Examen final d'ILP
2ème session

Christian Queinnec

26 janvier 2010

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 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.local`. 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/local/`

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

1 Une particularité de Javascript

Javascript ne connaît que deux types de variables : les variables globales et les variables locales à une fonction (signalées, en Javascript, par le mot-clé `var`). Il n'y a pas de variable locale à un bloc, la portée d'une variable locale est donc celle du corps de la fonction qui la contient.

On souhaite faire évoluer ILP4 en une nouvelle version, que l'on nommera ILP4local, ne procurant plus les blocs locaux unaire ou n-aire mais un mot clé (que l'on nommera `local`[1]) indiquant qu'une variable est locale à la fonction l'englobant. Ce mot-clé pourra s'utiliser, comme une sorte d'annotation, de la façon suivante (en pseudo-code) :

```
function f1 (x, y) {
   local z = x + y
   return 2 * z
}
function f2 (x, y) {
   z = x + y
   return 2 * (local z)
}
```

Les deux définitions de fonction ci-dessus sont équivalentes. Dans les deux définitions, `z` est une variable locale.

1.1 Difficulté

Le mot-clé `local` est délicat car il introduit un nouveau concept : la notion de variable non initialisée ! Considérez par exemple :

[1] Ce mot a été choisi par symétrie avec le mot-clé `global` que l'on trouve par exemple en PHP et qui indique que la variable ainsi qualifiée n'est pas locale.

```
function f3 (x, y) {
   print z     // Que vaut z ?
   z = x + y
   return 2 * (local z)
}
```

Comme l'on désire conserver la propriété qu'ILP4 est un langage sûr, un appel à la fonction `f3` devra signaler une exception « variable non initialisée ».

Question 1 – Exemple1 (1 point)

Qu'imprime le programme suivant ?

```
function f4 (x, y) {
   if ( x ) {
     t = (local z = x + y)
   } else {
     z = t * y
   }
   return z
}
print f4(1,     2)
print f4(false, 3)
```

Livraison

- un fichier `exemples.txt` placé dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/local/`.

Question 2 – Exemple2 (1 point)

Que peut imprimer le programme suivant ? Explicitez votre réponse.

```
function f5 (x, y) {
   if ( x ) {
     local z = x + y
   } else {
     local z = y
   }
   return z
}
print f5(1, 22)
```

Livraison

- un fichier `exemples.txt` placé dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/local/`.

Question 3 – Exemple3 (1 point)

Que peut imprimer le programme suivant ? Explicitez votre réponse.

```
function f6 (x, y) {
   if ( x ) {
     local z = x + (local y)
   } else {
     local z = y
   }
   return z
}
print f6(11, 2)
```

Livraison

- un fichier `exemples.txt` placé dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/local/`.

Question 4 – Grammaire (1 point)

Donnez les règles de grammaire RelaxNG décrivant le mot-clé `local` sans omettre des les situer relativement à `expression` et `instruction`.

Livraison

- un fichier `grammar4local.rnc` placé dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/local`

Question 5 – AST (2 points)

Une façon d'implanter ILP4local, que nous vous imposons, est d'écrire trois classes particulières modifiant les comportements d'ILP4. Il s'agit de la définition de fonction, de la lecture d'une variable locale et de son affectation.

Écrivez ces trois classes (que l'on nommera `CEASTfunctionDefinition`, `CEASTreference` et `CEASTlocalAssignm` Ces classes seront munies du (ou des) constructeur(s) nécessaire(s).

Livraison

- des fichiers `CEASTfunctionDefinition.java`, `CEASTreference.java` et `CEASTlocalAssignment.java` placé dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/local/`.

Question 6 – Analyse syntaxique (2 points)

Écrivez les méthodes `parse` appropriées pour les trois classes `CEASTfunctionDefinition`, `CEASTreference` et `CEASTlocalAssignment`. Vous fournirez également la fabrique `CEASTFactory` et l'analyseur syntaxique `CEASTParser` associés.

Livraison

- des fichiers `CEASTfunctionDefinition.java`, `CEASTreference.java`, `CEASTlocalAssignment.java`, `CEASTFactory.java` et `CEASTParser.java` placés dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/local/`.

Question 7 – Collecteur (4 points)

Écrivez un visiteur, nommé `LocalCollector`, implantant `IAST4Visitor`, parcourant une définition de fonction et en extrayant l'ensemble des variables locales (lecture ou écriture affublées du mot-clé `local`) présentes dans cette définition.

Livraison

- un fichier `LocalCollector.java` placé dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/loc`

Question 8 – Normalisation (4 points)

La normalisation est une réécriture du programme qui peut être mise à profit pour simplifier l'interprétation et la compilation des programmes d'ILP4local.

Explicitez votre stratégie et écrivez les méthodes `normalize` appropriées pour les trois classes `CEASTfunctionDef` `CEASTreference` et `CEASTlocalAssignment`.

Livraison

- des fichiers `CEASTfunctionDefinition.java`, `CEASTreference.java` et `CEASTlocalAssignment.java` placés dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/local/`.
- Indiquez votre stratégie de normalisation en commentaire de la méthode `normalize` de la définition de fonction.

Question 9 – Évaluation (2 points)

Donnez et explicitez les méthodes d'évaluation correspondant aux classes `CEASTfunctionDefinition`, `CEASTreference` et `CEASTlocalAssignment`.

Livraison

- des fichiers `CEASTfunctionDefinition.java`, `CEASTreference.java` et `CEASTlocalAssignment.java` placés dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/local/`.

Question 10 – Compilation (2 points)

Donnez les schémas de compilation correspondant aux classes `CEASTfunctionDefinition`, `CEASTreference` et `CEASTlocalAssignment`.

Livraison

- des fichiers `CEASTfunctionDefinition.java`, `CEASTreference.java` et `CEASTlocalAssignment.java` placés dans le répertoire `workspace/ILP/Java/src/fr/upmc/ilp/ilp4/local/`. Les schémas de compilation apparaîtront en commentaire global avant les méthodes associées.