# Un deuxième rappel: comment prouver qu'un algorithme est correct

# PROUVER QU'UN PROGRAMME EST CORRECT???

*"Program testing can be used to show the presence of bugs, but never to show their absence !"* Edsger W. Dijkstra

*"One does not need to give a formal proof of an obviously correct program; but one needs a thorough understanding of formal proof methods to know when correctness is obvious."* John C. Reynolds

# PROUVER LA CORRECTION D'UN PROGRAMME,

c'est prouver qu'il correspond à la spécification donnée.

La Logique de Hoare-Floyd a été introduite à la fin des années 60 pour formaliser la relation entre langage de programmation (impératif) et langage de spécification.

Elle est basée sur la notion de *triplet de Hoare*, de la forme $\{P\}$ `A` $\{Q\}$;

$P$ est la précondition,

$Q$ la postcondition,

$A$ étant une expression du langage de programmation.

# SPÉCIFICATION

$$\{P\}\ \mathtt{A}\ \{Q\};$$

$P$ et $Q$ sont des assertions, i.e. des formules du langage de spécification décrivant les valeurs des variables, l'état de la mémoire ou du système.

En toute rigueur, le langage de spécification devrait être un langage mathématique comme la logique des prédicats, mais dans un objectif de documentation plus que de preuve de programmes, le langage de spécification peut être le langage naturel (en évitant les ambiguités!).

# INTERPRÉTATION:

L'interprétation de $\{P\}$ `A` $\{Q\}$ est:

si le système vérifie $\{P\}$, on peut assurer qu'il vérifie $\{Q\}$ après l'exécution de `A`;

en fait, c'est un peu plus compliqué que cela; en effet, il se peut que l'exécution de `A` ne termine pas! On a donc deux interprétations possibles:

si $\{P\}$ est vérifiée, après l'exécution de `A`, si celle-ci termine, on peut assurer $\{Q\}$;

si $\{P\}$ est vérifiée, la terminaison de `A` est assurée et après son exécution, on peut assurer $\{Q\}$;

Dans le premier cas, on parle de correction partielle.

Prouver un programme revient donc à prouver:

{Précondition sur les données} `Programme` {condition exprimant les résultats attendus}

à partir des axiomes et des règles d'inférences de Hoare qui montrent comment inférer la correction d'instructions composées à partir de celles de base.

# La règle pour la Séquence

$$\{P\}\ \texttt{A ; B}\ \{Q\}?$$

si

$$\{P\}\ \texttt{A}\ \{\mathit{Intermede}\}\ \text{et}\ \{\mathit{Intermede}\}\ \texttt{B}\ \{Q\}$$

alors:

$$\{P\}\ \texttt{A ; B}\ \{Q\}$$

# La règle de la conditionnelle

$$\{P\}\ \texttt{si cond alors A sinon B}\ \{Q\}\ ?$$

si

$$\{P \wedge cond\}\ \texttt{A}\ \{Q\} \text{ et } \{P \wedge \neg cond\}\ \texttt{B}\ \{Q\}$$

alors:

$$\{P\}\ \texttt{si cond alors A sinon B}\ \{Q\}$$

# LA RÈGLE POUR LE "TANT QUE'

$$\{P\}\ \texttt{tant que cond faire A}\ \{Q\}??$$

si on trouve une assertion $i$, qu'on appelle *invariant*, telle que:
$P \Rightarrow i$

$\{i \wedge cond\}$ `A` $\{i\}$

$\{i \wedge \neg cond\} \Rightarrow \{Q\}$

alors:

$\{P\}\ \texttt{tant que cond faire A}\ \{Q\}$

Attention: pour avoir une preuve de correction totale, il faut aussi faire une preuve d'arrêt de la boucle!

# L'AFFECTATION

$P[x \leftarrow exp]$ `x:=exp` $P$

en supposant que exp ne contient pas de fonctions à effet de bord.

Par exemple, $x + y < 5$ `x:=x+y` $x < 5$

# Le renforcement de la précondition

si

$$P \Rightarrow Q$$

et

$$\{Q\}\ \mathtt{A}\ \{R\}$$

alors

$$\{P\}\ \mathtt{A}\ \{R\}$$

# L'AFFAIBLISSEMENT DE LA POSTCONDITION

si

$$\{P\}\ \mathtt{A}\ \{Q\}$$

et

$$Q \Rightarrow R$$

alors

$$\{P\}\ \mathtt{A}\ \{R\}$$

Soit: Qui peut le plus, peut le moins!

# UN EXEMPLE: LA MULTIPLICATION À LA RUSSE

```
\\ x >=y >=0
int Mult (int x, int y)
int R=0;
int a=x, b=y;
while (b>0){
   if ( b %2 =1) R=R+a;
   a=2*a;
   b=  b/2; }
\\ R= a*b
```

# UN EXEMPLE: LA MULTIPLICATION À LA RUSSE

```
\\ x >=y >=0
int Mult (int x, int y)
int R=0;
int a=x, b=y;
while (b>0){
   if ( b %2 =1) R=R+a;
   a=2*a;
   b=  b/2; }
\\ R= a*b?
```

# UN EXEMPLE: LA MULTIPLICATION À LA RUSSE

```
\\ x >=y >=0
int Mult (int x, int y)
int R=0;
int a=x, b=y;
\\ R=0, a=x, b=y
while (b>0){
   if ( b %2 =1) R=R+a;
   a=2*a;
   b=  b/2; }
\\ R= a*b?
```

# UN EXEMPLE: LA MULTIPLICATION À LA RUSSE

```
\\ x >=y >=0
int Mult (int x, int y)
int R=0;
int a=x, b=y;
\\ R=0, a=x, b=y
while (b>0){\\invariant R+a*b=x*y
   if ( b %2 =1) R=R+a;
   a=2*a;
   b=  b/2; }
\\ R= a*b
```

# REMARQUE: VERSION RÉCURSIVE

```
\\ x >=y >=0
int Mult (int x, int y)
 if y==0 retrun 0
else return  Mult(2*x, y / 2) +x* y % 2;
```

Preuve par induction sur y!