Examen final d'ILP

Christian Queinnec

15 décembre 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 et vise à produire un simplificateur. Le paquetage Java correspondant à cet examen sera nommé `fr.upmc.ilp.ilp4.simp`. 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/simp/`

Pour vous éviter de la taper à nouveau, voici l'url du site du master :

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

Simplification

À part l'intégration, aucune autre optimisation n'est effectuée en ILP4. Le but de cet examen est de remédier un peu à cet état. Les simplifications proposées correspondent à des évaluations statiques qui seront réalisées à la préparation du programme.

Question 1 – Simplification d'alternatives (3 points)

Comme en Java, on souhaite que, si la condition d'une alternative est une constante, l'alternative soit simplifiée (c'est-à-dire transformée) en sa conséquence ou en son alternant. Ainsi, le programme de gauche est-il simplifié en celui de droite. Cette transformation a lieu avant toute évaluation (interprétation ou compilation).

```
if false
then print "A"       ==>      print "B"
else print "B"
endif
```

Pour cette question, exceptionnellement, nous vous demandons un logiciel fonctionnant correctement c'est-à-dire les classes `Process`, `ProcessTest` ainsi qu'une classe `SimplifyerTest` (et toutes les autres classes que vous pourriez écrire dans le paquetage `fr.upmc.ilp.ilp4.simp`) mettant, sans erreur, en œuvre votre solution qui devra fonctionner sur tous les tests d'ILP4 ainsi que sur tous vos tests éprouvant votre simplification des alternatives.

Nota: Les schémas de transformation (dans le style des schémas de compilation) ainsi que vos propres remarques apparaîtront en commentaire juste avant les méthodes concernées. Parmi ces remarques, j'attends que vous commentiez votre stratégie de résolution de cette question (vous pouvez aussi jeter un coup d'œil aux questions suivantes) et la justification de l'exhaustivité de vos tests.

Question 2 – Simplification de bloc zéro-aires (1 point)

Lorsqu'un bloc n-aire n'introduit aucune variable locale, il peut être transformé en une séquence. Comment ?

Nota: Lorsque le code est simple, vous pouvez l'adjoindre au commentaire textuel décrivant votre réponse et la transformation que vous proposez.

Question 3 – Simplification de boucles (2 points)

Peut-on, similairement à l'alternative, simplifier une boucle `while` ?

Question 4 – Simplification de séquence (3 points)

Une séquence d'instructions réduite à une seule instruction peut être simplifiée, comment ? Une séquence d'instruction dont la première instruction est une constante peut-elle être simplifiée ? Et si la première instruction est une référence à une variable locale ? Et si la première instruction est une référence à une variable globale ?

Question 5 – Temps (1 point)

Maintenant que vous disposez de quelques simplifications, indiquez quand exactement et comment doit être effectuée cette simplification dans la chaîne des transformations d'AST.

Question 6 – Simplification d'affectations (2 points)

Peut-on simplifier une affectation d'une variable (locale ou globale) à elle-même (du genre $v = v$) ? Comment ?

Question 7 – Simplification d'opérateurs unaires (2 points)

On s'intéresse maintenant aux opérateurs unaires : la négation booléenne (`!`) et la négation arithmétique (`-`). Que peut-on faire ? Peut-on simplifier le programme suivant ? Comment ?

```
if ! false
then print "A"
else print "B"
endif
```

Question 8 – Simplification d'opérateurs binaires (4 points)

On ne s'intéressera qu'à l'opérateur binaire de multiplication et qu'aux seules constantes 0 et 1. Donnez les schémas de simplification correspondant aux multiplications par 0 ou 1. Vous préciserez bien quelles sont les hypothèses s'appliquant à l'autre opérande et comment elles peuvent être vérifiées.

Question 9 – Simplification d'invocations (2 points)

Soit le programme suivant :

```
function foo (x) {
   return 2 * x
}
let v = foo(3)
in print "OK"
```

Expliquez, pas à pas, comment ce programme peut être maximalement simplifié à l'aide des règles précédentes ou de nouvelles règles que vous expliciterez.