CPS - Notes de TD

Jordi Bertran de Balanda

TD1 - Dataflow Require/Provide

Ex1. Communication évènementielle

Q1. Interfaces de service

Service considéré: pouvoir recevoir un évènement.

```
public interface IntEventReceiverService {
    public void onIntEvent(IntEvent event);
}
```

Q2. Interfaces de liaison

```
public interface requireIntEventReceiver {
    public void bindIntEventReceiver(IntEventReceiverService receiver);
}

public interface requireActivator {
    public void bindActivator(ActivatorService activator);
}
```

Q3. Composant émetteur

Pas de IntEventSenderService, pas réifié (cf Q2).

Q4. Composant récepteur

```
public class Printer implements Component, IntEventReceiverService {
    private BigInteger msg;
    public Printer() {
        this.msg = null;
```

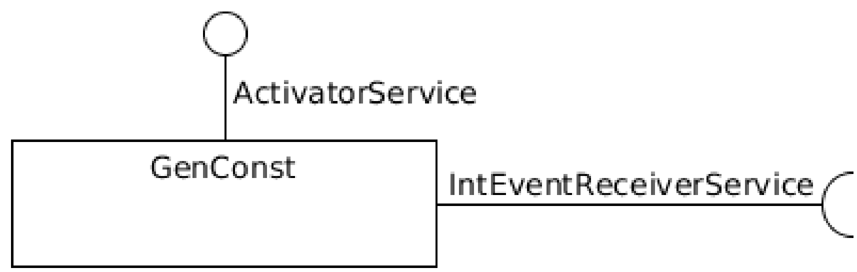

Figure 1: Diagramme de GenConst - Q3

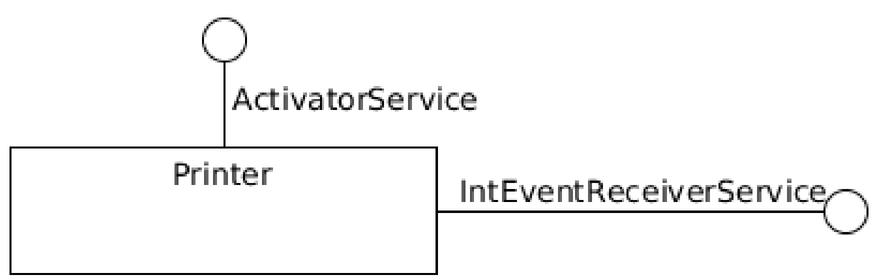

Figure 2: Diagramme de Printer - Q4

```
    }

    public void onIntEvent(IntEvent event){
        if (msg != null)
            throw new ReceptionException();
        else
            msg = event.getValue()
    }

    public void activate() {
        System.out.println(msg?msg:".");
        msg = null;
    }
}
```

Q5. Composition

```
public class Composition {
    public static void main (String[] args) {
        GenConst gen = new GenConst(1);
        Printer printer = new Printer();
        gen.bindIntEventReceiverService(printer);
        // Point (1)
        for (int i = 0; i < 10; i++) {
            gen.activate();
            printer.activate();
        }
    }
}
```

Le programme produit une suite de 10 fois le nombre 1 sur sa sortie standard.

Figure 3: Diagramme de composition - Q5

Ex2. Dataflow

Q1. Génération des entiers naturels

cf. Feuille.

Le composant add/mul/div/sub attend d'avoir obtenu 2 évènements.

Q2. Composant composite

But: créer un composant *composite* qui envoie des évènements de valeurs successives (1, 2...). Ce composant offre ActivatorService, et requiert in IntEventReceiverService.

```java
public class GenInt implements Component, requireIntEventReceiver {
    private GenConst gen;
    private Add plus;
    public GenInt (BigInteger value) {
        gen = new GenConst(value);
        plus = new Add();
        gen.bind(plus);
        plus.bind(plus);
        plus.onIntEvent(new IntEvent(value - 1));
    }

    public void bindIntEventReceiver (IntEventReceiverService rec) {
        plus.bind(rec);
    }

    public void activate () {
        gen.activate();
        plus.activate();
    }
}
```

Diagramme de composants: cf. feuille.

TD2 - Spécifications

Spécifications bancaires

Q1. Compte bancaire.

- **Service:** Compte

- **Observateurs:**
  - *const* nom : [Compte] $\rightarrow$ String
  - *const* numero : [Compte] $\rightarrow$ int
  - solde : [Compte] $\rightarrow$ double
  - *const* limite : [Compte] $\rightarrow$ double
  - decouvert : [Compte] $\rightarrow$ double
    - **pre:** decouvert(C) **require** aDecouvert(C)
  - aDecouvert : [Compte] $\rightarrow$ bool
  - retraitPossible : [Compte] * double $\rightarrow$ bool
- **Constructeurs:**
  - init : String * int * double $\rightarrow$ [Compte]
    - **pre:** init(nom, num, dec) require (num > 0) $\wedge$ (dec ≥ 0)
  - init : Compte $\rightarrow$ [Compte]
- **Opérateurs:**
  - depot : [Compte] * double $\rightarrow$ [Compte]
    - **pre:** depot(C, s) require s > 0
  - retrait : [Compte] * double $\rightarrow$ [Compte]
    - **pre:** retrait(C, s) require retraitPossible(C, s)
- **Observations:**
  - [invariants]
    - aDecouvert(C, s) (min)= solde < 0
    - retraitPossible(C, s) (min)= $0 <$ s $\leq$ solde(C) + limite(C)
    - decouvert(C) (min)= - solde(C)
    - solde(C) $\geq$ - limite(C) # PROPRIÉTÉS
    - $0 \leq$ decouvert(C) $\leq$ limite(C) # IMPORTANTES
  - [init]
    - nom(init(n, num, lim)) = n # Const:
    - numero(init(n, num, lim)) = num # plus besoin de faire
    - limite(init(n, num, lim)) = lim # d'observations après init
    - solde(init(n, num, lim)) = 0
    - nom(init(C)) = Compte::nom(C)
    - numero(init(C)) = Compte::numero(C)
    - limite(init(C)) = Compte::limite(C)
    - solde(init(C)) = 0
  - [retrait]
    - limite(retrait(C, s)) = limite(C) # Redondant avec const
    - solde(retrait(C, s)) = solde(C) - s

Q2. Propriétés

Complet: pour tout état, pour tout observateur accessible, il faut pouvoir donner une valeur.

Activable: dans tous les états, au moins une opération est accessible.

Le service est activable.

Convergence:

- depot: opération divergente
- retrait: opération convergente
- retrait : [Compte] * double → [Compte]
  - **pre:** retrait(C, s) require retraitPossible(C, s)
    - **converge** solde(C)

Q3. Agence

- **Service:** Agence
- **Observateurs:**
  - *const* nom : [Agence] → String
  - numeros : [Agence] → Set
  - nbComptes : [Agence] → int
  - compteExiste : [Agence] * int → bool
  - compte : [Agence] * int → Compte
    - **pre** getCompte(A, num) require compteExiste(A, num)
- **Constructeurs:**
  - init : String → Agence
- **Opérateurs:**
  - create : [Agence] * String * int * double → [Agence]
    - **pre** create(A, nom, num, lim) require not(compteExiste(A, num))
  - virement : [Agence] * int * int * double → [Agence]
    - **pre** virement(A, src, dst, s) require compteExiste(A, src) ∧ compteExiste(A, dst) ∧ Compte::retraitPossible(compte(A, src), s)
- **Observations:**
  - [invariants]
    - nbComptes(A) (min)= card(numeros(A)) card(E) le cardinal de l'ensemble E

    - ∗ compteExiste(A, num) (min)= num $\in$ numeros(A)
  - [init]
    - ∗ nom(init(n)) = n
    - ∗ numeros(init(nom)) = NONE
  - [create]
    - ∗ numero(create(A, nom, num, lim)) = numeros(A) $\cup$ {num}
  - [compte]
    - ∗ compte(create(A, nom, num, lim)) = Compte::init(nom, num, lim)
    - ∗ $\forall$ num2, num2 $\neq$ num $\Rightarrow$ compte(create(A, nom, num, lim), num2) = compte(A, num2)
  - [virement]
    - ∗ numeros(virement(A, src, dst, s)) = numeros(A)
    - ∗ compte(virement(A, src, dst, s), src) = Compte::retrait(compte(A, src), s)
    - ∗ compte(virement(A, src, dst, s), dst) = Compte::depot(compte(A, dst), s)
    - ∗ $\forall$ num, num $\neq$ src $\wedge$ num $\neq$ dst : compte(virement(A, src, dst, s), num) = compte(A, num)

TD3 - Conception par contrats

Idée $\rightarrow$ Spec $\rightarrow$ Contrat $\rightarrow$ Impl

Ex1. Compte bancaire

Interface

```
public interface ICompte {

    /********** Observators **********/

    String nom();               // Const
    int numero();               // Const
    double solde();
    double limite();            // Const
    // \pre estDecouvert() == true
    double montantDecouvert();
    bool estDecouvert();
    // \pre s > 0
    bool peutPrelever(double s);
```

```
    /*********** Operators ***********/

    // \pre somme > 0
    // \post solde() == solde()@pre + somme
    void depot(double somme);
    // \pre peutPrelever(somme) == true
    // \post somme() == somme()@pre + somme
    void retrait(double somme);

    /********** Initializers *********/
    /*********************************
     * \pre nom != ""
     * \pre num > 0
     * \pre dec >= 0
     * \post nom().equals(n)
     * \post numero() == num
     * \post limite() == dec
     * \post solde() == 0
     *********************************/
    void init(String nom, int num, double dec);
    void init(Compte c);

    /********** Invariants **********/
    /********************************
     * \inv montantDecouvert() == -solde()
     * \inv \forall s:double \with s>0 {peutPrelever(s) == (solde()-s)>=limite()
     ********************************/
```

Implem

```
public class CompteImpl implements ICompte {
    private String nom;
    ..
    public CompteImpl() { ... }
    public void init()  { ... }
    public String nom() { return nom; }
    ..
}
```

Decorateur

```
public abstract class CompteDecorateur implements ICompte {
    private ICompte delegate;
```

```
    protected CompteDecorateur(Compte delegate) {
        this.delegate = delegate;
    }
    public String nom() { return delegate.nom() }
    ..

}
```

Contrat

```
public class CompteContrat extends CompteDecorateur {
    public CompteContract(Compte delegate) {
        super(delegate);
    }
    public void checkInvariants() {
        // \inv montantDecouvert() == -solde()
        if (estDecouvert() && montantDecouvert() != -solde())
            throw new InvariantError(err);
        // \inv \forall s:double \with s>0 { peutPrelever(s) == (solde()-s)>=limite() }
        double s1 = solde() + limite();
        if (!(peutPrelever(s1) == true))
            throw new InvariantError("...");
        double s2 = s1 / 2;
        if (!(peutPrelever(s2) == true))
            throw new InvariantError("...");
                double s1 = solde() + limite();
        double s3 = s1;
        if (!(peutPrelever(s3) == true))
            throw new InvariantError("...");
    }
    public void retrait(double s) {
        // (1) Préconditions
        // \pre peutPrelever(s)
        if (!(peutPrelever(s) == true))
            throw new PreconditionException("...");
        // (2) Invariants
        checkInvariants();
        // (3) Capture
        String nom_pre = nom();
        int solde_pre = solde();
        // (4) Métier
        super.retrait(s);
        // (5) Invariant
        checkInvariant();
        // (6) Postconditions
```

```
        // \post solde() == solde()@pre - s
        if (!(solde() == solde_pre - s))
            throw new PostconditionException("...");
    }
}
```

Links

```
Compte compte = new Compte();
CompteContrat contrat = new CompteContrat();
contrat.init( .. );
compte.init( .. );
```

Ex2. Agence bancaire

```
public interface ...
```

TD 5 - Tests MBT

Service Commandes (non spécifié) suit 2 listes:

- uplist triée en ordre croissant les commandes pour monter
- downlist triée en ordre décroissant les commands pour descendre

Ex1. Couverture des préconditions

11 objectifs de précondition.

1. **init**
   1. Test *positif*:
      - *Conditions initiales*: aucune
      - *Opération*: L_{11} = init(2, 5)
      - *Oracle*: **pas** d'exception levée
   2. Test *négatif*:
      - *Conditions initiales*: aucune
      - *Opération*: L_{12} = init(-2, 5)
      - *Oracle*: Exception levée
2. **beginMoveUp:** non atteignable. Le service commande n'est pas spécifié.
3. **stepMoveUp:** non atteignable.

4. **endMoveUp:** non atteignable.
5. **beginMoveDown:** non atteignable.
6. **stepMoveDown:** non atteignable.
7. **endMoveDown:** non atteignable.
8. **openDoor**
    1. Test *positif*:
        - *Conditions initiales*: L_{01} = doorAck(closeDoor(init(2, 5)))
        - *Opération*: L_{81} = openDoor(L_0)
        - *Oracle*: Pas d'exception levée
    2. Test *négatif*:
        - *Conditions initiales*: L_{02} = init(2, 5)
        - *Opération*: L_{82} = openDoor(L)
        - *Oracle*: Exception levée
    3. Test *négatif*: Impossible d'atteindre un statut différent de IDLE sans avoir la spécification de Commandes
    4. Test *négatif*: Idem que 8.3
9. **closeDoor**
    1. Test *positif*:
        - *Conditions initiales*: L_{01} = init(2, 5)
        - *Opération*: L_{91} = openDoor(L_{01})
        - *Oracle*: Pas d'exception levée
    2. Test *négatif*:
        - *Conditions initiales*: L_{02} = doorAck(closeDoor(init(2, 5)))
        - *Opération*: L_{92} = closeDoor(L)
        - *Oracle*: Exception levée
    3. Test *négatif*: Impossible d'atteindre un statut différent de IDLE sans avoir la spécification de Commandes
    4. Test *négatif*: Idem que 9.3
10. **doorAck**
    1. Test *positif*:
        - *Conditions initiales*: L_{01} = closeDoor(init(2, 5))
        - *Opération*: L_{101} = doorAck(L_{01})
        - *Oracle*: Pas d'exception levée
    2. Test *positif*:
        - *Conditions initiales*: L_{02} = openDoor(doorAck(closeDoor(init(2, 5))))
        - *Opération*: L_{101} = doorAck(L_{02})
        - *Oracle*: Pas d'exception levée
    3. Test *négatif*:

    - *Conditions initiales*: $L_{03} = \text{init}(2, 5)$
    - *Opération*: $L_{101} = \text{doorAck}(L_{03})$
    - *Oracle*: Exception levée

11. **selectLevel**

Couverture des tests de préconditions, en l'absence de la spécification de Commandes: 27,2%.

Ex2. Couverture en termes d'automates

Q1. Couverture des transitions

1. **init**
    - *Conditions initiales*: aucune
    - *Opération*: $L_{11} = \text{init}(2, 5)$
    - *Oracle (Post)*: $\text{minLevel}(L_{11}) = 2$, $\text{maxLevel}(L_{11}) = 5 \ldots$
    - *Oracle (Inv)*: $\text{minLevel}(L_{11}) \leq \text{level}(L_{11}) \leq \text{maxLevel}(L_{11})$
2. **beginMoveUp:** non atteignable. Le service commande n'est pas spécifié.
3. **stepMoveUp:** non atteignable.
4. **endMoveUp:** non atteignable.
5. **beginMoveDown:** non atteignable.
6. **stepMoveDown:** non atteignable.
7. **endMoveDown:** non atteignable.
8. **openDoor**
    - *Conditions initiales*: $L_0 = \text{doorAck}(\text{closeDoor}(\text{init}(2, 5)))$
    - *Opération*: $L_8 = \text{openDoor}(L_0)$
    - *Oracle (Post)*: $\text{doorStatus}(L_8) = \text{OPENING}$
    - *Oracle (Inv)*: $\text{minLevel}(L_8) \leq \text{level}(L_8) \leq \text{maxLevel}(L_8)$ ..

Couverture de transitions: 18%.

Q2. Couverture des états remarquables

1. **STANDBY_UP**
    - *Conditions initiales*: aucune
    - *Opération*: $L_{331} = \text{selectLevel}(\text{doorAck}(\text{closeDoor}(\text{init}(2, 5)), 3))$
    - *Oracle(Inv)*: $\text{minLevel}(L_{331}) \leq \text{level}(L_{331}) \leq \text{maxLevel}(L_{331})$ ..

Q3. Couverture de paires de transitions

1. **doorAck; openDoor**
    - *Conditions initiales*: L_{41} = closeDoor(init(2, 5))
    - *Opération*: L_{42} = openDoor(doorAck(L_{41}))
    - *Oracle(P) 1*: ..
    - *Oracle(I) 1*: ..
    - *Oracle(P) 2*: ..
    - *Oracle(I) 2*: ..

Q4. Couverture des use case

Raconter des histoires (cf. IL)

Ex3. Couverture des données

Gestion des plages de valeurs.

1. **selectLevel:** tester selectLevel(L, m), avec par ex. init(0, 5)
    - Un cas dans les bornes - selectLevel(L, 2)
    - Trois cas aux bornes - selectLevel(0), selectLevel(5), selectLevel(2)
    - Deux cas hors bornes - selectLevel(-1), selectLevel(6)

TD6 - Logique de Hoare I

Pourquoi: analyse statique du code. Difficile à faire dans les programmes compliqués, plus sûr que des tests qui ne couvrent potentiellement pas tous les cas de figure.

Logique de Hoare: fondement scientifique de

Sémantique axiomatique

Denotational Semantics	Big Step OS	Small Step OS	Hoare Logic
$\vdash (P) = f$	$\vdash (P, \sigma) \Downarrow (v, \sigma')$	$\vdash (P, \sigma) \rightarrow (P', \sigma')$	$\{P\}P\{Q\}$

Règles de Hoare

$$\frac{}{\{Q[\frac{\text{expr}}{v}]\}v = \text{expr}\{Q\}}(\text{aff})$$

$$\frac{\{P\}I_1\{Q_1\} \;\; \{Q_1\}I_2\{Q_2\} \;\; .. \;\; \{Q_{n-1}\}I_n\{Q\}}{\{P\}I_1; I_2; ..; I_n\{Q\}}(\text{seq})$$

$$\frac{P \Rightarrow P' \;\; \{P'\}C\{Q\}}{\{P\}C\{Q\}}(\text{mp-pre})$$

$$\frac{\{P_1\}C_1\{Q\} \;\; \{P_2\}C_2\{Q\}}{\{(B \Rightarrow P_1); (\neg B \Rightarrow P_2)\} \;\; \text{if } B \text{ then } C_1 \text{ else } C_2\{Q\}}(\text{if})$$

Ex1. Plus faible précondition

Q1. Affectations

Satisfaire $\{P\}i = i + 1\{i > 0\}$

1. $\{(i > 0)[\frac{i+1}{i}]\}i = i + 1\{i > 0\}$
    - $(i > 0)[\frac{i+1}{i}] \Leftrightarrow (i + 1 > 0) \Leftrightarrow (i > -1) \Leftrightarrow i \geq 0$

Satisfaire $\{P\}k = (lo + hi)\text{div}2\{lo \leq k \leq hi\}$

1. $\{(lo \leq \frac{lo+hi}{2} \leq hi)\}k = \frac{lo+hi}{2}\{lo \leq k \leq hi\}$
    - $(lo \leq \frac{lo+hi}{2} \leq hi)$
    - $\Leftrightarrow lo \leq \frac{lo+hi}{2} \wedge hi \geq \frac{lo+hi}{2}$
    - $\Leftrightarrow \frac{lo}{2} \leq \frac{hi}{2} \wedge \frac{lo}{2} \leq \frac{hi}{2}$
    - $\Leftrightarrow lo \leq hi \wedge lo \leq hi$
    - $\Leftrightarrow lo \leq hi$

Q2. Séquencement

Satisfaire $\{P\}x = x - 1; y = y - 1\{x = y\}$

1. $\{x = y - 1\}y = y - 1\{x = y\}$ (aff)
2. $\{x - 1 = y - 1\}x = x - 1; \{x = y\}$ (aff)
    - $\Leftrightarrow x = y$
3. $\{x = y\}x = x - 1; y = y - 1\{x = y\}$ (seq)(2)(1)

Satisfaire $\{P\}y = x; u = 4 * x + 3 * y; t = 3 * x + 5 * y\{t = 8 \wedge u = 7\}$

```
$\left\{
\begin{tabular}{@{}l@{}}
    tomatoes \\
    onions \\
    cucumbers
\end{tabular}
\right\}$
```

1. $\{3 * x + 5 * y = 8u = 7\}\, t = 3 * x + 5 * y\{t = 8 \wedge u = 7\}$ (aff)
2. $\{3*x+5*y = 8 \wedge 4*x+3*y = 7\}u = 4*x+3*y\{3*x+5*y = 8 \wedge u = 7\}$ (aff)
3. $\{3*x+5*x = 8 \wedge 4*x+3*x = 7\}y = x\{3*x+5*y = 8 \wedge 4*x+3*y = 7\}$ (aff)
   - $\Leftrightarrow 8x = 8; 7x = 7$
   - $\Leftrightarrow x = 1$
4. $\{x = 1\}y = x; u = 4*x+3*y; t = 3*x+5*y\{t = 8 \wedge u = 7\}$ (seq)(3)(2)(1)

Q3. Alternative

Satisfaire $\{P\}$ if $(x > 0)$ $z = x$ else $z = -x\{z = |x|\}$

1. $\{-x = |x|\}z = -x\{z = |x|\}$(aff)
2. $\{x = |x|\}z = x\{z = |x|\}$(aff)
3. $\{(x > 0) \Rightarrow x = |x| \wedge \neg(x > 0) \Rightarrow -x = |x|\}$if..else..$\{z = |x|\}$ (if)(2)(1)
   - $\Leftrightarrow (x > 0) \Rightarrow x = |x| \wedge (x \leq 0) \Rightarrow -x = |x|$
   - $\Leftrightarrow \top$

Satisfaire $\{P\}$ $x = 4;$ if $(x > y)$ $z = x$ else $z = y$ $\{z = 3\}$

1. $\{y = 3\}z = y\{z = 3\}$(aff)
2. $\{x = 3\}z = x\{z = 3\}$(aff)
3. $\{x > y \Rightarrow x = 3 \wedge \neg x > y \Rightarrow y = 3\}if..\{z = 3\}$(if)(2)(1)
   - $\Leftrightarrow (x > y) \Rightarrow x = 3 \wedge (x \leq y) \Rightarrow y = 3$
4. $\{(y < 4) \Rightarrow 4 = 3 \wedge (y \geq 4) \Rightarrow y = 3\}x = 4\{(x > y) \Rightarrow x = 3 \wedge (x \leq y) \Rightarrow y = 3\}$(aff)
5. $\{(y < 4) \Rightarrow 4 = 3 \wedge (y \geq 4) \Rightarrow y = 3\}Prog\{(x > y) \Rightarrow x = 3 \wedge (x \leq y) \Rightarrow y = 3\}$(seq)(4)(3)
   - $(y < 4) \Rightarrow 4 = 3 :$ explosion si $y < 4$
   - $y \geq 4 \wedge y \geq 4 \Rightarrow y = 3 \wedge y = 3 \Leftrightarrow \perp \wedge y \geq 4 \Rightarrow y = 3$

Ex2. Preuve de programme

Démontrer $\{x > 2\}\ a = 1; y = x; y = y - a\ \{y > 0 \wedge x > y\}$

1. $\{y > a \wedge x + a > y\}\ y = y - a\ \{y > 0 \wedge x > y\}$(aff)
2. $\{x > a \wedge a > 0\}\ y = x\ \{y > a \wedge x + a > y\}$(aff)
3. $\{x > 1 \wedge 1 > 0\}\ a = 1\ \{x > a \wedge a > 0\}$(aff)
4. $\{x > 1 \wedge 1 > 0\}\ Prog\ \{y > 0 \wedge x > y\}$(seq)(3)(2)(1)
    - $x > 1$
5. $x \geq 2 \Rightarrow x \geq 1$(tautologie)
6. $\{x > 2\} Prog \{y > 0 \wedge x > y\}$(mp)(5)(4)

Ex3. Des contrats aux preuves

Code:

```
PRE
INV
<capture>
<corps>
INV
POST
```

Preuve: $\{PRE \wedge INV\} < capture >; < corps > \{POST \wedge INV\}$ # TD7 - Logique de Hoare II

Règles

$$\frac{\{P\}C\{Q\} \quad v_1, v_2..v_n \notin P \cup Q}{\{P\}\mathrm{var}v_1, \mathrm{var}v_2, .., \mathrm{var}v_n, C\{Q\}}(\mathrm{let})$$

$$\frac{}{\{Q[\frac{A(ex_1 \leftarrow ex_2)}{A}]\}A[ex_1] = ex_2\{Q\}}(\mathrm{tab})$$

$$\frac{I \wedge S \Rightarrow I' \quad \{I'\}C\{I\} \quad I \wedge \neq S \Rightarrow Q}{\{I\}\text{While } S \text{ do } C\{Q\}}(\mathrm{while})$$

Ex1.

Q1.

- Invariant:
  - Invariant
  - Vrai en entrée (et en sortie)
- Q: y=x
- I invariant
- $I \land \neg(a \neq 0) \Rightarrow (y = x)$
- $I \land (a = 0) \Rightarrow (y = x)$
- Invariant: x = a + y
- Variant: a

```
{
    var a;
    a = x;
    y = 0;
    while (a != 0) {
        y = y + 1;
        a = a - 1;
        assert(x == a + y);
    }
}
```

Q2.

$$\{x \geq 0\}Prog_1\{x = y\}$$

1. $\{x = a - 1 + y\}a = a - 1\{x = a + y\}$(aff)
2. $\{x = a - 1 + y + 1\}y = y + 1\{x = a + y\}$(aff)
   - $\Leftrightarrow x = a + y$
3. $(x = a + y) \land (a \neq 0) \Rightarrow (x = a + y)$(CQFD)
4. $(x = a + y) \land (a = 0) \Rightarrow (x = y)$(CQFD)
5. $\{x = a + y\}..;..;\{x = a + y\}$(seq)(2)(1)
6. $\{x = a + y\}$while $(a \neq 0)$ do $..\{x = y\}$(while)(3)(5)(4) − On rajoute la contrainte du variant ici
7. $\{a \geq 0 \land x = a\}y = 0\{a \geq 0 \land x = a + y\}$(aff)
8. $\{x \geq 0 \land x = x\}a = x\{a \geq 0 \land x = a\}$(aff)
9. $\{x \geq 0\}..;..;..\{x = y\}$(seq)(8)(7)(6)
10. $\{x \geq 0\}vara;...\{x = y\}$(var)(9)

Ex2.

1. $\{tab'[i] = y \wedge tab'[j] = x \wedge tab' = tab(j \leftarrow r)\} tab[j] = r \{tab[i] = x \wedge tab[j] = y\}$(tab)
2. $\{tab'[i] = y \wedge tab'[j] = x \wedge tab' = tab(i \leftarrow tab[j])\} tab[i] = tab[j] \{tab'[i] = x \wedge tab'[j] = y \wedge\}$(tab)
3. $\{tab'[i] = y \wedge tab'[j] = x \wedge tab' = tab(i \leftarrow tab[j])(j \leftarrow tab[i])\} r = tab[i] \{tab'[i] = y \wedge tab'[j] = x \wedge tab' = tab(i \leftarrow tab[j])(j \leftarrow r)\}$(aff)
4. $\{tab'[i] = y \wedge tab'[j] = x \wedge tab' = tab(i \leftarrow tab[j])(j \leftarrow tab[i])\} ..; ..; .. \{tab[i] = y \wedge tab[j] = x\}$(seq)(3)(2)(1)
5. $\{tab'[i] = y \wedge tab'[j] = x \wedge tab' = tab(i \leftarrow tab[j])(j \leftarrow tab[i])\} varr; ... \{tab[i] = y \wedge tab[j] = x\}$(let)

Précondition obtenue:

$$tab'[i] = y \wedge tab'[j] = x \wedge tab' = tab(i \leftarrow tab[j])(j \leftarrow tab[i]$$

6. $P \Rightarrow P'$(CQFD)
7. $\{P\} Prog_2 \{Q\}$(mp-pre)

Ex3.

- Variant : m
- Terminaison de l'invariant: $m \geq 0$
- Invariant : $x^n = y^m * p \wedge m \geq 0$

Programme:

```
{
    var m; var y;
    m = n; p = 1; y = x;
    if (x != 0)
        while (m != 0) {
            if (odd(m)) p = p * y;
            else p = p;
            m = m div 2;
            y = y * y;
        }
    else p = 0;
}
```

Q1.

$$\{n \geq 0\} Prog_3 \{p = x^n\}$$

1. $\{x^n = 0\}p = 0\{p = x^n\}$(aff)
2. $\{x^n = y^{2m} * p \wedge m \geq 0\}y = y * y\{x^n = y^m * p \wedge m \geq 0\}$(aff)
3. $\{x^n = y^{2(mdiv2)} * p \wedge m \geq 0\}m = mdiv2\{x^n = y^{2m} * p \wedge m \geq 0\}$(aff)
4. $\{x^n = y^{2(mdiv2)} * p \wedge m \geq 0\}p = p\{x^n = y^{2m} * p \wedge m \geq 0\}$(aff)
5. $\{x^n = y^{2(mdiv2)+1} * p \wedge m \geq 0\}p = p * y\{x^n = y^{2(mdiv2)} * p \wedge m \geq 0\}$(aff)
6. $\{odd(m) \Rightarrow x^n = y^{2(mdiv2)+1} * p \wedge even(m) \Rightarrow x^n = y^{2(mdiv2)}\}if..else..\{x^n = y^{2(mdiv2)} * p \wedge m \geq 0\}$(if)