AFGL – TD 2

MACHINES ABSTRAITES

1 Composition séquentielle

Rappel : $[S_1;S_2](P) \equiv [S_1]([S_2](P))$

Q1.1 *Développer* $[x := x+2; x := x+4](x > 9)$

$$\begin{aligned}[x := x+2; x := x+4](x > 9) &\equiv [x := x+2]([x := x+4](x > 9)) \\ &\equiv [x := x+2](x+4 > 9) \\ &\equiv (x+2+4 > 9) \\ &\equiv x > 3\end{aligned}$$

Q1.2 *Développer* $[y := x+y; x := y-x](y \geq 5 \wedge x \leq 3)$

$$\begin{aligned}[y := x+y; x := y-x](y \geq 5 \wedge x \leq 3) &\equiv [y := x+y]([x := y-x](y \geq 5 \wedge x \leq 3)) \\ &\equiv [y := x+y](y \geq 5 \wedge y-x \leq 3) \\ &\equiv (x+y \geq 5 \wedge y \leq 3)\end{aligned}$$

Q1.3 *Développer* $[x := y; y := x^2](y > x)$

$$\begin{aligned}[x := y; y := x^2](y > x) &\equiv [x := y]([y := x^2](y > x)) \\ &\equiv [x := y](x^2 > x) \\ &\equiv y^2 > y\end{aligned}$$

2 Variant et invariant

Soit la boucle suivante qui calcule le produit des éléments d'un tableau a.

```
1 prod := 1;
2 i := 0;
3 while i < N do
4     i := i + 1;
5     prod := prod * a(i);
6 INVARIANT ..............
7 VARIANT ................
8 end
```

Q2.1 *Donner l'invariant et le variant de la boucle permettant d'établir la postcondition* $prod = \prod_{j=1}^{N} a(j)$

Rappel : la valeur entière de l'expression du variant doit diminuer jusqu'à arriver à 0, sans jamais être négatif.

INVARIANT $\text{prod} = \prod_{j=1}^{i} a(j) \wedge i \in 0..N$

VARIANT (N - i)

3 Somme d'entiers

On se propose de développer une boucle pour calculer la somme des éléments d'un tableau $a \in 1..N \rightarrow \mathbb{N}$. La postcondition est donnée par $sum = \sum_{j=1}^{N} a(j)$

```
sum := 0;
i := 0;
while i ≠ N do
    i := i + 1;
    sum := sum + a(i);
INVARIANT sum = ∑_{j=1}^{i} a(j) ∧ i ∈ ℕ ∧ i ≤ N
VARIANT N - i
end
```

Q3.1 *Réécrire la boucle en modifiant la postcondition* $sum = \sum_{j=i}^{N} a(j) \wedge i \in \mathbb{N} \wedge i = 0$.

Correspond a un changement d'ordre de la boucle.

```
sum := 0;
i := N;
while i > 0 do
    sum := sum + a(i);
    i := i - 1;
INVARIANT sum = ∑_{j=i}^{N} a(j) ∧ i ∈ ℕ ∧ i ≤ N
VARIANT i
end
```

4 Multiplication russe

Soit le programme suivant avec la postcondition $total = a \times b$

```
x := a;
y := b;
total := 0;
while x > 0 do
    if x mod 2 = 1 then
        total := total + y
        x := x - 1;
    end;
    x := x / 2;
    y := y * 2;
INVARIANT x ∈ ℕ ∧ total + x * y = a * b
VARIANT x
end
```

Q4.1 *Prouver la conformité de ce programme.*

1. Invariance : $I \wedge E \Rightarrow [S]I$
   $(x \in \mathbb{N} \wedge total + x \times y = a \times b \wedge x > 0) \Rightarrow [if\ x \mod 2 = 1\ then\ total := total + y; x := x - 1; end; x := x / 2; y := y \times 2](x \in \mathbb{N} \wedge total + x \times y = a \times b)$
   - si x est impair (on omet le prédicat $x \in \mathbb{N}$ qui est vérifié) :
     $[total := total + y; x := x - 1; x := x / 2; y := y \times 2](total + x \times y = a \times b)$
     $\equiv total + y + \frac{x-1}{2} \times y \times 2 = a \times b$
     $\equiv total + y + x \times y - y = a \times b$
     $\equiv total + x \times y = a \times b$
   - si x est pair :
     $[x := x / 2; y := y \times 2](total + x \times y = a \times b) \equiv total + \frac{x}{2} \times y \times 2 = a \times b$
     $\equiv total + x \times y = a \times b$
2. Typage : $I \wedge E \Rightarrow V \in \mathbb{N}$
   $(x \in \mathbb{N} \wedge total + x \times y = a \times b \wedge x > 0) \Rightarrow x \in \mathbb{N} \Rightarrow V \in \mathbb{N}$ car $V = x$
3. Terminaison : $I \wedge E \wedge (V = \gamma) \Rightarrow [S](V < \gamma)$
   $x \in \mathbb{N} \wedge total + x \times y = a \times b \wedge x > 0 \wedge x = \gamma \Rightarrow [if\ x \mod 2 = 1\ then\ total := total + y; x := x - 1; end; x := x / 2; y := y \times 2](x < \gamma)$
   - si x est impair :
     $[total := total + y; x := x - 1; x := x / 2; y := y \times 2](x < \gamma) \equiv \frac{x-1}{2} < \gamma$
     $[...\wedge]x > 0 \wedge x = \gamma \Rightarrow \frac{x-1}{2} < \gamma$
   - si x est pair :
     $[x := x / 2; y := y \times 2](x < \gamma) \equiv \frac{x}{2} < \gamma$
     $[...\wedge]x > 0 \wedge x = \gamma \Rightarrow \frac{x}{2} < \gamma$
4. Finalisation : $I \wedge \neg E \Rightarrow R(total := a \times b)$
   $x \in \mathbb{N} \wedge total + x \times y = a \times b \wedge x = 0 \Rightarrow total = a \times b$

5 Feux de circulation

Considérons un carrefour à 4 voies contrôlé par des feux de circulation dans les deux directions *EstOuest* et *NordSud*. A tout moment, les feux *Est* et *Ouest* sont de la même couleur, de même pour *Nord* et *Sud*. Les feux peuvent prendre les couleurs *Rouge*, *Vert* et *Orange*.

- **Propriété de séquentialité :** La suite de couleurs possible est *Rouge*, *Vert*, *Orange*, *Rouge*, ...
- **Propriété de sûreté :** Si dans une direction les feux sont *Vert* ou *Orange*, dans l'autre direction les feux sont *Rouge*.

Il y a 3 opérations à spécifier :

- `PasseAuRouge(dir)` : met les feux au *Rouge* dans la direction `dir`
- `PasseAuVert(dir)` : met les feux au *Vert* dans la direction `dir`
- `PasseAlOrange(dir)` : met les feux à l'*Orange* dans la direction `dir`

Chaque opération doit avoir les préconditions appropriées qui assurent que la propriété de sûreté est vérifiée.

Q5.1 *Écrire la Machine Abstraite correspondant à la description de ce problème.*

```
MACHINE FeuxTricolores
SETS
    DIRECTION = {NordSud, EstOuest};
    FEU = {Rouge, Vert, Orange}
VARIABLES
    feux
INVARIANT
    feux ∈ DIRECTION ⟶ FEU ∧
    (feux(NordSud) = Rouge ∨ feux(EstOuest) = Rouge)
INITIALISATION
    feux := {NordSud ↦ Rouge, EstOuest ↦ Rouge}
OPERATIONS
    PasseAuRouge(dir) =
        PRE
    //propriété de séquentialité : seul un feu Orange peut passer au Rouge
            dir ∈ DIRECTION ∧ feux(dir) = Orange
        THEN feux(dir) := Rouge
        END;
    PasseAuVert(dir) =
        PRE
    //propriété de séquentialité : seul un feu Rouge peut passer au Vert
            dir ∈ DIRECTION ∧ feux(dir) = Rouge ∧
    //propriété de surete : les feux de la direction opposée doivent être Rouge
            (dir = NordSud ⇒ feux(EstOuest) = Rouge) ∧
            (dir = EstOuest ⇒ feux(NordSud) = Rouge)
        THEN feux(dir) := Vert
        END;
    PasseAlOrange(dir) =
        PRE
    //propriété de séquentialité : seul un feu Vert peut passer à l'Orange
            dir ∈ DIRECTION ∧ feux(dir) = Vert
        THEN feux(dir) := Orange
        END
END
```

Invariant de la correction d'Allel (logiquement équivalent car $p \Rightarrow q \equiv \neg p \vee q$) :

```
INVARIANT
    feux ∈ DIRECTION ⟶ FEU ∧
    (feux(NordSud) ∈ {Vert, Orange} ⇒ feux(EstOuest) = Rouge) ∧
    (feux(EstOuest) ∈ {Vert, Orange} ⇒ feux(NordSud) = Rouge)
```

6 Auto-école

Une auto-école décide d'informatiser la gestion des candidats au permis de conduire. On considère pour simplifier qu'il n'existe qu'un seul type de permis. Pour cela elle gère :

- un ensemble d'inscrits (*inscrit*),

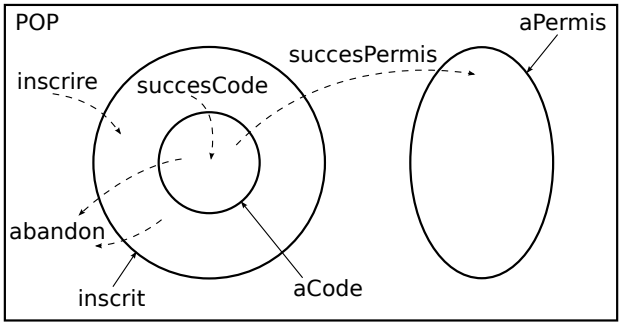
- un ensemble d'élèves qui ont passé le code avec succès mais qui n'ont pas encore le permis (*aCode* $\subseteq$ *inscrit*),
- un ensemble d'anciens élèves qui ont le permis (*aPermis*).

Tous ces ensembles sont exprimés sur l'ensemble support *POP*. Les différentes opérations répertoriées sont :

- `inscrire`,
- `succesCode` : un inscrit obtient le code,
- `succesPermis` : un possesseur du code obtient le permis,
- `abandon` : un inscrit abandonne et quitte l'auto-école.

Q6.1 *Analyser informellement le problème.*

- les variables sont des ensembles reliés
- les opérations sont des basculements d'ensembles
- `succesPermis` fait sortir des inscrits, mais pas `succesCode`
- un abandon fait perdre le bénéfice du code



Q6.2 *Compléter et rédiger la machine abstraite* `autoEcole` *suivante.*

```
MACHINE
    autoEcole
SETS
    POP
VARIABLES
    inscrit, aCode, aPermis
INVARIANT
    ......................
INITIALISATION
    inscrit, aCode, aPermis = ∅, ∅, ∅
OPERATIONS
    ......................
```

```
MACHINE
    autoEcole
SETS
    POP
```

```
VARIABLES
    inscrit, aCode, aPermis
INVARIANT
    inscrit ∪ aPermis ⊆ POP
    ∧ inscrit ∩ aPermis = ∅
    ∧ aCode ⊆ inscrit
INITIALISATION
    inscrit, aCode, aPermis = ∅, ∅, ∅
OPERATIONS
    inscrire(e) =
        PRE
            e ∈ POP - (inscrit ∪ aPermis)
        THEN
            inscrit := inscrit ∪ {e}
        END;
    succesCode(e) =
        PRE
            e ∈ inscrit - aCode
        THEN
            aCode := aCode ∪ {e}
        END;
    succesPermis(e) =
        PRE
            e ∈ aCode
        THEN
            inscrit := inscrit - {e};
            aCode := aCode - {e};
            aPermis := aPermis ∪ {e};
        END;
    abandon(e) =
        PRE
            e ∈ inscrit
        THEN
            inscrit := inscrit - {e};
            aCode := aCode - {e}; // OK même si e ∉ aCode
        END;
END
```

Q6.3 *Poser et simplifier les obligations de preuve.*

1. L'initialisation établit l'invariant : C (constraints) $\Rightarrow [U]I$
   $(\emptyset \cup \emptyset \subseteq POP) \wedge (\emptyset \cap \emptyset = \emptyset) \wedge (\emptyset \subseteq \emptyset)$
2. Les opérations conservent l'invariant : $C \wedge I \wedge P \Rightarrow [S]I$
   Exemple : succesCode
   $(inscrit \cup aPermis \subseteq POP \wedge inscrit \cap aPermis = \emptyset \wedge aCode \subseteq inscrit) \wedge (e \in inscrit - aCode) \Rightarrow [aCode := aCode \cup \{e\}](inscrit \cup aPermis \subseteq POP \wedge inscrit \cap aPermis = \emptyset \wedge aCode \subseteq inscrit)$
   $(inscrit \cup aPermis \subseteq POP \wedge inscrit \cap aPermis = \emptyset \wedge aCode \subseteq inscrit) \wedge (e \in inscrit - aCode) \Rightarrow (inscrit \cup aPermis \subseteq POP \wedge inscrit \cap aPermis = \emptyset \wedge aCode \cup \{e\} \subseteq inscrit)$