TESTS ET VÉRIFICATIONS

E.I.S.T.I. G.I.

INTRODUCTION À LA VÉRIFICATION LOGICIELLE

LOGIQUE DE HOARE
INTRODUCTION

- *Objectif*: la logique de Hoare sert à formaliser la preuve de la correction des programmes informatiques:

<pre-condition>
programme
<post-condition>

- *Signification*: si la pre-condition est vraie alors, après l'exécution du programme, la post-condition est vraie.

- *Méthode*: utilisation de règles de déductions logiques.

LOGIQUE DE HOARE

RAPPEL: LOGIQUE DES PRÉDICATS

LANGAGE D'ALPHABET {V,C,F,P,L}

- V : variables, C : constantes
- F : fonctions de {C x V} -> {C x V} d'arité quelconque
- P : predicats de {C x V} -> Bool d'arité quelconque
- L : connecteurs entre predicats:

```
!       : negation
||      : ou logique non exclusif
&&      : et logique
==>     : implication logique
\forall : quantificateur universel
\exist  : quantificateur existentiel
```

LOGIQUE DE HOARE

TRIPLET DE HOARE

Les formules à démontrer sont appelées des *triplets de Hoare*, du nom de leur inventeur Charles Antony Richard Hoare (http://fr.wikipedia.org/wiki/Charles_Antony_Richard_Hoare)

```
//Forme générale
{p}S{q} avec p et q des formules de la logique des prédicats et S un programme

//Exemples
{x>=0}x=x+1;{x>0}

{(x>0)&&(y>=0)}z=y/x;{(x>0)&&(y>=0)&&(z>=0)}
```

LOGIQUE DE HOARE
RÈGLES DE DÉDUCTION

- Pour démontrer la validité d'un *triplet de Hoare* nous allons utiliser des *règles de déductions* qui auront toutes la même forme:

> **premisses**
> **|**
> **conclusion**

- *Signification*: si toutes les premisses sont vraies, la conclusion est vraie

- *Résultat*: un arbre de déduction dont les feuilles sont des axiomes (formule toujours vraie dans la logique de Hoare) et la racine le triplet de Hoare à démontrer.

LOGIQUE DE HOARE
RÈGLES DE DÉDUCTIONS

- Axiome:

```
{p(t)} x=t; {p(x)}
```

- Composition:

```
{p}S1{q}    {q}S2{r}
    |             |
   {p} S1;S2 {r}
```

- Conditionnelle:

```
{(p&&b)}S1{q}    {(p&&!b)}S2{q}
      |                   |
   {p} if b S1 else S2 {q}
```

LOGIQUE DE HOARE

RÈGLES DE DÉDUCTIONS (SUITE)

- Conséquence:

```
{p=>q}    {q}S{r}    {r=>s}
   |         |          |
      {p} S {s}
```

- Boucle (p: invariant de boucle):

```
{(p&&b)}S{p}
      |
{p} while b S {p&&!b}
```

- Exemple: prouver le triplet

```
{(val>=0)                                                   //pre_condition
{cpt=val;res=0; while(cpt>0) {res=res+val;cpt=cpt-1}}       //programme
{(res=val²)}                                                //post-condition
```

  - prouver le premier triplet: {(val>=0) {cpt=val;res=0} ??
  - trouver et pouver un invariant de boucle p pour le while

LOGIQUE DE HOARE

DÉMONSTRATION DU TRIPLET

```
{(VAL>=0)}

{CPT=VAL;RES=0; WHILE(CPT>0) {RES=RES+VAL;CPT=CPT-1}}

{(RES=VAL²)}
```

PREMIER TRIPLET: {(VAL>=0) {CPT=VAL;RES=0} ??

```
--------------------------------------/*axiome*/----------------------------------------
{(val>=0)}              cpt=val       {(cpt>=0)&&(cpt=val)}
--------------------------------------/*axiome*/----------------------------------------
{(cpt>=0)&&(cpt=val)}   res=0          {(cpt>=0)&&(cpt=val)&&(res=0)}
--------------------------------------/*composition*/-----------------------------------
{(val>=0)}              cpt=val;res=0  {(cpt>=0)&&(cpt=val)&&(res=0)}
```

LOGIQUE DE HOARE

DÉMONSTRATION DU TRIPLET(SUITE)

{(VAL>=0)}

{CPT=VAL;RES=0; WHILE(CPT>0) {RES=RES+VAL;CPT=CPT-1}}

{(RES=VAL²)}

INVARIANT DE BOUCLE P: (RES=VAL*VAL-VAL*CPT)&&(CPT>=0)

```
//verification en entrée de boucle:
-------------------------------------/*logique*/--------------------------------------
{(cpt>=0)&&(cpt=val)&&(res=0)} => {(res=val*val-val*cpt)&&(cpt>=0)}
-------------------------------------/*conséquence*/----------------------------------
{(val>=0)}  cpt=val;res=0        {(res=val*val-val*cpt)&&(cpt>=0)}
//vérification pour une itération:
-------------------------------------/*axiome*/---------------------------------------
{(res=val*val-val*cpt)&&(cpt>=0)}res=res+val{(res=val*val-val*cpt+val)&&(cpt>0)}
-------------------------------------/*conséquence*/----------------------------------
{(res=val*val-val*cpt)&&(cpt>=0)}res=res+val{(res=val*val-val*(cpt-1))&&(cpt>0)}
-------------------------------------/*axiome*/---------------------------------------
{(res=val*val-val*(cpt-1))&&(cpt>0)}cpt=cpt-1{(res=val*val-val*cpt))&&(cpt>=0)}
-------------------------------------/*composition*/----------------------------------
{(res=val*val-val*cpt)&&(cpt>=0)}res=res+val;cpt=cpt-1{(res=val*val-val*cpt))&&(cpt>=0)}
```

LOGIQUE DE HOARE

DÉMONSTRATION DU TRIPLET(FIN)

{(VAL>=0)}

{CPT=VAL;RES=0; WHILE(CPT>0) {RES=RES+VAL;CPT=CPT-1}}

{(RES=VAL²)}

SORTIE DE BOUCLE

```
-------------------------------------/*while*/-----------------------------------------
{(res=val*val-val*cpt)&&(cpt>=0)}
while(cpt>0){res=res+val;cpt=cpt-1}}
{(res=val*val-val*cpt)&&(cpt=0)}
-------------------------------------/*composition*/-----------------------------------
{(val>=0)}cpt=val;res=0;while(cpt>0)
{res=res+val;cpt=cpt-1}}
{(res=val*val-val*cpt)&&(cpt=0)}
-------------------------------------/*conséquence*/-----------------------------------
{(val>=0)}
cpt=val;res=0;while(cpt>0){res=res+val;cpt=cpt-1}}
{(res=val²}
```

LOGIQUE DE HOARE
COHÉRENCE DU SYSTÈME

- Le système présenté ci-dessus est *cohérent* (tout ce qu'il prouve est vrai)
- Il n'est pas *complet* : certaines formules vraies ne sont pas prouvables
- Il permet uniquement d'établir la *correction partielle* d'un programme (si le programme termine, alors le résultat produit est correct).
- Pour avoir la *correction totale*, il faut prouver sa terminaison.
    - Problème: question en générale indécidable (cf cours de décidabilité :-)
    - En pratique, il est souvent possible de prouver qu'un programme termine: *variant de boucle*

LOGIQUE DE HOARE

PREUVE DE TERMINAISON (CAS DES BOUCLES)

- Idée: Trouver une variable v entière appellée *variant de boucle* telle que:
  - v a une valeur positive avant la boucle et après chaque itération
  - v diminue à chaque itération

- Exemples:

```
static void loop1(int n) {
  //@ loop_variant = n ;
  while (n > 0) n--;
}

static void loop2(int n) {
  //@ loop_variant = 100-n ;
  while (n < 100) n++;
}
```