LOGIQUE DE HOARE

En 1969, Charles Anthony Richard Hoare[1] a inventé une logique propositionnelle spécialisée dans l'algorithmique, destinée à démontrer qu'un algorithme donné fait réellement ce qu'il est censé faire.
À partir de l'algorithme, l'utilisation de la logique de Hoare permet d'avoir une ***preuve de programme***, c'est-à-dire une démonstration de la correction du programme.
Mais on s'est rendu compte que la même méthode, appliquée à une feuille blanche, permet de ***construire*** l'algorithme. Cette méthode de programmation a été longtemps défendue par Jacques Arsac en France, et peut donc être considérée comme placée à l'origine de l'enseignement français de l'algorithmique.

I/ Notations

1) JavaScript

Dans la suite, les exemples seront écrits en JavaScript, contrairement au choix de Hoare, et les instructions écrites en rouge. JavaScript n'impose pas que les variables soient déclarées au début du script, et ne leur impose pas de type non plus. Aussi les exemples vus ci-dessous seront-ils choisis entiers, comme dans l'article de Hoare.

2) Logique

Les propositions logiques seront écrites en bleu. La négation de p est notée $\neg p$, la conjonction de p et q est notée $p \wedge q$, leur disjonction est notée $p \vee q$ et la proposition $\neg p \vee q$ est notée $p \Rightarrow q$. On note $\top$ une tautologie (comme par exemple $x = x$) et $\bot$ une antilogie (comme $2 + 2 = 5$).

3) Arithmétique

Les premiers axiomes de la logique de Hoare, ce sont les axiomes de la logique comme $p \wedge q \Leftrightarrow q \wedge p$ ou $p \Rightarrow p$. On retiendra notamment le tiers exclu $p \vee \neg p$ et l'axiome du modus ponens $((p \Rightarrow q) \wedge (q \Rightarrow r)) \Rightarrow (p \Rightarrow r)$.
Mais on considère également comme axiome, tout théorème de l'arithmétique comme $\forall n \in \mathbb{N}, 0 \leqslant n$ ou $\forall a, b \in \mathbb{N},\ a + b = b + a$. Ces théorèmes sont des conséquences des ***axiomes de Peano*** mais dans le cas présent, on ne s'intéresse pas à la raison pour laquelle tout entier naturel peut se décomposer en facteurs premiers, on s'intéresse à l'utilité de ladite décomposition, ou du moins de son existence.

II/ Affectation et égalité

1) Programme inutile

Dans la logique de Hoare, la valeur de vérité de propositions p, q etc. dépend de l'endroit où on en est dans l'exécution d'un programme P. Si p est vraie avant l'exécution de P et si, après l'exécution de P, c'est q qui est vraie, on note

$$p\,\{P\}\,q$$

L'affectation se note en JavaScript par le signe `=` ce qui risque de prêter à confusion avec la vraie égalité, elle aussi notée =... L'instruction JavaScript `x=x ;` ne fait rien (à part perdre du temps), puisqu'elle affecte à la variable x, sa propre valeur. Il lui correspond le schéma d'axiomes suivant :

$$p\,\{\texttt{x=x ;}\}\,p$$

Par exemple $y = 2\,\{\texttt{x=x ;}\}\,y = 2$: Si $y = 2$ avant le programme qui ne fait rien (en tout cas qui ne touche pas à y), alors $y = 2$ après aussi (puisqu'il ne s'est rien passé).

2) Affectation

On a aussi le schéma d'axiomes suivant :

$$p_e\,\{\texttt{x=e ;}\}\,p$$

[1] inventeur de l'algorithme de tri "Quicksort" utilisé notamment dans JavaScript, et célèbre pour la citation "Il y a deux manières pour construire un logiciel : Ou bien on le fait si simple qu'il n'y a à l'évidence pas de défauts, ou bien on le fait si compliqué qu'aucun défaut n'est évident. La première méthode est de loin la plus difficile."

où p_e désigne la proposition obtenue en remplaçant dans p, chaque occurrence de x par l'expression e. On a l'impression de faire les choses à l'envers puisque on remplace x par e avant l'exécution du programme qui fait ce remplacement. Mais voici des exemples :

$$2 \geqslant 0 \,\{\texttt{x=2 ;}\}\, x \geqslant 0$$

Effectivement, dans la mesure où 2 est positif, si on remplace x par 2, après le remplacement, x sera positif (il sera aussi pair et premier, pour des raisons analogues). Dans ce cas l'écriture de $2 \geqslant 0$ est optionnelle puisque cette proposition est une tautologie. On peut donc aussi écrire $\{\texttt{x=2 ;}\}\, x \geqslant 0$.

$$x + 1 = 3 \,\{\texttt{x=x+1 ;}\}\, x = 3$$

Si $x = 2$, après incrémentation de x, celui-ci devient égal à 3.

$$x + 1 \geqslant 0 \,\{\texttt{x=x+1 ;}\}\, x \geqslant 0$$

Si $x \geqslant -1$, après incrémentation de x, celui-ci sera positif.

3) Implication

On a implicitement utilisé ci-dessus le fait que si $p \Rightarrow q$ et $q\,\{P\}\,r$ alors $p\,\{P\}\,r$. Ce n'est pas un axiome mais une règle de déduction, appelée règle de la précondition.
La règle de la postcondition stipule que si $p\,\{P\}\,q$ et $q \Rightarrow r$ alors $p\,\{P\}\,r$.

4) Conjonction et disjonction

Encore deux règles de déduction :
Si $p\,\{P\}\,q$ et $p\,\{P\}\,r$ alors $p\,\{P\}\,q \wedge r$.
Et si $p\,\{P\}\,r$ et $q\,\{P\}\,r$ alors $p \vee q\,\{P\}\,r$.
Exemples : De

$$x \in \mathbb{Z} \,\{\texttt{x=x*x ;}\}\, x \in \mathbb{Z}$$

(le carré d'un entier est entier), et de

$$x \in \mathbb{Z} \,\{\texttt{x=x*x ;}\}\, x \geqslant 0$$

on déduit par la règle de la conjonction

$$x \in \mathbb{Z} \,\{\texttt{x=x*x ;}\}\, x \in \mathbb{N}$$

Des exemples un peu plus intéressants seront vus plus bas...
De

$$x \geqslant 3 \,\{\texttt{x=x*x ;}\}\, x \geqslant 9$$

et

$$x \leqslant -3 \,\{\texttt{x=x*x ;}\}\, x \geqslant 9$$

on déduit

$$x \leqslant -3 \vee x \geqslant 3 \,\{\texttt{x=x*x ;}\}\, x \geqslant 9$$

III/ Instructions en séquence

1) Régle de déduction

De $p\,\{P\}\,q$ et $q\,\{Q\}\,r$ on déduit $p\,\{P;Q\}\,r$. C'est cette règle de déduction qui exprime que l'exécution de Q a lieu après celle de P. Et la déduction se fait dans le même ordre. Avec ça on peut faire des preuves de programmes écrits sans boucles ni tests. Quelques exemples :

2) Affectations multiples

```
x=2 ;
x=x+1 ;
x=x*x ;
```

La question est "que contient la variable x à la fin de l'exécution de ce script ? Si on appelle c la valeur cherchée, on a (en remontant de droite à gauche et de bas en haut) :

$$(2+1)^2 = c \,\{\texttt{x=2 ;}\}\, (x+1)^2 = c$$

$$(x+1)^2 = c \,\{\texttt{x=x+1 ;}\}\, x^2 = c$$

$$x^2 = c \,\{\texttt{x=x*x ;}\}\, x = c$$

d'où on tire $c = (2+1)^2 = 3^2 = 9$.
Un autre exemple, donné en contrôle en Seconde, et jugé difficile par les élèves (parce qu'il y a deux variables) :

```
x=8 ;
y=2*x-1 ;
x=y+2 ;
y=x-1 ;
```

La question est "que contiennent les deux variables x et y à la fin de l'exécution du script ?". Si on appelle respectivement a et b ces deux valeurs, une preuve du programme (toujours établie de bas en haut et de droite à gauche) donne ceci :

$$(2\times 8 = a-1) \wedge (2\times 8 = b) \,\{\texttt{x=8 ;}\}\, (2x = a-1) \wedge (2x = b)$$

$$(2x-1 = a-2) \wedge (2x-1 = b-1) \,\{\texttt{y=2*x-1 ;}\}\, (y = a-2) \wedge (y = b-1)$$

$$(y+2 = a) \wedge (y+2 = b+1) \,\{\texttt{x=y+2 ;}\}\, (x = a) \wedge (x = b+1)$$

$$(x = a) \wedge (x-1 = b) \,\{\texttt{y=x-1 ;}\}\, (x = a) \wedge (y = b)$$

qui donne par résolution des deux équations $16 = a-1$ et $16 = b$ les deux valeurs $a = 17$ et $b = 16$.

3) Exercice donné en contrôle commun

```
A=1 ;
B=2 ;
A=A+B ;
B=A-2*B ;
A=A*B ;
```

Le but de l'exercice était de vérifier que si, au début, $A = x$ et $B = y$ (indépendamment des deux premières lignes) alors, à la fin, $A = x^2 - y^2$. Comme dans l'exemple suivant, un corrigé sous forme d'algorithme de calcul formel (donc algébrique avec développement d'un produit remarquable) peut être fait (et était attendu dans le contrôle commun). Mais la méthode de Hoare peut aussi le démontrer, et sans réellement changer de cadre (là encore, la preuve est établie de bas en haut et de droite à gauche) :

$$1 = d+4 \,\{\texttt{A=1 ;}\}\, A^2 = d+4$$

$$A^2-4=d\,\{\texttt{B=2 ;}\}\,A^2-B^2=d$$

$$(A+B)(A+B-2B)=d\,\{\texttt{A=A+B ;}\}\,A(A-2B)=d$$

$$A(A-2B)=d\,\{\texttt{B=A-2*B ;}\}\,AB=d$$

$$AB=d\,\{\texttt{A=A*B ;}\}\,A=d$$

À la fin, la première ligne fournit la valeur finale $d=3$ demandée, mais aussi à la troisième ligne la démonstration de $d=x^2-y^2$. On remarque que la propriété $d=x^2-y^2$ a été démontrée sans être conjecturée. C'est toute la puissance de la méthode de Hoare, et c'est de là que vient la possibilité de partir de la preuve pour construire l'algorithme.

4) Sortie constante

L'exemple suivant est un classique (sous forme de "programme de calcul" à la fin du XXe siècle en collège) :

```
x=Math.round(Math.random()*100);
y=x-1;
z=x+1;
t=y*z-Math.pow(x,2);
```

Il consiste, pour un entier quelconque x, à calculer l'expression $(x-1)(x+1)-x^2$. L'objet de l'exercice n'est pas tant de trouver ce que fait ce programme (il suffit de le simuler sur quelques valeurs de x) mais de prouver qu'il donne toujours -1 en sortie. La preuve "classique" consiste à développer et réduire $(x-1)(x+1)-x^2$ pour vérifier que cette fonction est constante, mais elle se solde par un double changement de cadre, d'abord de l'algorithmique à l'algèbre, puis de l'algèbre aux fonctions.

Ici par contre, encore une fois, on va chercher à éviter d'émettre une conjecture sur la valeur c de t :

$$(x-1)(x+1)-x^2=d\,\{\texttt{y=x-1 ;}\}\,y(x+1)-x^2=d$$

$$y(x+1)-x^2=d\,\{\texttt{z=x+1 ;}\}\,yz-x^2=d$$

$$yz-x^2=d\,\{\texttt{t=y*z-Math.pow(x,2) ;}\}\,t=d$$

Cependant là encore il faut développer $(x-1)(x+1)-x^2$ pour connaître la valeur de d.

Si maintenant on a conjecturé que $d=-1$, c'est une vraie démonstration qu'on obtient :

$$(x-1)(x+1)=x^2-1\,\{\texttt{y=x-1 ;}\}\,y(x+1)=x^2-1$$

$$y(x+1)-x^2=-1\,\{\texttt{z=x+1 ;}\}\,yz-x^2=-1$$

$$yz-x^2=-1\,\{\texttt{t=y*z-Math.pow(x,2) ;}\}\,t=-1$$

La preuve que $d=-1$ vient de ce qu'on trouve une tautologie $(x-1)(x+1)=x^2-1$ à la fin (cette façon de travailler "à l'envers" n'est pas la meilleure manière d'éviter la confusion entre théorème et réciproque !).

IV/ Tests

1) Règle des tests

On suppose que r est une proposition (un booléen en JavaScript). Alors de $r \wedge p\,\{P\}\,q$ et $\neg r \wedge p\,\{Q\}\,q$ on peut déduire

$$p\,\{\texttt{if(}r\texttt{)}\,\{P\}\,\texttt{else}\,\{Q\}\}\,q$$

En particulier, de $r \wedge p\,\{P\}\,q$ on peut déduire $p\,\{\texttt{if(}r\texttt{)}\,\{P\}\}\,q$. Cette dernière règle de déduction peut être considérée comme une version algorithmique du *modus ponens*.

2) Valeur absolue

La fonction "valeur absolue" n'est plus au programme de Seconde, et en Première S elle semble plus considérée comme une expression que comme une fonction. Mais si on définit ladite fonction par intervalles, avec

$$|x| = \begin{cases} x \text{ si } x \geqslant 0 \\ -x \text{ sinon} \end{cases}$$

on peut utiliser l'algorithmique pour conjecturer le signe de $|x|$ et la logique de Hoare pour le démontrer, avec l'algorithme consistant à choisir un nombre aléatoire pour x (compris entre $-\frac{1}{2}$ et $\frac{1}{2}$), calculer y avec la formule ci-dessus, et regarder le signe de y :

```
x=Math.random()-0.5 ;
if(x>=0){
        y=x ;
} else {
        y=-x ;
}
```

On a

$$x \geqslant 0\,\{\texttt{y=x ;}\}\,y \geqslant 0$$

d'après l'axiome d'affectation, mais aussi

$$-x \geqslant 0\,\{\texttt{y=-x ;}\}\,y \geqslant 0$$

soit

$$x < 0\,\{\texttt{y=-x ;}\}\,y \geqslant 0$$

De

$$x \geqslant 0\,\{\texttt{y=x ;}\}\,y \geqslant 0$$

et

$$x < 0\,\{\texttt{y=-x ;}\}\,y \geqslant 0$$

on déduit alors

$$\top\,\{\texttt{if(x>=0)\{ y=-x ;\}else\{ y=-x ;\}}\}\,y \geqslant 0$$

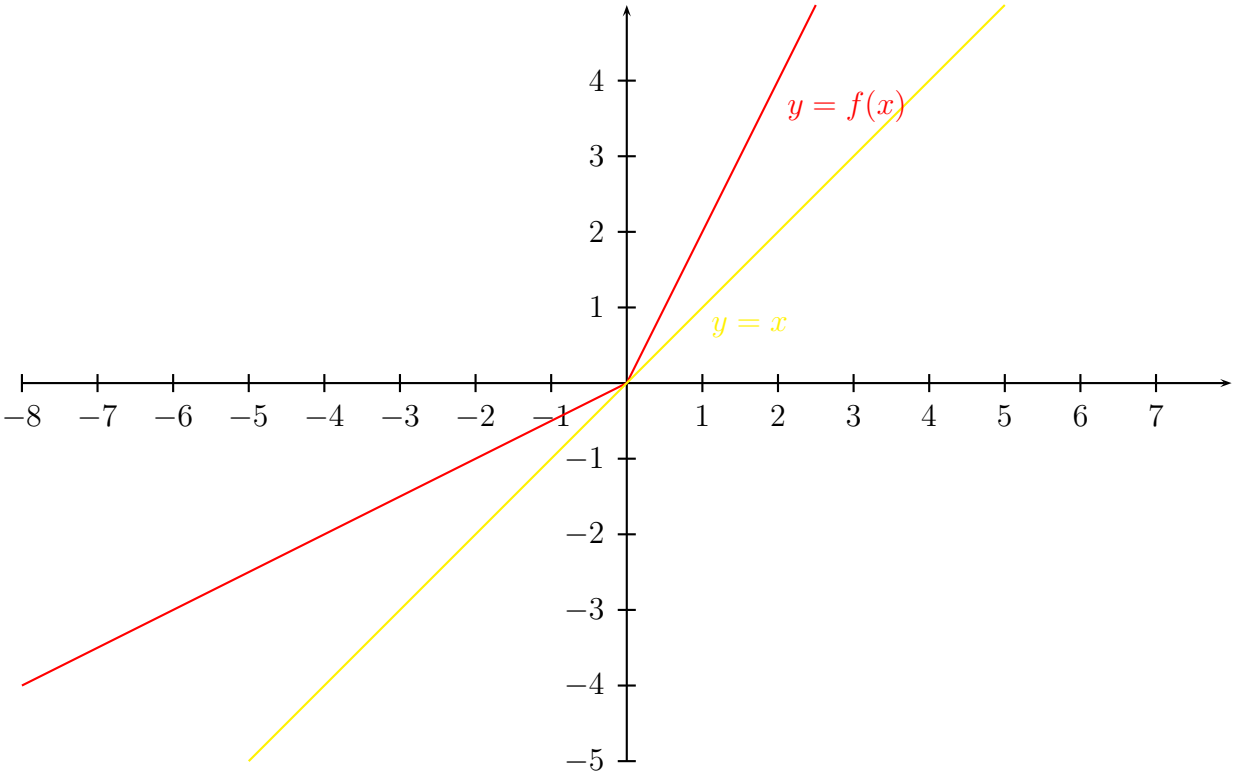
soit $\forall x \in \mathbb{R}, |x| \geqslant 0$.

3) Exercice

Soit $f(x)$ la fonction affine par intervalles définie comme suit :

$$f(x) = \begin{cases} 2x \text{ si } x \geqslant 0 \\ \dfrac{x}{2} \text{ sinon} \end{cases}$$

et représentée ci-dessous dans un repère orthonormé, en rouge :



Sur le même graphique, on a représenté en jaune la fonction $y = x$, ce qui mène à la conjecture suivante : $\forall x \in \mathbb{R}, f(x) \geqslant x$. Démontrer cette conjecture à l'aide de la logique de Hoare, sur le programme suivant :

```
x=Math.random()-0.5 ;
if(x>=0){
        y=2*x ;
} else {
        y=x/2 ;
}
```

V/ Boucles

1) Régle des boucles

Hoare ne donne qu'une règle de déduction, concernant les boucles "tant que" :
De $r \wedge p \{P\} p$ on peut déduire $p \{\texttt{while(}r\texttt{)}\{P\}\} \neg r \wedge p$.
Une proposition telle que p est appelée un *invariant de boucle*.

2) Et les autres boucles ?

Si Hoare n'a rien proposé pour les boucles "for to", c'est parce que c'est plus difficile à formaliser, mais surtout parce que c'est inutile : Une boucle dans laquelle l'indice i va de a jusqu'à b par pas de h, qui en JavaScript s'écrit

```
for(i=a ;i<=b ;i=i+h){
...
}
```

peut s'écrire sous cette forme :

```
i=a ;
while(i<=b){
...
i=i+h ;
}
```

3) Exemple

Pour calculer la somme des 10 premiers entiers par une boucle, on invente une variable s (comme

"somme") qu'on initialise à zéro, puis dans une boucle, on additionne ***successivement*** les valeurs de l'indice i à la somme courante ce qui en JavaScript se rédige ainsi :

```
s=0 ;
for(i=1 ;i<=10 ;i=i+1){
        s=s+i ;
}
```

mais aussi

```
s=0 ;
i=0 ;
while(i<=10){
        s=s+i ;
        i=i+1 ;
}
```

L'invariant de boucle qui permet d'avoir rapidement (rapidement ça veut dire plus vite qu'en calculant 1+2+3+4+5+6+7+8+9+10) la valeur finale de s est $s = \frac{i(i-1)}{2}$. En le démontrant on a la réponse puisqu'on connaît la valeur de i grâce à la condition de sortie de la boucle ($i = 11$). La démonstration se fait en deux temps :

a) D'abord on montre que $s = \frac{i(i-1)}{2}$ est effectivement un invariant de boucle, à l'aide de la règle de l'affectation ;

b) Puis on utilise la règle des boucles donnée au (1) ci-dessus, pour en déduire la valeur finale de s.

Ça donne ceci :

a) Toujours de droite à gauche et de bas en haut :

$$s + i = \frac{i(i+1)}{2} \{\texttt{s=s+i ;}\} s = \frac{i(i+1)}{2}$$

$$s = \frac{(i+1)(i+1-1)}{2} \{\texttt{i=i+1 ;}\} s = \frac{i(i-1)}{2}$$

Et comme $s + i = \frac{i(i+1)}{2} \Leftrightarrow s = \frac{i(i+1)}{2} - i = \frac{i(i+1)-2i}{2} = \frac{i(i-1)}{2}$, on a bien en $s = \frac{i(i-1)}{2}$ un invariant de boucle :

$$s = \frac{i(i-1)}{2} \{\texttt{s=s+i ;i=i+1 ;}\} s = \frac{i(i-1)}{2}$$

b) Finalement

$$(i \leqslant 10) \wedge \left(s = \frac{i(i-1)}{2}\right) \{\texttt{while(i<=10)\{s=s+i ;i=i+1 ;\}}\} (i = 11) \wedge \left(s = \frac{i(i-1)}{2}\right)$$

Pour finir, toujours de droite à gauche puis de bas en haut,

$$\top \wedge 0 = 0 \{\texttt{s=0 ;}\} \top \wedge s = 0$$

$$(0 \leqslant 10) \wedge \left(s = \frac{0(-1)}{2}\right) \{\texttt{i=0 ;}\} (i \leqslant 10) \wedge \left(s = \frac{i(i-1)}{2}\right)$$

(l'invariant de boucle était déjà vrai avant d'entrer dans la boucle)

On a prouvé ci-dessus qu'à la sortie de la boucle, $i = 11$ (condition de sortie) et $s = \frac{i(i-1)}{2}$ ce qui donne la valeur finale $s = \frac{11 \times 10}{2} = 55$.
On remarque l'extraordinaire ressemblance entre le raisonnement ci-dessus et une démonstration par récurrence, ressemblance qui n'a rien de fortuit. Ceci dit, la démonstration par récurrence est nécessaire pour les démonstrations abordées en Terminale S, car la conclusion de la démonstration par récurrence est une conjonction infinie de propositions (pour 0, pour 1, pour 2 etc.) alors qu'ici un tableau des valeurs successives de s et de $\frac{i(i-1)}{2}$ suffit comme preuve puisqu'on évolue dans un cadre fini. L'avantage essentiel de la méthode de Hoare sur une démonstration par disjonction des 11 cas est qu'elle est immédiatement transposable à d'autres valeurs finales de l'indice (par exemple la somme des 1000 premiers entiers).
On peut alors considérer que puisqu'elle est équivalente dans certains cas à des démonstrations par récurrence, la logique de Hoare est trop difficile pour être envisagée en Seconde. On peut aussi considérer qu'en tant que "récurrence light" elle peut au contraire servir à préparer les élèves de Seconde à une future découverte de la vraie récurrence. Il y a ici matière à débattre...

4) Preuve partielle
En fait on ne démontre pas qu'une propriété p est vraie à l'issue de la boucle, mais qu'elle est vraie à condition qu'on sorte un jour de la boucle. Cette dernière condition ayant été montrée indécidable par Turing en 1936, on sait qu'on ne peut pas faire mieux.

5) Rallye mathématique 2010 de la Réunion
L'exercice 10 de Seconde du Rallye 2010 de la Réunion est intéressant parce qu'il comprend à la fois une boucle et un test dans la boucle.
On a le programme suivant :

```
A=42 ;
B=60 ;
while(A !=B){
        if(A>B){
                A=A-B ;
        } else {
                B=B-A ;
        }
}
```

On demande la valeur finale de A (à la sortie de la boucle) et ce que représente cette valeur par rapport aux valeurs initiales de A et de B.
On note d la valeur finale de A (et donc de B). On cherche alors un invariant de boucle concernant d. Une intuition (conséquence d'une certaine habitude en arithmétique et de simulations) mène à la recherche de $d|A$ ("d divise A") comme invariant. C'est comme moteur de l'intuition que l'aspect expérimental est le plus utile, en faisant émettre une conjecture *ad hoc*. Maintenant en essayant de gérer le test, on constate que $d|A$ n'est ***pas*** un invariant de boucle, et que c'est dû à une rupture de symétrie : Le script est invariant par échange de A et de B, alors que $d|A$ ne l'est pas. Ce qui mène à la recherche de $d|A \wedge d|B$ comme invariant de boucle. La preuve de programme marche bien :

$$d|(A-B) \wedge d|B \,\{\texttt{A=A-B ;}\}\, d|A \wedge d|B$$

et comme $(d|A \wedge d|B) \Rightarrow (d|(A-B) \wedge d|B)$ et $(A > B \wedge d|A \wedge d|B) \Rightarrow (d|A \wedge d|B)$,

$$A > B \wedge d|A \wedge d|B \,\{\texttt{A=A-B ;}\}\, d|A \wedge d|B$$

De même,

$$A \leqslant B \wedge d|A \wedge d|B \,\{\texttt{B=B-A ;}\}\, d|A \wedge d|B$$

D'après la règle des tests, on a alors

$$d|A \wedge d|B \,\{\texttt{if(A>B)\{A=A-B ;\}else\{B=B-A ;\}}\}\, d|A \wedge d|B$$

$d|A \wedge d|B$ est donc bien un invariant de boucle, et d'après la règle de la précondition,

$$A \neq B \wedge d|A \wedge d|B \,\{\texttt{if(A>B)\{A=A-B ;\}else\{B=B-A ;\}}\}\, d|A \wedge d|B$$

donc, d'après la règle des boucles,

$$d|A \wedge d|B \,\{\texttt{while(A !=B)\{if(A>B)\{A=A-B ;\}else\{B=B-A ;\}\}}\}\, A = B \wedge d|A \wedge d|B$$

À ce stade, on a démontré que d divise à la fois 42 et 60, et vaut donc 1, 2, 3 ou 6. Pour savoir laquelle des valeurs est la bonne, on peut remplir un tableau comme le demandait l'énoncé du Rallye, soit chercher à affiner la conjecture sur d en se disant que non mais tout de même, il n'y a pas de raison que d soit n'importe quel diviseur commun à A et B, mais le plus grand d'entre eux. Il se trouve effectivement que $d = pgcd(A, B)$ est aussi un invariant de boucle et que ça se démontre exactement comme ci-dessus. Les détails sont laissés en exercice (l'étape la plus difficile est encore $pgcd(x, y) = pgcd(x - y, y)$).