Examen réparti d'ILP

Christian Queinnec

16 mai 2013

Conditions générales

Cet examen est formé d'un unique problème en plusieurs questions auxquelles vous pouvez répondre dans l'ordre qui vous plait.

Le barème est fixé à 20 ; la durée de l'épreuve est de 3 heures. Tous les documents sont autorisés et notamment ceux du cours.

Votre copie sera formée de fichiers textuels que vous laisserez aux endroits spécifiés dans votre espace de travail pour Eclipse. L'espace de travail pour Eclipse sera obligatoirement nommé `workspace` et devra être un sous-répertoire direct de votre répertoire personnel.

À l'exception des clés USB en lecture seule, tous les appareils électroniques communiquants sont prohibés (et donc notamment les téléphones portables). Vos oreilles ne doivent pas être reliées à ces appareils.

L'examen sera corrigé automatiquement sur une batterie de tests. Il est donc nécessaire que vos classes soient compilables et portent les noms imposés. Vous ne pouvez modifier les fichiers qui vous sont fournis et vous devez les installer où prescrit.

Le langage à étendre est obligatoirement ILP4.

Le paquetage Java correspondant sera nommé `fr.upmc.ilp.ilp4ar`. Sera ramassé, à partir de votre *workspace* (situé sous ce nom et directement dans votre répertoire HOME), le seul répertoire nommé `ILP/Java/src/fr/upmc/ilp/ilp4ar/` et tout ce qu'il contient.

Pour vous éviter de la taper à nouveau, voici l'url du site du master (mais il faut, pour y accéder, ne plus passer par le proxy) et celle de l'ARI où se trouvent de nombreuses documentations dont celle de Java :

```
http://www-master.ufr-info-p6.jussieu.fr/site-annuel-courant/
http://www-ari.ufr-info-p6.jussieu.fr/
```

Introduction

Tout comme l'examen de janvier 2013, cet examen s'intéresse aux vecteurs mais dans le cadre exclusif d'ILP4. Il vous sera demandé d'ajouter quelques fonctions pour manipuler les vecteurs puis d'introduire une nouvelle syntaxe pour la création de vecteurs. Cette syntaxe usera de crochets, ainsi, on pourra écrire des programmes tels que :

```
let v = [ 1, 1+1, "ab" ]     // nouvelle syntaxe
in vectorSet(v, 0, vectorLength(v));
   print(vectorGet(v, 1-1)); // imprime 3
   print(vectorGet(v, 4/4)); // imprime 2
   if ( isVector(v) ) {
     vectorSet(v, 6/vectorLength(v), makeVector(3-1, pi));
     print(v); // pourrait imprimer [3, 2, [3.14, 3.14]]
   }
```

Cet examen ne s'intéressera pas à l'impression des vecteurs.

C'est quasiment le même énoncé qu'en janvier 2013 mais cet examen sera, lui, corrigé automatiquement. Pour ce faire, vos classes seront compilées avec ma classe `ProcessTest` (ci-dessous) et toutes mes autres classes d'ILP1 à ILP4 ainsi qu'avec ma batterie de tests incluant ceux d'ILP4. Prenez garde en travaillant sur une question à ne pas compromettre le bon fonctionnement des questions précédentes !

```
package fr.upmc.ilp.ilp4ar.test;

import java.io.IOException;
import java.util.Collection;

import org.junit.Before;
import org.junit.runner.RunWith;
```

```
import fr.upmc.ilp.ilp1.test.AbstractProcessTest;
import fr.upmc.ilp.tool.File;
import fr.upmc.ilp.tool.Parameterized;
import fr.upmc.ilp.tool.Parameterized.Parameters;

@RunWith(value=Parameterized.class)
public class ProcessTest extends fr.upmc.ilp.ilp4.test.ProcessTest {

    public ProcessTest (final File file) {
        super(file);
    }

    @Before
    @Override
    public void setUp () throws IOException {
        this.setProcess(new fr.upmc.ilp.ilp4ar.Process(finder));
        getProcess().setFinder(finder);
        getProcess().setVerbose(options.verbose);
    }

    @Parameters
    public static Collection<File[]> data() throws Exception {
        initializeFromOptions();
        AbstractProcessTest.staticSetUp(samplesDir, "u\\d+-[1234](vector)?");
        return AbstractProcessTest.collectData();
    }
}
```

Question 1 – Interprétation (8 points)

Ajouter à l'interprète les fonctions suivantes, nommées exactement ainsi :

```
makeVector(taille, valeur)
isVector(valeur)
vectorLength(vecteur)
vectorGet(vecteur, index)
vectorSet(vecteur, index, valeur)
```

La fonction nommée `makeVector` prend une taille et une valeur et crée un vecteur de cette taille dont toutes les positions sont occupées par cette valeur. Le prédicat `isVector` ne rend vrai que si son argument est un vecteur. La fonction `vectorLength` retourne la taille de son argument si celui-ci est un vecteur. La fonction `vectorGet` prend un vecteur et un index et rend la valeur stockée à cet index dans le vecteur. La fonction `vectorSet` permet de modifier, dans un vecteur et à un index précis, la valeur stockée.

On rappelle qu'ILP est et doit rester un langage sûr.

La représentation des vecteurs d'ILP devra impérativement être `Object[]` c'est-à-dire des tableaux d'objets et non des instances de la classe `java.util.Vector`.

Livraison

- les fichiers en Java dans `workspace/ILP/Java/src/fr/upmc/ilp/ilp4ar/` et notamment `Process.java` et probablement `VectorStuff.java`.

La notation s'effectuera en compilant puis en lançant ma classe `ProcessTest` sur ma batterie de tests pour ILP4 puis sur les tests relatifs aux fonctions sur les vecteurs.

Question 2 – Compilation (6 points)

On désire maintenant adjoindre ces mêmes fonctions au compilateur vers C. L'impression de vecteurs n'est pas demandée.

Livraison

- les classes Java enrichies avec les méthodes de compilation et le fichier C `templateVector.c` dans `workspace/ILP/Java/src/fr/upmc/ilp/ilp4ar/`, ce fichier contiendra tout le code C nécessaire (aucun autre fichier C n'est autorisé).

La notation s'effectuera en compilant puis en lançant ma classe `ProcessTest` sur ma batterie de tests pour ILP4 puis sur les tests relatifs aux fonctions sur les vecteurs.

Question 3 – Création de vecteur (6 points)

On souhaite disposer d'une syntaxe spécifique pour la création de vecteurs de taille quelconque et dont les divers éléments sont les valeurs des expressions entre crochets (cf. exemple ci-dessus).

Voici la grammaire, nommée `grammar4vector.rnc`, introduisant cette nouvelle syntaxe. Ce fichier est définir, sous ce nom exact, dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4ar/`.

```
include "../../../../../../Grammars/grammar4.rnc"

expression |= vecteur

vecteur = element vecteur {
   expression *
}
```

Écrire alors les classes Java nécessaires pour l'interprétation et la compilation de cette nouvelle syntaxe. Enrichissez éventuellement le fichier C `templateVector.c`.

Livraison

– Le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4ar/`

La notation s'effectuera en compilant puis en lançant ma classe `ProcessTest` sur ma batterie de tests.