TD6 : Preuve de programmes

TOP : Techniques and tOols for Programming – 1A

Dans ce TD, nous allons démontrer que les algorithmes étudiés sont corrects en calculant la plus faible précondition nécessaire (*weakest precondition*) pour que l'algorithme donne bien dans la post-condition souhaitée (le résultat est celui attendu). Cette méthode est l'une des plus simples et mécaniques pour démontrer formellement la correction d'algorithmes. Pour cela, il faut appliquer les cinq règles suivante.

1. $\mathbf{WP}(no-op, Q) \equiv Q$
2. $\mathbf{WP}(x := E, Q) \equiv Q[x := E]$
3. $\mathbf{WP}(C; D, Q) \equiv \mathbf{WP}(C, \mathbf{WP}(\text{D,Q}))$
4. $\mathbf{WP}(\texttt{if}\ Cond\ \texttt{then}\ C\ \texttt{else}\ D) \equiv (Cond = \texttt{true} \Rightarrow \mathbf{WP}(C, Q)) \ \wedge\ (Cond = \texttt{false} \Rightarrow \mathbf{WP}(D, Q))$
5. $\mathbf{WP}(\texttt{while}\ E\ \texttt{do}\ C\ \texttt{done}\ \{\text{inv I var V}\},\text{Q}) \equiv I$
   Plus les obligations de preuves suivantes :
   - $(E = \texttt{true} \wedge I \wedge V = z) \Rightarrow \mathbf{WP}(C, I \wedge V < z))$ (preuve que chaque passage décrémente le variant)
   - $I \Rightarrow V \geq 0$ (preuve que le variant reste valide lors les passages successifs)
   - $(E = \texttt{false} \wedge I) \Rightarrow Q$ (preuve qu'après le dernier passage, Q est bien atteint)

La première règle signifie que la plus faible précondition à assurer pour que Q soit vraie après l'exécution de `no-op` (après l'exécution de rien du tout) est Q elle-même.

La seconde règle explicite les affectations de variable : Pour que Q soit vraie après une affectation, il faut que Q soit vraie au préalable avec une réécriture.

La troisième règle indique que la plus faible précondition permettant que Q soit vraie après l'exécution de `C` puis `D`, c'est la précondition à `C` pour $\mathbf{WP}(D, Q)$ soit vrai, c'est à dire pour que soit vérifiée la précondition de D permettant à Q d'être vraie.

La quatrième règle est plus facile à écrire avec l'écriture mathématique qu'avec une paraphrase.

La cinquième règle, au sujet des boucles, n'est pas très compliquée non plus, mais elle impose d'annoter chaque boucle `while` par un invariant et un variant (syntaxe : $\{inv\ I\ var\ V\}$). Par ailleurs, l'application de cette règle produit trois obligations de preuves supplémentaires. Il s'agit d'expression qu'il faudra démontrer par ailleurs pour avoir le droit d'appliquer la règle de calcul du **WP**.

Ensemble, ces cinq règles permettent de démontrer le triplet de Hoare $\{\text{P}\}\ \text{C}\ \{\text{Q}\}$ en montrant la proposition $P \Rightarrow \mathbf{WP}(\text{C,Q})$ ainsi que toutes les obligations de preuves engendrées lors du calcul de $\mathbf{WP}(C, Q)$.

FIB

```
1 i:=1
2 a:=1
3 b:=1
4 while i<n do
5   i:=i+1
6   u:=a
7   a:=a+b
8   b:=u
9 done
```

★ **Exercice 1: Fibonnacci** (d'après Ralf Treinen).
Calculez la plus faible précondition pour que code donné ci-contre admette comme post-condition que $a = fib(n)$ (ie, que l'algorithme calcule Fibonacci de n dans a).

★ **Exercice 2: Calcul du minimum** (d'après Alexandre Miquel).
On considère le code ci-contre, qui calcule le minimum d'une fonction entre 1 et n.

MIN

```
1 m:=f(1)
2 i:=2
3 while i<n do
4   if (f(i)<m) then m:=f(i)
5   i++
6 done
```

▷ **Question 1:** Quelle est la spécification formelle de ce programme ? Spécifiez en particulier la post-condition.

▷ **Question 2:** Trouvez l'invariant et le variant de la boucle while.

▷ **Question 3:** Calculez les obligations de preuve correspondantes, et vérifiez qu'elles sont satisfaites.

★ **Exercice 3: Tri par sélection** (d'après Alexandre Miquel).
On considère le code ci-contre, qui trie un tableau d'entier.
Dans la logique de Hoare, t[i] ne peut être accédé que si la condition suivante est satisfaite : $O \leq i < taille(t)$.

SEL

```
1 for i := 0 to n-2 do
2   for j := i+1 to n-1 do
3     if t[j] < t[i] then
4       tmp := t[i]
5       t[i] := t[j]
6       t[j] := tmp
7     end
8   done
9 done
```

▷ **Question 1:** Réécrivez ce programme en changeant les boucles `for` en boucles `while` équivalentes.

▷ **Question 2:** Montrez la correction du code de la permutation circulaire :

$$\{t[i] = x \wedge t[j] = y\}\ tmp := t[i]; t[i] := t[j]; t[j] := tmp\ \{t[i] = y \wedge t[j] = x\}$$

▷ **Question 3:** Montrez à l'aide de ce qui précède la correction de l'algorithme du tri par sélection.