TP : Sémantique et Logique de Hoare

Ce TP comporte 2 parties. La première partie consiste en quelques exercices vous permettant de prendre en main le système d'aide à la preuve Isabelle/HOL [NPW02]. Dans la deuxième partie, il s'agira de prouver formellement la correction partielle de quelques programmes en utilisant la logique de Hoare.

1 Premiers pas avec Isabelle/HOL

1.1 Contexte

Le logiciel Isabelle2005 (http ://www.cl.cam.ac.uk/Research/HVG/Isabelle/) est un environnement générique utilisé pour définir des systèmes formels, puis les utiliser pour prouver des théorèmes (formules exprimées dans ces systèmes). Dans ce cadre, il est possible de décrire son propre système formel. Il existe plusieurs instantiations différentes d'Isabelle ; parmi elles, l'une est basée sur la logique d'ordre supérieur (Isabelle/HOL). C'est celle que nous utiliserons dans ce TP. Isabelle/HOL dispose d'une bibliothèque très complète sur les résultats de base sur les données et les théorèmes représentables en logique d'ordre supérieur. Par ailleurs, ce système permet de prouver de nombreux théorèmes de manière automatique.

1.2 Utilisation d'Isabelle/HOL en TP

Depuis les terminaux des salles de TP, le logiciel Isabelle/HOL peut être lancé en utilisant la commande suivante :

```
/users/prof/magaud/bin/Isabelle
```

L'exécution de cette commande lance une session `xemacs` et démarre un processus `isabelle`. Toute l'interaction avec Isabelle se fera ensuite par l'intermédiaire de la fenêtre `xemacs` ouverte.

Un fichier contenant des définitions et des énoncés de théorèmes ainsi que leurs preuves s'appelle une théorie et a pour extension `.thy`. Cette extension est nécessaire pour permettre à `xemacs` d'identifier ce fichier comme un script de preuves pour Isabelle/HOL.

- Interaction avec Isabelle/HOL dans `xemacs` : boutons, menus ou commandes au clavier
- Terminer un processus `isabelle` : cliquer bouton droit dans le buffer et sélectionner Exit Isabelle.
- Démarrer un processus `isabelle` : Ctrl-c Ctrl-n ou boutons Next, Use, Goto.
- Isabelle supporte l'utilisation de *X-Symbol* dans `xemacs`. Cela permet d'afficher joliment $\forall$, $\rightarrow$ plutot que d'avoir les notations `forall` et `-->`. Pour l'activer, aller dans le menu Proof General/Options/X-Symbol.

Tous les fichiers .thy nécessaires pour ce TP peuvent être recopiés depuis `/users/prof/magaud/Isabelle/TP`.

2 Premières preuves

Une preuve se développe interactivement dans un buffer `xemacs`. On commence par énoncer un théorème puis on tape les différentes commandes permettant de démontrer le théorème recherché.
L'énoncé du théorème est donné entre guillements après le mot-clé `lemma`. La preuve se fait par étapes en appliquant successivement des commandes `apply` (`apply simp`, `apply auto`, `apply arith`...), puis en concluant par la commande `done` une fois la preuve terminée. On peut aussi procéder en une seule étape avec une commande du genre `by (induct t) auto`.

Vous trouverez dans l'annexe A un exemple de développement en Isabelle/HOL sur les entiers naturels.

Une bibliothèque sur les entiers naturels existe déjà et nous l'utiliserons dans la suite du TP.

2.1 Exercices

1. Chargez le fichier `TP_logique.thy` contenant quelques démonstrations de notions de base de la logique, terminez les preuves imcomplètes de ce fichier en vous inspirant des preuves déjà effectuées. Vous devez remplacer toutes les occurences de `oops` par les étapes de démonstration adéquates.
2. Etudiez le fichier `TP_nat.thy` donné en annexe A. Exécutez-le pas-à-pas pour voir comment les opérations comme `plus` ou `mult` sont construites et comment on prouve des propriétés à leur sujet.

3. Chargez maintenant le fichier `TP_listes.thy` et les preuves sur la concaténation et la longueur des listes. Complètez les définitions et les preuves en rajoutant si nécessaire des lemmes intermédiaires. Terminez la preuve que `reverse` est une fonction involutive.

3 Preuves avec la logique de Hoare

Vous trouverez une description du langage de programmation WHILE ainsi que quelques informations utiles pour l'utilisation de l'extension de Isabelle/HOL pour la logique de Hoare à l'adresse suivante `file ////users/prof/magaud/Isabelle2005/src/HOL/Hoare/README.html`. Il s'agit d'un langage très simple que nous utiliserons pour prouver la correction partielle de quelques programmes.

3.1 Syntaxe du langage utilisé

ne rien faire :	SKIP
affectation :	_ := _
séquence :	_; _
conditionnelle :	IF _ THEN _ ELSE _ FI
boucle :	WHILE _ INV _ DO _ OD

Vous trouverez en annexe B le début du fichier TP_Hoare.thy à utiliser pour réaliser vos preuves.

4 Exercices

1. Testez les exemples fournis dans le fichier TP_hoare.thy.
2. Prouvez la correction de l'opération d'échange vue au TD 1 (échange par addition et soustraction).
3. Prouvez la correction d'une opération de multiplication (n*m) par additions successives (addition de m, répétée n fois).
4. Ecrivez un programme de calcul de la division euclidienne de a par b. Précisez bien que les variables sont des entiers naturels par la notation `VARS (x : :nat) ...` au lieu de `VARS x ...`
5. Ecrivez un programme de recherche de la racine carrée d'un entier entier par additions successives en partant de 0.
6. Ecrivez un programme de calcul du pgcd de deux entiers et prouvez sa correction partielle (voir la définition de pgcd dans /users/prof/magaud/Isabelle2005/src/HOL/Hoare/Arith2.thy).
7. Ecrivez un boucle while pour calculer la factorielle de N, montrez sa correction partielle vis-à-vis de la définition récursive de fac fournie dans Arith2.thy.
8. Fibonacci : écrire une définition récursive de fibonacci avec primrec sur nat puis montrer qu'une implantation impérative calcule bien la même chose...

A Fichier TP_nat.thy

```
theory TP_nat=Main:

(* définition d'un type de données pour les entiers naturels *)
datatype NAT =
    ZERO
  | SUCC "NAT"

consts (* types des opérations sur les entiers naturels *)
  plus :: "NAT => NAT => NAT"     (infixr "+" 65)
  mult :: "NAT => NAT => NAT"     (infixr "*" 65)

primrec
  "plus ZERO x = x" (* règles de calcul pour l'addition *)
  "plus (SUCC x) y = (SUCC (plus x y))"

print_theorems (* pour voir ce qui vient d'être défini *)
```

```
primrec
  "mult ZERO x = ZERO"
  "mult (SUCC n) m = (plus m (mult n m))"

lemma "(a*(b+c))=(a*b)+(a*(c::nat))" (* exemple de preuve par induction *)
apply (induct a)
apply auto
done

lemma [simp] : "((a+b)+c)=(plus a (plus b c))"
by (induct a) auto

lemma [simp] : "a+b=b+(a::nat)"
apply auto
done

end
```

B Fichier TP_Hoare.thy

```
theory Tp_hoare
imports "~~/src/HOL/Hoare/Hoare" "~~/src/HOL/Hoare/Arith2"
begin

lemma " VARS a
{true} SKIP {true}"
apply vcg;
done

(* affectation *)
lemma "VARS a
  {true}
  a := 3
  {a = 3}"
apply vcg
done

(* Somme de 2 entiers n et m *)

lemma "VARS s
  {true}
  s:=N+M
  {s=N+M}"
by vcg

[...]
```

Références

[Hoa69] C. A. R. Hoare. An Axiomatic Basis for Computer Programming. *Communications of the ACM*, 12(10) :576–580, 1969.

[NPW02] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.