétés simples de l'égalité, comme celle-ci.

```
>PhoX> prop eq_ref /\x x = x.
intro. (* soit x quelconque  *)
intro. (* on sait que "=" est réflexive et c'est une façon "standard"
          de montrer l'égalité *)
save.
```

Montrez que l'ordre est réflexif

```
>PhoX> fact ordre_reflexif /\x x <= x.
....
save.
```

Pour l'antisymétrie vous aurez besoin d'un propriété de l'absurde, `False`, qui est que `l'absurde` entraîne n'importe quelle proposition.

```
>PhoX> prop taut_abs /\ A(False -> A).
intros. (* supposons H:= False montrons A *)
elim H.  (* False entraîne n'importe quoi, en particulier A *)
save.
```

Vous avez également besoin d'utiliser une disjonction en *raisonnant par cas.*
on comprend mieux ce qui suit en supprimant la détection automatique des axiomes :

```
>PhoX> flag auto_lvl 0.

prop taut5 /\A,B( (A or B) -> (~ A -> B)).
intros. (* supposons H:= A or B, H0 := ~ A, montrons B *)
elim H. (* deux cas suivant que l'on a A ou l'on a B *)
 (* cas H1:= A *)
 elim H0 with H1. (* de ~A et A on déduit n'importe quoi, en particulier B
                     remarquez que l'on a "factorisé" plusieurs éliminations *)

 (* cas H1:= B *)
 axiom H1. (* c'est évident *)
save.
```

Les réponses seraient moins lourdes avec la commande `left`

```
>PhoX> prop taut6 /\A,B( (A or B) -> (~ A -> B)).
intros.
left H.
 (* cas H:= B *)
  axiom H.
 (* cas H:= A *)
  elim H0 with H.
save.
```

Si l'on repasse en détection automatique des axiomes l'un des cas est traité automatiquement.

```
>PhoX> flag auto_lvl 1.

prop taut7 /\A,B( (A or B) -> (~ A -> B)).
intros.
left H.
 (* cas H:= B traité automatiquement *)
 (* cas H:= A *)
  elim H0 with H.
save.
```

Vous pouvez avoir besoin aussi d'autres propriétés simples de l'égalité.

```
>PhoX> prop eq_sym /\x,y (x=y -> y = x).
intros. (* Soient x, y quelconques, supposons H:= x=y montrons y =x *)
from H. (* PhoX sait que l'égalité est symétrique,
           voir la documentation pour from *)
save.
```

Montrez maintenant que l'ordre est anti-symétrique. Vous pouvez utiliser la proposition déjà démontrée `ordre_strict_antisymetrique`.

```
>PhoX> prop ordre_large_antisymetrique
/\x,y (x <= y -> y <= x -> x = y).
....
save.
```

Pour la transitivité de l'ordre nous avons besoin d'une nouvelle propriété de l'égalité, qui est exprimée dans la proposition ci-dessous, par la commande `rewrite`

```
>PhoX> prop egalc /\A /\x,y(y=x -> A x -> A y).
intros. (* Supposons H:= y=x, H0:= A x, montrons A y *)
rewrite H. (* A y est A x car y=x *)
save.
```

la commande `rewrite` utilise l'équation de gauche à droite, on peut lui demander de l'utiliser dans l'autre sens :

```
>PhoX> prop egal /\A /\x,y(x=y -> A x -> A y).
intros. (* Supposons H:= x=y, H0:= A x, montrons A y *)
rewrite -r H.
save.
```

on peut également vouloir transformer une hypothèse, ce que permet la commande `rewrite_hyp`

```
>PhoX> prop egal1 /\A /\x,y(x=y -> A x -> A y).
intros.
rewrite_hyp H0 H.
save.

prop egalc1 /\A /\x,y(y=x -> A x -> A y).
intros.
rewrite_hyp H0 -r H.
save.
```

Remarque. L'ordre des arguments est important : `-r` juste avant l'equation qu'il faut lire en sens inverse.

Vous pouvez maintenant montrer la transitivité de l'ordre large

```
>PhoX> fact ordre_large_transitif /\x,y,z (x<= y -> y <= z -> x <= z).
....
save.
```

3 Seconde Partie.

On va maintenant faire l'exercice inverse. Vous aller définir la théorie des ordres larges, avec le symbole `<=` comme symbole primitif, et les trois axiomes usuels, définir l'ordre strict `<` comme `x <=y & x != y`. Ici `x != y` est déjà défini comme `~ x=y`. Pour pouvoir faire ceci, comme nous souhaitons garder les mêmes noms avec un sens différent, il faut réinitialiser le système, ce que permettent les commandes :

```
>PhoX> flag auto_lvl 0.
del $<=. del $<.
```

Ces commandes suppriment "$<=$" et "$<$" et tout ce qui les utilisent. On enlève aussi la détection des axiomes pour les premières preuves.

Vous avez vu toutes les commandes nécessaires sauf celles qui utilisent la conjonction `&`. Voici des énoncés qui illustrent leur usage.

```
>PhoX> prop taut_et /\A,B(A -> B -> (A & B)).
intros.
intro.
axiom H.
axiom H0.
save.

prop taut_etl /\A,B((A & B) -> A).
intros.
elim H with[l].
save.

prop taut_etr /\A,B((A & B) -> B).
intros.
elim H with[r].
save.
```

Très souvent on utilise plutôt la commande `left`

```
>PhoX> prop taut_etl1 /\A,B((A & B) -> A).
intros.
left H.
axiom H.
save.

flag auto_lvl 1.

prop taut_et_imp /\A,B,C((A -> B -> C) -> ((A & B) -> C)).
intros.
left H0.
elim H with A and B.
save.
```

un exemple qui illustre la façon de prouver une conjonction, avec la détection automatique des axiomes :

```
>PhoX> prop taut_imp_et /\A,B,C (((A & B) -> C) -> (A -> B -> C)).
intros.
elim H.
intro.
save.
```

Répondez maintenant à la question. Certains lemmes peuvent être utiles comme l'antisymétrie de l'ordre strict ou plus simplement le lemme suivant : `lem ordre1 /\x,y (x <= y -> ~ y < x).`

```
>PhoX> ....
```