Validation et vérification du logiciel - Test et preuve

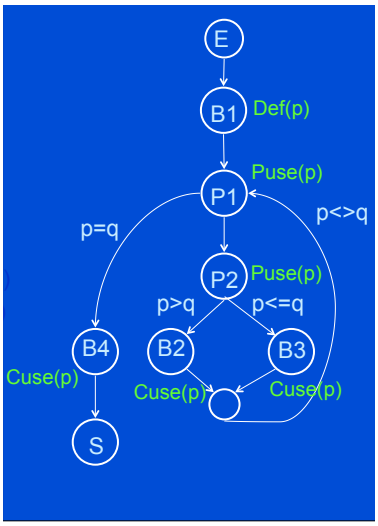
Exercice 1 (**Test structurel**)

Soit la procédure suivante qui calcule le pgcd de deux entiers positifs non nuls.

```
int pgcd(int p, int q)
 while p<>q do
   if p > q
   then p := p-q
   else q:= q-p
   end -- if
 end -- while
 return p
end-- pgcd
```

1. Donnez les graphes de flot de contrôle et de flot de données de cette fonction.

   Correction :



   Le nœud B1 représente l'initialisation des variables p et q à partir des arguments de l'appel.

   Ce graphe de flot de contrôle est dans le cours (avec une instruction read).

   Correction :

   Les chemins E, B1, P1, P2, B2, P1, B4, S (ch1) et E, B1, P1, P2, B3, P1 B4, S (ch2) satisfont le critère.

   Calculons le prédicat de chemin du premier chemin :

$$p > 0 \wedge q > 0 \wedge p <> q \wedge p > q \wedge p - q = q$$

   Un jeu de test qui convient est p=4, q=2 (résultat attendu = 2).

Calculons le prédicat de chemin du deuxième chemin :

$$p > 0 \wedge q > 0 \wedge p <> q \wedge p \leq q \wedge p = q - p$$

Un jeu de test qui convient est p=2, q=4 (résultat attendu = 2).

Remarque : un seul chemin (plus long) peut suffire : E B1 P1 P2 B2 P1 P2 B3 P1 B4 S.

2. Donner une suite de tests qui couvre le critère *toutes les définitions de* `p` *et* `q`.

   Correction :

   - Pour chaque définition de `p`, couvrir au moins une utilisation de p.

   Définition de p en B1 : on couvre l'utilisation de p en P1 (sans avoir de redéfinition de p entre les deux). Le chemin ch1 convient.

   Définition de p en B2 : on couvre l'utilisation de p en B4 (sans avoir de redéfinition de p entre les deux). Le chemin E, B1, P1, P2, B2, P1, B4, S (ch1) convient.

   Remarque : ce chemin couvre aussi la paire (définition en B2 , utilisation en p1).

   - Pour chaque définition de `q`, couvrir au moins une utilisation de q.

   Définition de q en B1 : on couvre l'utilisation de q en P1 (sans avoir de redéfinition de q entre les deux). E, B1, P1, P2, B3, P1, B4, S (ch2) convient.

   Définition de q en B3 : on couvre l'utilisation de q en P1 (sans avoir de redéfinition de q entre les deux). Le chemin E, B1, P1, P2, B3, P1, B4, S (ch2) convient.

   Pour couvrir le critère, il suffit de prendre les chemins ch1 et ch2. Donc les jeux de test : p=4, q=2 et p=2, q=4 par exemple.

3. Donner une suite de tests qui couvre le critère *toutes les utilisations de* `p`.

   Correction : le critère est plus exigeant. Il s'agit cette fois pour toute définition de p de couvrir toutes les définitions de p.

   Rappel de la définition : L'ensemble de chemins T couvre le critère toutes les utilisations si pour toute variable x, pour chaque définition d_x de x en B, pour chaque utilisation de u_x en B' que le définition d_x atteint, il existe un chemin de T qui contient BCB' où C est un chemin sans redéfinition de x.

B	B'	*chemin*
$B1$	$P1$	$ch4$
$B1$	$P2$	$ch4$
$B1$	$B2$	$ch5$
$B1$	$B3$	$ch4$
$B1$	$B4$	$ch2$
$B2$	$P1$	$ch4$
$B2$	$P2$	$E, B1, P1, P2, B2, P1, P2, B3, P1, B4, S(ch4)$
$B2$	$B2$	$E, B1, P1, P2, B2, P1, P2, B2, P1, B4, S(ch5)$
$B2$	$B3$	$ch4$
$B2$	$B4$	$ch5$

Il suffit donc de 3 chemins (ch2, ch4, ch5) donc 3 DTs.
Calculons le prédicat de chemin pour ch4 :

$$p > 0 \wedge q > 0 \wedge p \neq q \wedge p > q \wedge p - q \neq q \wedge p - q \leq q \wedge p - q = q - (p - q)$$

ou encore

$$p > 0 \wedge q > 0 \wedge p \neq q \wedge p > q \wedge p - q \neq q \wedge p - q \leq q \wedge 2p = 3q$$

Pour couvrir ce chemin, on peut choisir DT4 : p=3, q=2, résultat attendu = 1

Calculons le prédicat de chemin pour ch5 :

$$p > 0 \wedge q > 0 \wedge p \neq q \wedge p > q \wedge p - q \neq q \wedge p - q > q \wedge p - 2q = q$$

ou encore

$$p > 0 \wedge q > 0 \wedge p \neq q \wedge p > q \wedge p - q \neq q \wedge p - q > q \wedge p = 3q$$

DT4 : p=9, q=3, résultat attendu = 3

4. Les chemins précédents couvrent-ils le critère *tous les DU chemins pour la variable* `p` ? Si non, complétez.
   Correction : il s'agit ici de couvrir tous les chemins simples (sans circuit ou élémentaire) pour chaque couple (d_p, u_p).
   Pour la définition en B1 et l'utilisation en B4, on a emprunté à la question précédente le chemin élémentaire (ch2) mais on doit aussi couvrir le chemin élémentaire E, B1, P1, B4, S (ch3) qui a pour prédicat de chemin $p > 0 \wedge q > 0 \wedge p = q$. La DT p=5, q=5 convint pour exercer ce chemin.

Exercice 2

Soit le programme suivant

```
a := a + b; b := a - 2*b; a := a * b;
```

Montrer que le triplet de Hoare suivant est valide.

$$\{a = x \wedge b = y\} Prog \{a = x^2 - y^2\}$$

x et y modélisent ici les valeurs initiales des variables `a` et `b`.

Correction : Faire le calcul en remontant pour calculer la plus faible précondition puis montrer que la pré-condition implique cette plus faible précondition. Dans ce premier exo, on ne parlera pas cependant de plus faible précondition.

Exercice 3

Pour chaque triplet ci-dessous donnez une formule logique f qui rend le triplet valide. Vous déterminerez la formule la plus *simple* (dans quel sens ?).

— $\{f\}x := x + 2\{x = 5\}$
— $\{f\}x := y\{x = y\}$
— $\{f\}if\ x > 0\ then\ x := x + 1\ else\ x := -x\ fi\{x > 1\}$

Correction : La plus simple signifie ici la plus faible. C'est-à-dire la formule la moins contraignante.
— $\{x + 2 = 5\}x := x + 2\{x = 5\}$
— $\{true\}x := y\{x = y\}$
— $\{x > 0 \vee x < -1\}if\ x > 0\ then\ x := x + 1\ else\ x := -x\ fi\{x > 1\}$

Exercice 4

Prouver que le triplet de Hoare ci-dessous est valide.

$$\{x \geq 0\}if\ x >= 0\ then\ y := 8\ else\ y := 9\ fi\{y = 8\}$$

Correction : Idem méthode en 2 temps : calcul de la plus faible précondiiton W et on démontre que $x \geq 0$ implique W .

Exercice 5 (**Logique de Hoare**)

On considère le programme $Prog$ suivant :

```
while y ≠ x do
   x:=x-1; y:=y-2;
end;
```

Question 1

Démontrer que le triplet de Hoare $\{x \leq y\}Prog\{x = y\}$ est valide.

Correction : Notons I la propriété et C le corps de la boucle. Pour montrer que I est un invariant, il faut démontrer le triplet de Hoare suivant $\{y \neq x\ \wedge\ I\}C\{I\}$.

Question 2

On s'intéresse maintenant à la correction totale du programme $Prog$. Que faut-il démontrer de plus ?

Correction : Il faut démontrer la terminaison de la boucle. On propose un variant. Prendre pour variant l'expression $y - x$.
Il nous faut donc démontrer la validité du triplet $\{I \wedge e \wedge V \geq 0 \wedge V = n\}C\{V \geq 0 \wedge V < n\}$.
Comme C ne contient pas de boucle, correction totale et correction partielle sont identiques.
On suit la méthode déjà suivie pour montrer le triplet : on calcule la plus faible précondition de C par rapport à $V \geq 0\ \wedge\ V < n$, soit W. Puis on montrera que la formule $I \wedge e \wedge V \geq 0 \wedge V = n \Rightarrow W$.

Exercice 6

Soit `P` le programme suivant

```
y1 := 0;  y2 := 1;  y3 := 1;
while (y3 <= x) do
    y1 := y1 + 1;
    y2 := y2 + 2;
    y3 := y3 + y2
od
```

Dans la suite `S` désigne le corps de la boucle et `S0` la séquence formée des 3 premières instructions.

Nous allons montrer avec la logique de Hoare que ce programme calcule la racine carrée entière `y1` de `x` si `x` est un entier positif ou nul.

Question 3

Ecrire la spécification du programme à l'aide d'une pré et d'une post condition (contrat).

Correction : Il s'agit de démontrer le triplet de Hoare suivant

$$\{x \geq 0\} P \{(y1 * y1 \leq x) \wedge ((y1+1) * (y1+1) > x)\}$$

Question 4

Proposer un invariant I pour la boucle. Pour cela calculer les valeurs que prennent les variables `y1`, `y2` et `y3` pour les premières itérations de la boucle et à partir de là essayer de calculer la valeur de `y2` et `y3` en fonction de `y1`. Attention, pensez également que l'invariant devra vous aider à prouver la postcondition du programme.

Correction : On propose pour I la formule

$$(y2 = 2 * y1 + 1) \wedge (y3 = (y1+1) * (y1+1)) \wedge (y1 * y1 \leq x)$$

Question 5

Démontrer formellement que I est un invariant pour la boucle.

Correction : Il faut pour cela démontrer le triplet $\{I \wedge y3 \leq x\} S \{I\}$.

On procède pour cela en remontant S :

$(y2 + 2 = 2 * (y1+1) + 1) \wedge (y3 + y2 + 2 = ((y1+1) + 1) * ((y1+1) + 1)) \wedge ((y1+1) * (y1+1) \leq x)$ `(f)`

```
    y1 := y1 + 1;
```

$(y2 + 2 = 2 * y1 + 1) \wedge (y3 + y2 + 2 = (y1 + 1) * (y1 + 1)) \wedge (y1 * y1 \leq x)$

```
    y2 := y2 + 2;
```

$(y2 = 2 * y1 + 1) \wedge (y3 + y2 = (y1 + 1) * (y1 + 1)) \wedge (y1 * y1 \leq x)$

```
    y3 := y3 + y2
```

$(y2 = 2 * y1 + 1) \wedge (y3 = (y1 + 1) * (y1 + 1)) \wedge (y1 * y1 \leq x)$

il nous reste à montrer que $(I \wedge y3 \leq x) \Rightarrow f$. Commençons par réécrire f :

$(y2 = 2 * y1 + 1) \wedge (y3 + y2 + 2 = (y1 + 1 + 1) * (y1 + 1 + 1) \wedge (y1 + 1) * (y1 + 1) \leq x$

On peut encore mettre f sous la forme : $(y2 = 2 * y1 + 1) \wedge (y3 + y2 = (y1 + 1) * (y1 + 1) + 2 * y1 + 1 \wedge (y1 + 1) * (y1 + 1) \leq x$.
On a alors $(I \wedge y3 \leq x) \Rightarrow f$.

Question 6

Montrer que $\{x \geq 0\}S0\{I\}$ est valide.

Correction : de la même façon on a :

$\{(1 = 2 * 0 + 1) \wedge (1 = (0 + 1) * (0 + 1)) \wedge (0 * 0 \leq x)\}$ `(g)`
`y1 := 0;`
$\{(1 = 2 * y1 + 1) \wedge (1 = (y1 + 1) * (y1 + 1)) \wedge (y1 * y1 \leq x)\}$
`y2 := 1;`
$\{(y2 = 2 * y1 + 1) \wedge (1 = (y1 + 1) * (y1 + 1)) \wedge (y1 * y1 \leq x)\}$
`y3 := 1;`
$\{(y2 = 2 * y1 + 1) \wedge (y3 = (y1 + 1) * (y1 + 1)) \wedge (y1 * y1 \leq x)\}$

g se simplifie en $1 = 1 \wedge 1 = 1$ $0 \leq x$. Et puisque l'on a $0 \leq x \Rightarrow g$ on en déduit le triplet recherché.

Question 7

Conclure.

Correction : la correction partielle du programme initial s'obtint grâce aux triplets démontrés dans les questions précédentes. En effet puisque I est un invariant de la boucle, on peut en déduire qu'à la fin de la sortie de la boucle, l'invariant et la négation de la condition sont vrais, donc le triplet $\{I\}while\ y3 \leq x\ do\ S\ end\{I \wedge y3 > x\}$ est valide. En revenant à ce que veut dire un triplet en termes de sémantique et d'exécution, et en combinant avec le triplet concernant S0, on en déduit que $\{x \geq 0\}P\{I \wedge y3 > x\}$ est un triplet valide.
On remarque que $I \wedge y3 > x$ implique $(y1 * y1 \leq x) \wedge ((y1 + 1) * (y1 + 1) > x)$, la postcondition recherchée. Ainsi si un état mémoire satisfait $I \wedge y3 > x$, il satisfait aussi la propriété $Q = (y1 * y1 \leq x) \wedge ((y1 + 1) * (y1 + 1) > x)$. Comme on a démontré que $\{x \geq 0\}P\{I \wedge y3 > x\}$ est un triplet valide, alors on en déduit que le triplet $\{x \geq 0\}P\{Q\}$ est un triplet valide, cqfd.