Examen final d'ILP
1ère session

Christian Queinnec

15 décembre 2009

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 (téléphones portables). Les oreilles ne doivent pas être reliées à ces appareils.

L'examen sera corrigé à la main, il est donc absolument inutile de s'acharner sur un problème de compilation ou sur des méthodes à contenu informatif faible. Il est beaucoup plus important de rendre aisé, voire plaisant, le travail du correcteur et de lui indiquer, par tout moyen à votre convenance, de manière claire, compréhensible et terminologiquement précise, comment vous surmontez cette épreuve. À ce sujet, vos fichiers n'auront que des lignes de moins de 80 caractères, n'utiliseront que le codage ASCII ou UTF-8 enfin, s'abstiendront de tout caractère de tabulation.

Le langage à étendre est ILP4. Le paquetage Java correspondant à cet examen sera donc nommé `fr.upmc.ilp.ilp4.jsgen`. Sera ramassé, à partir de votre *workspace* (situé sous ce nom directement dans votre HOME), le seul répertoire `ILP/Java/src/fr/upmc/ilp/ilp4/jsgen/`

L'URL du site site du master : `http://www-master.ufr-info-p6.jussieu.fr/site-annuel-courant/`

1 Compilation vers Javascript

On souhaite compiler ILP vers Javascript de façon à pouvoir le faire s'exécuter dans des pages HTML. Voici un rappel bref mais suffisant des caractéristiques de Javascript.

La syntaxe est des constantes est celle de C mais définit en plus les booléens. Les chaînes de caractères, comme en Java, ne peuvent tenir que sur une seule ligne. Il n'y a pas d'entier en Javascript, il n'y a que des flottants (doubles). On écrit donc :

```
1 3.14 "foo" true false
```

Les opérateurs s'écrivent comme en C sauf qu'il n'y a pas d'entier en Javascript et qu'il y a des conversions automatiques de type ! Ainsi

```
1 + false == 1
1 + true == 2
1 + "2"  == "12"
5 / 2 == 2.5
5 % 1.5 == 0.5
```

Les structures de contrôle sont comme en C (Javascript est cependant plus laxiste sur les points-virgules en fin de ligne mais il est plus simple pour l'esprit de toujours les placer). Deux différences syntaxiques : Javascript a une instruction **try-catch-finally** (avec **catch** ou **finally** optionnel) et une définition de fonction avec le mot clé `function`. Ainsi

```
function syracuse (x) {
  try {
    var count = 0;
    while (x > 1) {
      count = count + 1;
      if (x%2) {
        x = 3*x + 1;
```

```
      } else {
        x = x/2;
      }
    }
    return x;
  } catch (pb) {
    throw count;
  } finally {
    globalcount = globalcount + count;
  }
}
globalcount = 0;
/* Calculs */
syracuse(23)              // 1
globalcount               // 15
syracuse(syracuse(23)-1); // 0
globalcount               // 30
```

Une grande différence avec C et Java, visible en deux endroits dans l'exemple précédent : une variable locale à une fonction est introduite avec le mot clé `var`. Toutefois, sa portée est celle du corps entier de la fonction et non celle du bloc où elle est introduite.

Comme en ILP (qui était basé sur Javascript) la simple mention d'une variable non locale la crée avec une portée globale. Toujours comme en ILP, les fonctions et variables globales sont en mutuelle récursion et peuvent être mélangées.

Pour tester simplement vos programmes (et notamment les exemples précédents), vous disposez de la commande `js` (un interprète utilisant le moteur Rhino (écrit en Java et d'ailleurs présent en Java 1.6 et utilisable grâce à la JSR223)) ou le moteur Javascript de votre navigateur.

En `js`, la fonction `print()` est n-aire, imprime un blanc entre deux arguments, un retour à la ligne en fin et ramène une valeur indéfinie. Dans votre navigateur, il faut plutôt utiliser la fonction unaire `document.write()`.

2 Contraintes additionnelles

Vous trouverez ces codes en `/Info/lmd/2009/master/ue/ilp-2009oct/E1/`.

Tout d'abord, voici la classe `ProcessTest` que vous pourrez employer telle quelle :

```
package fr.upmc.ilp.ilp4.jsgen;

import java.util.Collection;

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

import fr.upmc.ilp.tool.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.ProcessTest {

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

    @Before
    @Override
    public void setUp () {
        this.process = new Process();
    }

    @Parameters
    public static Collection<File[]> data() {
        // Sauter u13 (car pas d'entier en js)
```

```
        // sauter u645 (pas une erreur en js)
        AbstractProcessTest.staticSetUp(
            "u(0\\d+"
          + "|1[^3]\\d*"
          + "|[2-5]\\d+"
          + "|6[^4]\\d*|64[^5]\\d*"
          + "|[7-9]\\d+)-[1-4]");
        // Pour un (ou plusieurs) test(s) en particulier:
        //AbstractProcessTest.staticSetUp("u52-2");
        return AbstractProcessTest.data();
    }
}
```

La classe `Process` héritera du `Process` d'ILP4. Dans cette classe, seules les méthodes `compile()` et `runCompiled()` sont intéressantes et à écrire :

```
package fr.upmc.ilp.ilp4.jsgen;
public class Process extends fr.upmc.ilp.ilp4.Process {

    public Process () {
        this.cFile = new File("Grammars" + File.separator
                + "Samples" + File.separator + "theProgram4.js");
        IAST4Factory<CEASTparseException> factory = new CEASTFactory();
        setParser(new CEASTParser(factory));
    }
    ...
}
```

Le compilateur vers Javascript sera écrit en une seule classe implantant l'interface `IAST4visitor`. Les méthodes de compilation ne doivent pas dépendre de l'évaluateur final (`js` ou navigateur). On pourra adopter la forme suivante :

```
package fr.upmc.ilp.ilp4.jsgen;
public class Compiler implements IAST4visitor {

    public static class Data {

        public StringBuffer buffer;
        public ICgenLexicalEnvironment lexenv;
        public ICgenEnvironment common;
        public IDestination destination;

        public Data(final StringBuffer buffer,
                    final ICgenLexicalEnvironment lexenv,
                    final ICgenEnvironment common,
                    final IDestination destination) {
            this.buffer = buffer;
            this.lexenv = lexenv;
            this.common = common;
            this.destination = destination;
        }
    }

    public Object visit(IAST4xxxx iast, Object odata) {
        final Data data = (Data) odata;
        final StringBuffer buffer = data.buffer;
        final ICgenLexicalEnvironment lexenv = data.lexenv;
        final ICgenEnvironment common = data.common;
        final IDestination destination = data.destination;

        ...

        return null;
    }
    ...
}
```

Le tableau suivant relie les méthodes du compilateur à écrire aux programmes de test :

Tests	Méthodes
	visit(IAST4program
	visit(IAST4localBlock
	visit(IAST4functionDefinition
	visit(IAST4globalInvocation
u01 à u06	visit(IAST4constant
u01 à u07	visit(IAST4unaryOperation
u01 à u20	visit(IAST4binaryOperation
u01 à u22	visit(IAST4sequence iast
u01 à u25	visit(IAST4alternative
	visit(IAST4reference
u01 à u36	visit(IAST4unaryBlock
u01 à u48	visit(IAST4primitiveInvocation
u01 à u51	visit(IAST4assignment
u01 à u59025	visit(IAST4while
u01 à u80	visit(IAST4try

Il faut sauter le test `u13` qui ne peut fonctionner du fait qu'il n'y a pas d'arithmétique entière en Javascript ainsi que le test `u645` qui ne provoque pas d'erreur en Javascript.

Question 1 – Page HTML (1 point)

Soit le programme ILP (en pseudo-code) :

```
1+2
```

On suppose compiler ce programme en Javascript inclus dans une page HTML afin d'afficher le résultat dans un navigateur pourvu d'un moteur Javascript. On rappelle que la balise pour introduire du code Javascript est `<script type='text/javascript'>`.

Donnez le contenu de cette page HTML.

Livraison

– un fichier `1+2.html` placé dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/jsgen/`.

Question 2 – Fichier Javascript (1 point)

On désire compiler le même programme vers un fichier Javascript que l'on évaluera avec la commande `js`. Lorsqu'évalué, ce fichier doit imprimer son résultat final. Donnez le contenu de ce fichier.

Livraison

– un fichier `1+2.js` placé dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/jsgen/`.

Question 3 – Définition de fonction (2 points)

Voici le schéma de compilation d'une opération binaire. Comme les noms des opérateurs d'ILP sont identiques à ceux de Javascript, on se dispense (pour simplifier) de l'appel à `common`.

```
        -----------------> d
        expr1 opBin expr2

{ var tmp1;
  -----> tmp1
  expr1;

  var tmp2;
  -----> tmp2
  expr2;

  d (tmp1 opBin tmp2);
}
```

Donnez le schéma de compilation puis l'implantation de la méthode associée pour la compilation d'une définition de fonction (`IAST4functionDefinition`) en suivant les conventions d'ASCII-art utilisées pour le schéma des opérations binaires.

Livraison

- un fichier `Compiler.java` placé dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/jsgen/`. Le schéma de compilation apparaîtra en commentaire global avant la méthode associée.

Question 4 – Invocation de fonction globale (2 points)

Donnez le schéma puis l'implantation de la méthode associée pour la compilation d'une invocation de fonction globale et définie en ILP (`IAST4globalInvocation`) en suivant les conventions d'ASCII-art utilisées précédemment.

Livraison

- un fichier `Compiler.java` placé dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/jsgen/`. Le schéma de compilation apparaîtra en commentaire global avant la méthode associée.

Question 5 – Constante (2 points)

Donnez le schéma puis l'implantation de la méthode associée pour la compilation d'une constante (`IAST4constant`) en suivant les conventions d'ASCII-art utilisées précédemment.

Livraison

- un fichier `Compiler.java` placé dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/jsgen/`. Le schéma de compilation apparaîtra en commentaire global avant la méthode associée.

Question 6 – Séquence (2 points)

Donnez le schéma puis l'implantation de la méthode associée pour la compilation d'une séquence d'expressions (`IAST4sequence`) en suivant les conventions d'ASCII-art utilisées précédemment.

Livraison

- un fichier `Compiler.java` placé dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/jsgen/`. Le schéma de compilation apparaîtra en commentaire global avant la méthode associée.

Question 7 – Alternative (2 points)

Donnez le schéma puis l'implantation de la méthode associée pour la compilation d'une alternative (`IAST4alternative`) en suivant les conventions d'ASCII-art utilisées précédemment.

Livraison

- un fichier `Compiler.java` placé dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/jsgen/`. Le schéma de compilation apparaîtra en commentaire global avant la méthode associée.

Question 8 – Invocation de la primitive `print` (2 points)

Donnez le schéma puis l'implantation de la méthode associée pour la compilation d'une invocation à la primitive `print` (`IAST4primitiveInvocation`) en suivant les conventions d'ASCII-art utilisées précédemment. On indiquera également les ajouts corrélés à la bibliothèque d'exécution si besoin.

Livraison

- un fichier `Compiler.java` placé dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/jsgen/`. Le schéma de compilation apparaîtra en commentaire global avant la méthode associée.

Question 9 – Try-catch-finally (2 points)

Donnez le schéma puis l'implantation de la méthode associée pour la compilation d'une instruction **`try`** (`IAST4try`) en suivant les conventions d'ASCII-art utilisées précédemment.

Livraison

- un fichier `Compiler.java` placé dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/jsgen/`. Le schéma de compilation apparaîtra en commentaire global avant la méthode associée.

Question 10 – Variables (3 points)

Expliquez comment vous traitez la singularité de Javascript en ce qui concerne la portée des variables locales à une fonction.

Livraison

– un fichier `Compiler.java` placé dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/jsgen/`.
Votre réponse apparaîtra en commentaire global avant la méthode associée au bloc local n-aire.

Question 11 – Variables prédéfinies (1 point)

Donnez le schéma de compilation d'une référence à la constante π (`pi`) en suivant les conventions d'ASCII-art utilisées précédemment. On indiquera également les ajouts corrélés à la bibliothèque d'exécution si besoin.

Livraison

– un fichier `Compiler.java` placé dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/jsgen/`.
Votre réponse apparaîtra en commentaire global avant la méthode associée à la référence aux variables.