Composition d'Informatique
Les Principes des Langages de Programmation (INF 321)
Promotion 2011

Corrigé proposé par Eric Goubault et Xavier Rival

12 juillet 2012

1 Types de données et constructeurs

1.a) Définir une classe Java *concrète* `AExpr` qui implémente le type correspondant aux expressions arithmétiques.

```
public enum operator {
 Plus, Times, Div
}

public enum kind {
 Bot,
 Cst,
 Cell,
 Neg,
 Bin
}

class AExpr {
 kind k;
 float val;
 int i;
 int j;
 operator op;
 AExpr l;  // stores the only sub-expr of "-e"
 AExpr r;  // should be null for Neg
}
```

Barême: 2 points

1.b) Définir les constructeurs:

```
AExpr()
AExpr(float n)
AExpr(int i, int j)
AExpr(operator op, lexpr, rexpr)
AExpr(expr)
```

qui rendent respectivement $\bot$, un `AExpr` représentant le flottant n, $c_{i,j}$, l'opération binaire `op` appliqué à une sous-expression gauche `lexpr` et à une sous-expression droite `rexpr`, et le moins appliqué à la sous-expression `expr`.

```
AExpr() {
 k = kind.Bot;
}

AExpr(float f) {
 k = kind.Cst;
 v = f;
}

AExpr(int i, int j) {
 k = kind.Cell;
 this.i = i;
 this.j = j;
}

AExpr(operator op, AExpr lexpr, AExpr rexpr) {
 k = kind.Bin;
 o = op;
 l = lexpr;
 r = rexpr;
}

AExpr(AExpr expr){
 k = kind.Neg;
 l = expr;
}
```

Barême: .5 point par constructeur (2.5 points en tout)

1.c) Définir une classe permettant de stocker une feuille de calcul. On l'appellera `Feuille`.

```
class Feuille{
 AExpr[][] cells;
}
```

Barême: 1 point

2 Sémantique et évaluation

2.a) On considère dans cette question la feuille:

$$\mathcal{D} = \begin{array}{|c|c|}\hline c_{1,1} + c_{0,1} & 3 * c_{1,0} \\ \hline 2 & c_{0,0} + c_{0,1} \\ \hline \end{array}$$

Quelles en sont les cellules définies ? (On donnera pour ce faire le graphe $G_{\rho_{\mathcal{D}}}$).

Les arêtes du graphe sont:

$$\begin{array}{lcl} c_{0,0} & \rightarrow & c_{0,1} \\ c_{0,0} & \rightarrow & c_{1,1} \\ c_{0,1} & \rightarrow & c_{1,0} \\ c_{1,1} & \rightarrow & c_{0,0} \\ c_{1,1} & \rightarrow & c_{0,1} \end{array}$$

Les cellules définies sont donc $c_{1,0}$ et $c_{0,1}$.
Barême: 2 points.

2.b) Écrire la sémantique dénotationnelle $[\![expr]\!]$ des expressions arithmétiques $a \in AExpr$, par récurrence structurelle sur $AExpr$. Expliquer brièvement pourquoi cette récurrence structurelle est bien définie.

- si e n'est pas définie, $[\![e]\!](p) = \bot$
- $[\![\bot]\!](p) = \bot$
- $[\![f]\!](p) = f$
- $[\![c_{i,j}]\!](p) = [\![p(i,j)]\!](p)$
- $[\![-e]\!](p) = -[\![e]\!](p)$
- $[\![e_0 \ o \ e_1]\!](p) = [\![e_0]\!](p) \circ [\![e1]\!](p)$ si $o \in \{+, -, *\}$

La sémantique des expressions non définies est bien définie. Si e est une expression définie, le sous-graphe de dépendance obtenu en ne considérant que les noeuds atteignables à partir des cellules apparaissant dans l'expression est un arbre ; de plus, le sous-graphe défini de la même manière pour chacun des appels récursifs dans l'évaluation de e correspond à un sous arbre strict de l'arbre initial. La récurrence est donc bien fondée.

Barême: 2 points pour la sémantique, 1 point pour l'explication de la correction de cette définition.

2.c) Écrire le code de la méthode dynamique et récursive `Eval` de la classe `AExpr` prenant en argument une feuille p, et permettant d'évaluer l'expression sur laquelle elle s'applique.

```
AExpr Eval(Feuille p) {
 if (Defined(this)) {
   switch(k) {
     case Bot:
       return this;
     case Cst:
       return this;
     case Cell:
       return p.cells[i][j].eval(p);
     case Neg:
       return new AExpr(-1.eval(p).v);
     case Bin:
       switch(o) {
         case Plus:
           return new AExpr(l.eval(p).v+r.eval(p).v);
        case Times:
           return new AExpr(l.eval(p).v*r.eval(p).v);
        case Div:
           float den = r.eval(p).v;
           if (den != 0)
             return new AExpr(l.eval(p).v/den);
           else
             return new AExpr();
       }
   }
 } else {
     return new AExpr();
   }
}
```

Barême: 2.5 points.

2.d) optionnel Comment faire pour se passer de la méthode `Defined` pour programmer `Eval` ?

```
AExpr Eval(boolean [][]dejavu, Feuille p){
 switch(k) {
   case Bot:
     return this;
   case Cst:
     return this;
   case Cell:
     if (dejavu[i][j])
       return new AExpr();
     else {
       dejavu[i][j] = true;
       return p.cells[i][j].eval(p);
     }
   case Neg:
     return new AExpr(-1.eval(p).v);
   case Bin:
     switch(o) {
     case Plus:
       return new AExpr(l.eval(p).v+r.eval(p).v);
     case Times:
       return new AExpr(l.eval(p).v*r.eval(p).v);
     case Div:
       float den = r.eval(p).v;
       if (den != 0)
         return new AExpr(l.eval(p).v/den);
       else
         return new AExpr();
     }
  }
}

AExpr Eval(Feuille p) {
 boolean[][] dejavu = new boolean[p.cells.length][];
 for (int i = 0; i < p.cells.length; i++) {
   dejavu[i] = new boolean[p.cells[i].length];
 }
 return eval(dejavu, p);
}
```

Barême: 2 points.

3 Gestion de la feuille de calcul

3.a) Écrire une méthode dynamique `Compute`, de la classe `Feuille`, qui évalue chaque cellule de la feuille courante.

```
Compute() {
 for (i=0; i < cells.length; i++)
   for (j=0; j < cells[i].length; j++) {
     cells[i][j] = cells[i][j].eval(this);
   }
}
```

Barême: 2 points.

3.b) Que doit donner `Compute` sur $\mathcal{C}$? et sur $\mathcal{D}$?

Sur $\mathcal{C}$, seul $c_{0,0}$ est défini égal à 1, les trois autres cellules sont à bottom.
Sur $\mathcal{D}$, on a vu que seules $c_{0,1}$ et $c_{1,0}$ sont définis, respectivement à 6 et à 2, les deux autres cellules sont à bottom.
Barême: 1 point pour chaque cas (2 points en tout).

3.c) Par quel mécanisme peut-on traiter simplement ces cas de division par zéro ?

Par le mécanisme d'exception, disponible en Java.
Barême: 1 point.

3.d) Dire comment modifier le code d'`Eval` (sans tout réécrire) de la question (2.c) et le code de `Compute` de la question (3.a) afin d'implémenter ce mécanisme.

Par exemple, on peut se souvenir de l'expression de la feuille responsable de la division par zéro (indicatif, non nécessaire pour avoir les points de cette question):

```
class ArithException extends Exception {
  AExpr d;
  ...
}

AExpr Eval(Feuille p) throws ArithException {
 if (Defined(this)) {
   switch(k) {
     case Bot:
       return this;
     case Cst:
       return this;
     case Cell:
       return p.cells[i][j].eval(p);
     case Neg:
       return new AExpr(-l.eval(p).v);
     case Bin:
       switch(o) {
         case Plus:
           return new AExpr(l.eval(p).v+r.eval(p).v);
        case Times:
           return new AExpr(l.eval(p).v*r.eval(p).v);
        case Div:
           AExpr exprden = r.eval(p);
           float den = exprden.v;
           if (den != 0)
             return new AExpr(l.eval(p).v/den);
           else
             throw new ArithException(exprden);
             return new AExpr();
       }
   }
 } else {
     return new AExpr( );
   }
}

Compute() {
 for (i=0; i < cells.length; i++)
```

```
    for (j=0; j < cells[i].length; j++) {
      try {
       cells[i][j] = cells[i][j].eval(this);
      } catch (ArithException e) {
          // Ideally, should pretty print the expression e.d
          System.out.println("Expression au denominateur "+e.d+" est nulle!")
        }
    }
}
```

Barême: 2 points (1 pour la définition d'une exception, même vide, et pour la modification dans `Eval`, 1 pour le `try/catch` de la méthode `Compute`).

4 Méthode alternative d'évaluation

4.a) Quels sont les points fixes du $F^{\mathcal{C}}$ particulier défini ci-dessus ? De même pour $F^{\mathcal{D}}$ la fonctionnelle induite par la feuille $\mathcal{D}$ de la question 2.a.

On a

$$F^{\mathcal{C}} \begin{pmatrix} c_{0,0} \\ c_{0,1} \\ c_{1,0} \\ c_{1,1} \end{pmatrix} = \begin{pmatrix} 1 \\ c_{1,1} \\ \bot \\ c_{0,0} - c_{0,1} \end{pmatrix}$$

Les points fixes de $F^{\mathcal{C}}$ sont donc tels que: $c_{0,0} = 1$, $c_{0,1} = c_{1,1}$, $c_{1,0} = \bot$ et $c_{1,1} = c_{0,0} - c_{0,1} = 1 - c_{1,1}$. Donc on a deux solutions:

- $c_{0,1} = c_{1,1} = c_{1,0} = \bot$ et $c_{0,0} = 1$
- $c_{0,1} = c_{1,1} = \frac{1}{2}$, $c_{0,0} = 1$ et $c_{1,0} = \bot$

On a

$$F^{\mathcal{D}} \begin{pmatrix} c_{0,0} \\ c_{0,1} \\ c_{1,0} \\ c_{1,1} \end{pmatrix} = \begin{pmatrix} c_{1,1} + c_{0,1} \\ 3c_{1,0} \\ 2 \\ c_{0,0} + c_{0,1} \end{pmatrix}$$

Les points fixes de $F^{\mathcal{D}}$ sont donc tels que: $c_{0,0} = c_{1,1} + c_{0,1}$, $c_{0,1} = 3c_{1,0}$, $c_{1,0} = 2$ et $c_{1,1} = c_{0,0} + c_{0,1}$. Donc $c_{0,0} = c_{0,0} + 2c_{0,1} = c_{0,0} + 6c_{1,0} = c_{0,0} + 12$, $c_{0,1} = 6$, $c_{1,0} = 2$ et $c_{1,1} = c_{0,0} + 6$. Donc on n'a qu'une plus petite solution (plus petit point fixe): $c_{0,1} = \bot$, $c_{0,1} = 6$, $c_{1,0} = 2$ et $c_{1,1} = \bot$.

Barême: 1 point par fonctionnelle (2 points en tout). Pour la fonctionnelle $F^{\mathcal{C}}$ un demi-point par cas trouvé (2 points fixes).

4.b) Quel est le plus petit point fixe de $F^{\mathcal{C}}$? de $F^{\mathcal{D}}$?

Pour $F^{\mathcal{C}}$: $c_{0,1} = c_{1,1} = c_{1,0} = \bot$ et $c_{0,0} = 1$. Pour $F^{\mathcal{D}}$: $c_{0,1} = \bot$, $c_{0,1} = 6$, $c_{1,0} = 2$ et $c_{1,1} = \bot$.
Barême: .5 point par plus petit point fixe (1 point en tout).

4.c) Donner les itérées successives du calcul de plus petit point fixe de $F^{\mathcal{C}}$ et de $F^{\mathcal{D}}$ par le théorème de Kleene.

On calcule:

$$F^{\mathcal{C}^1}(\bot) = \begin{pmatrix} 1 \\ \bot \\ \bot \\ \bot \end{pmatrix}$$

$$F^{\mathcal{C}^2}(\bot) = \begin{pmatrix} 1 \\ \bot \\ \bot \\ \bot \end{pmatrix}$$

Donc le plus petit point fixe de $F^{\mathcal{C}}$ est $(1, \bot, \bot, \bot)$.

Maintenant pour $F^{\mathcal{D}}$:

$$F^{\mathcal{D}^1}(\bot) = \begin{pmatrix} \bot \\ \bot \\ 2 \\ \bot \end{pmatrix}$$

$$F^{\mathcal{D}^2}(\bot) = \begin{pmatrix} \bot \\ 6 \\ 2 \\ \bot \end{pmatrix}$$

$$F^{\mathcal{D}^3}(\bot) = \begin{pmatrix} \bot \\ 6 \\ 2 \\ \bot \end{pmatrix}$$

et on atteint le plus petit (et unique) point fixe.

Barême: 1 point par fonctionnelle (2 points en tout).

4.d) Prouver que si $c_{i,j}$ n'est pas défini dans la feuille ρ, alors pour tout $k \in \mathbb{N}$, $(F^k)_{i,j}(\bot) = \bot$, où $(F^k)_{i,j}$ est la composante (i,j) de l'itérée $k$ième de F.

Une cellule $c_{i,j}$ n'est pas définie si elle est dans un cycle de dépendance, ou si elle dépend d'une valeur $\bot$. Dans tous les cas, l'evaluation en $\bot$ de la fonctionnelle correspondant au calcul de $c_{i,j}$, $F_{i,j}$, et même de la fonctionnelle itérée k fois, utilise au moins une valeur à $\bot$ ($c_{i,j}$ même, si dans un cycle de dépendance, sinon une cellule à $\bot$), or les fonctions arithmétiques sont toutes strictes, donc par induction structurelle simple, le résultat est toujours $\bot$.

Barême: 1 point.

4.e) Prouver maintenant que si tous les $c_{i,j}$ sont définis dans la feuille ρ, $F^k(\bot)$, $k \geq 1$, donne la valeur numérique (donc différente de $\bot$, au sens de la sémantique dénotationnelle de la question (2.a)) de l'ensemble C^k des cellules $c_{i,j}$ telles que $k > k_{i',j'}^{c_{i,j}}$ pour toutes cellules $c_{i',j'} \in C^1$.

Soit donc C^1 l'ensemble des cellules $c_{i,j}$ telles que $\rho(c_{i,j})$ est une constante numérique, et de façon générale C^k l'ensemble des cellules $c_{i,j}$ de profondeur inférieure strictement à k par rapport aux cellules de C^1. On effectue une récurrence sur k.

Le cas de base est $k = 1$: l'ensemble des cellules de C^1 est précisément celui calculé par $F^1(\bot)$.

On suppose que l'ensemble des cellules de C^k est calculé par $F^k(\bot)$, et on calcule $F^{k+1}(\bot) = F(F^k(\bot))$. On voit aisément que les cellules $c_{i,j}$ de C^{k+1} sont à profondeur inférieure ou égale à 1 par rapport aux cellules de C^k. Pour les cellules de profondeur 0, ce sont des cellules de C^k donc sont déjà calculées par $F^k(\bot)$, a fortiori par $F^{k+1}(\bot)$ (pour s'en convaincre facilement, on peut utiliser la monotonie de F). Pour les cellules de profondeur 1 par rapport à C^k, elles dépendent directement de la valeur des cellules de C^k, qui sont données par $F^k(\bot)$: ainsi $F^{k+1}(\bot)$ les calcule.

Barême: 1 point.

4.f) En déduire que le plus petit point fixe de la fonctionnelle F est égal à la sémantique dénotationnelle donnée à la question (2.a).

Pour les cellules $c_{i,j}$ non-définies, par la question 4.d), on sait que $F^k_{i,j}(\bot)$ est $\bot$ pour tout k, donc leur valeur dans la sémantique de plus petit point fixe (de F) est $\bot$. Cela correspond à la sémantique dénotationnelle de la question 2.a).

Pour les cellules $c_{i,j}$ définies, par la question 4.e), on sait qu'il existe k tel que $F^k_{i,j}(\bot)$ donne la valeur numérique de $c_{i,j}$, car le graphe de dépendance est fini, donc toutes les profondeurs sont finies. Donc leur valeur est calculée (par la même sémantique que la sémantique dénotationnelle de la question 2.a) par le plus petit point fixe de F.

Barême: 1 point.

4.g) Écrire une version itérative de la méthode *Eval* de la question (2.c), implémentant l'itération de Kleene, que l'on sait donner le même résultat que la méthode récursive de la question (2.c), grâce à la question (4.f).

En fait, on doit d'abord écrire `Compute()` avant `Eval`.

```
AExpr EvalAExpr(Feuille p, Feuille cur) {
// Evaluating expression this, with functional p, under context cur
 AExpr resl, resr;
 switch(k) {
   case Bot:
     return new AExpr();
   case Cst:
     return this;
   case Cell:
     return EvalAExpr(cur.cells[i][j],p);
   case Neg:
     resl = EvalAExpr(l,p);
     if (resl.k == Bot) return new AExpr();
     else return new AExpr(-resl.v);
   case Bin:
     resl = EvalAExpr(l,p);
     relr = EvalAExpr(r,p);
     if (resl.k == Bot) return new AExpr();
     if (resr.k == Bot) return new AExpr();
     switch(e.o) {
       case Plus:
         return new AExpr(resl.v+resr.v);
       case Times:
         return new AExpr(resl.v*.resr.v);
       case Div:
         float den = resr.v;
         if (den != 0)
           return new AExpr(resl.v/den);
         else
           return new AExpr();
     }
 }
}

// Auxiliary functions

static boolean equalAExpr(AExpr e0, AExpr e1) {
// supposing all expressions are allocated
 if (e0.k != e1.k) return false;
```

```
 switch(e0.k) {
   case Bot:
     return true;
   case Cst:
     return (e0.k == e1.k);
   case Cell:
     return ((e0.i == e1.i) && (e0.j == e1.j));
   case Neg:
     return equalAExpr(e0.l, e1.l);
   case Bin:
     if (e0.o != e1.o) return false;
     return (equalAExpr(e0.l, e1.l) && equalAExpr(e0.r, e1.r));
 }
}

static boolean equalFeuille(Feuille cur, Feuille next) {
// supposing they have the same size, and well allocated
  boolean res = true;
  for (int i=0; i<cur.cells.length; i++)
    for (int j=0; j<cur.cells[i].length; j++)
      if (!equalAExpr(cur.cells[i][j],next.cells[i][j]))
        res = false;
  return res;
}

static copycells(AExpr[][] from, AExpr[][] to) {
// supposing same size and well allocated square cell arrays
 for (int i=0; i<from.length; i++)
   for (int j=0; i<from[i].length; j++)
     to.cells[i][j] = from.cells[i][j];
}

Compute() {
  AExpr prev = new AExpr();
  bool unstable = true;
  Feuille cur = new Feuille();
  Feuille next = new Feuille();

  // suppose p.cells contains at least one cell, and that the sheet is square
  cur.cells = new AExpr[cells.length][p.cells[0].length];
  next.cells = new AExpr[cells.length][p.cells[0].length];

  // initializing current iterate to bot
  for (int i=0; i<cells.length; i++)
    for (int j=0; i<cells[i].length; j++)
      cur.cells[i][j] = new AExpr();

  // Kleene iterate
  while (unstable) {
    for (int i=0; i<cells.length; i++)
      for (int j=0; i<cells[i].length; j++)
        next.cells[i][j] = cells[i][j].EvalAExpr(this, cur);

    // test whether fixpoint
    if (equalFeuille(cur, next)) unstable = false;
    else unstable = true;

    // copy iterate k+1 into cur
```

```
    copycells(next.cells, cur.cells);
  }

  // copying result into this
  copycells(cur.cells, cells);
}

AExpr Eval(Feuille p) {
  p.Compute();
  return EvalAexpr(p, p);
}
```

Barême: 2 points dès que la structure de l'itérée de Kleene est correcte (même si le code des fonctions auxiliaires `equalAExpr`, `copycells` etc. est absent). +1 point si plus complet.

4.h) optionnel Soit P le prédicat suivant sur les $a \in AExpr$ et les feuilles ρ: $P(a, \rho)$ est vrai si a est définie dans ρ. Donner les assertions nécessaires à la preuve à la Hoare du code itératif d'`Eval` de la question (4.d), à chaque ligne du code, permettant de prouver

$$\{\neg P(a, \rho)\}\texttt{x=a.Eval(}\rho\texttt{)}\{x = \bot\}$$

Vérifier les assertions de preuve en utilisant les règles de preuve à la Hoare du cours.

Barême: 2 points dès que l'idée est là.