Examen deuxième session
« Implantation d'un langage de programmation »
Revision: 1.6

Christian Queinnec

2 septembre 2005

Conditions générales

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

Le contenu du répertoire nommé *workspace/ilp4exam/* dans votre répertoire personnel sera récupéré par l'équipe système de l'ARI-CCE à l'issue de l'épreuve. Toute votre production dans le cadre de cet examen devra y être placée.

Faites très attention aux noms des fichiers demandés. Toute erreur sera nuisible à votre copie ! Veillez également à introduire des commentaires illuminant les correcteurs humains sur vos intentions : cet examen ne sera corrigé que manuellement c'est donc l'intention qui compte plus que le code !

Le barème est fixé à 20, la durée de l'épreuve est de 3 heures. Tous les documents sont autorisés. Le répertoire */Infos/lmd/2004/master/ue/ilp-2004oct/* est toujours accessible.

L'examen doit être impérativement traité en ILP2 (les ajouts des versions suivantes d'ILP posent quelques conflits sémantiques qu'il serait trop complexe d'analyser pendant cet examen. Ces problèmes ne se posent pas en ILP2).

1 Installation

Vous avez 15 minutes pour vous installer c'est-à-dire effectuer les opérations qui suivent. Vous pouvez commencer à lire les fichiers, le reste de l'énoncé ne sera visible qu'à 14h environ. Normalement, il faut trois minutes pour faire les opérations qui suivent.

Pour installer les fichiers nécessaires à votre environnement de travail pendant cet examen, veuillez exécuter le script suivant :

```
/Infos/lmd/2004/master/ue/ilp-2004oct/E/installer-examen.sh
```

Les fichiers et répertoires suivants seront installés dans votre répertoire personnel. Ils respectent la hiérarchie usuelle d'ILP telle que visible en */Infos/lmd/2004/master/ue/ilp-2004oct/ILP/* et ci-dessous rappelée :

`~/.emacs`	*pour lire les .rnc et .xml commodément*
`~/workspace/ilp4exam/C/`	*les bibliotheques C usuelles*
`~/workspace/ilp4exam/Grammars/`	*les grammaires RelaxNG*
`~/workspace/ilp4exam/Grammars/Samples/`	*des exemples*
`~/workspace/ilp4exam/Java/jars/`	*les archives usuelles*
`~/workspace/ilp4exam/Java/src/`	*les classes en Java*
`~/workspace/`	*pour Eclipse*

Lancez `eclipse`. À la question *Select a workspace*, répondez OK. Dans la fenêtre qui s'ouvre, cliquez sur *Workbench*. À partir du menu *File*, *New Project*, *Next* (car le type *Java Project* est déjà sélectionné), indiquez le nom du projet qui doit être `ilp4exam`. Eclipse doit alors découvrir que le projet existe déjà (si ce n'est pas le cas, c'est que vous n'avez pas dû donner le nom attendu correct à savoir `ilp4exam`), cliquez alors sur *Finish*. À la question *Confirm Perspective Switch*, répondez Yes.

Ensuite, sélectionnez le projet à gauche, puis avec le menu contextuel (bouton de droite) choisissez *Properties*, onglet *Java Build path* puis onglet *Libraries*, ajoutez, grâce au bouton *add external jars*, les archives correspondant à `isorelax`, `jing`, `junit`, `saxon`, `trang`, `velocity`, `xercesImpl`, `xml-apis` et `xmlunit`. Comme l'indique le s de *jars*, vous pouvez les ajouter toutes en une seule fois (la touche *SHIFT* permet de sélectionner multiplement). En cliquant sur OK, eclipse recompile le projet tout entier normalement sans erreur.

Vous pouvez écrire vos programmes avec Eclipse ou Emacs (ou tout autre moyen à votre convenance). Les fichiers que vous aurez à créer seront, ainsi qu'indiqué question par question, à placer sous le répertoire `workspace/ilp4exam/`. Vos classes Java seront compilées en mode Java 1.4.

Pour septembre, comme Eclipse (en `/usr/local/eclipse/eclipse` *n'a pas été réinstallé, il a gardé trace de votre espace de travail pendant le semestre. Choisissez* `/workspace/` *comme nouvel espace de travail. Les archives sont déjà sélectionnées.*

2 Boucles anonymes

Le but de ce problème est de procurer, dans les boucles, les instructions de sortie de boucle (usuellement notée `break` en C et en Java) et de passage à l'itération suivante (usuellement notée `continue` en C et en Java). La boucle a été introduite dès ILP2. En terme de syntaxe concrète, nous noterons ces nouvelles instructions (comme en Perl) : `last` et `next`. Ces deux nouvelles instructions ne sont relatives qu'à la boucle immédiatement englobante. Ainsi, avec une syntaxe concrète à la C :

```
{ i = 1;
  while ( i < 20 ) {
    i = i + 1;
    if ( i % 2 == 0 ) {
      next;
    }
    print i;
    if ( i > 10 ) {
      last;
    }
  }
}    // Ce programme imprime 357911
```

Le langage ILP2esc correspond au langage ILP2 enrichi de ces deux nouvelles instructions. Un certain nombre de programmes XML (de suffixe `-2esc.xml`) utilisant ces nouvelles instructions sont disponibles en `Grammars/Samples/` si besoin.

Les programmes Java demandés dans cette question devront appartenir au paquetage `fr.upmc.ilp.ilp2esc`.

Question 1

Écrire le complément de grammaire correspondant au langage ILP2esc en RelaxNG compact (ce complément inclut donc `grammar2.rnc`). Les programmes d'ILP2esc (ainsi que tous les programmes que l'on peut écrire en ILP2) devront pouvoir être validés vis-à-vis de cette nouvelle grammaire.

Les programmes donnés en exemples utilisent, respectivement, les éléments XML `boucleAvecEchappement`, `suivant` et `dernier` pour la boucle, l'instruction `next` et `last`.

Livraison

– le fichier *grammar2esc.rnc*

Notation sur 2 points

– 2 points si votre grammaire reconnaît bien, en plus de ses capacités antérieures, les deux nouvelles instructions `next` et `last`.

Question 2

Écrire un nouvel analyseur syntaxique nommé `CEASTparser` (dans le paquetage `fr.upmc.ilp.ilp2esc`) lisant les programmes d'ILP2esc. Ce nouvel analyseur héritera de `fr.upmc.ilp.ilp2.CEASTparser` afin de ne s'intéresser qu'aux seules instructions concernées et ainsi d'être court.

On ne se souciera pas à ce niveau de vérifier que les instructions `next` et `last` sont bien incluses dans une boucle.

Livraison

– les fichiers *fr/upmc/ilp/ilp2esc/CEASTparser.java*

Notation sur 3 points

– 3 points Si votre analyseur répond à la question.

Question 3

On suppose que les classes des deux nouvelles instructions sont `CEASTlast` et `CEASTnext` (dans le paquetage `fr.upmc.ilp.ilp2esc`). L'implantation de la boucle sera modifiée et deviendra `CEASTwhileWithEscape` (dans le même paquetage).

Écrire les méthodes d'interprétation (méthode `eval`) pour ces trois classes.

Indice : comme, lorsque l'on interprète `next` ou `last`, on ne sait pas où se trouve la boucle en question et plutôt que d'introduire un nouvel environnement recensant les boucles courantes, on pourra utiliser des exceptions.

Remarque : les boucles anonymes sont un exemple de caractéristiques langagières plus délicates à interpréter qu'à compiler (dans le cadre d'ILP2).

Livraison

– les fichiers *fr/upmc/ilp/ilp2esc/CEASTlast.java* et *fr/upmc/ilp/ilp2esc/CEASTnext.java* et *fr/upmc/ilp/ilp2esc/CEAS* et toute autre classe nécessaire.

Notation sur 4 points

– 4 points si vos classes permettent d'interpréter les exemples fournis.

Question 4

Écrire les méthodes de compilation vers C (méthode `compile_instruction`) nécessaires pour compiler ILP2esc. Pensez à décrire par un commentaire adapté la forme du code engendré (c'est plus compréhensible que le code Java brut).

Livraison

– les fichiers *fr/upmc/ilp/ilp2esc/CEASTlast.java* et *fr/upmc/ilp/ilp2esc/CEASTnext.java* et *fr/upmc/ilp/ilp2esc/CEAS*
et toute autre classe nécessaire.

Notation sur 3 points

– 3 points si vos classes permettent de compiler les exemples fournis.

3 Boucles nommées

Les instructions `next` et `last` précédentes ne permettaient de contrôler que la boucle courante. Si les boucles étaient nommées, il serait possible de sortir de telle boucle ou d'itérer telle autre boucle. On va donc étendre les instructions `next`, `last` et `boucleAvecEchappement` pour prendre (optionnellement) une étiquette les nommant. En XML, cette étiquette figurera dans un attribut dénommé `label`. En syntaxe concrète (en Java par exemple), on écrirait :

```
{ i = 1;
  A: while ( i < 10 ) {
       j = 0;
       i = i + 1;
       B: while ( j < 10 ) {
            j = j + 1;
            if ( (j % 2) == 0 ) {
              next; // équivalent à next B
            };
            print j;
            if ( j == i*i ) {
              last A;
            };
       }
} // imprime 1357913579
```

Le langage ILP2escl correspond au langage ILP2esc étendu avec ces nouvelles instructions prenant des étiquettes. Vous trouverez des exemples de tels programmes en `Grammars/Samples/e*-2escl.xml` si besoin. Les programmes demandés appartiendront au paquetage `fr.upmc.ilp.ilp2escl`. De nombreuses fonctionnalités pourront être héritées du paquetage `fr.upmc.ilp.ilp2esc`.

Question 5

Écrire le complément de grammaire correspondant au langage ILP2escl en RelaxNG compact (ce complément inclut donc `grammar2esc.rnc`). Les programmes d'ILP2esc (ainsi que tous les programmes que l'on peut écrire en ILP2) devront pouvoir être validés vis-à-vis de cette nouvelle grammaire.

Les programmes donnés en exemples utilisent encore les éléments `boucleAvecEchappement`, `suivant` et `dernier`.

Livraison

– le fichier *grammar2escl.rnc*

Notation sur 1 point

– 1 point si votre grammaire admet maintenant des étiquettes.

Question 6

Écrire un nouvel analyseur syntaxique nommé `fr.upmc.ilp.ilp2escl.CEASTparser` lisant les programmes d'ILP2escl. Ce nouvel analyseur héritera de `fr.upmc.ilp.ilp2esc.CEASTparser` afin de ne s'intéresser qu'aux seules instructions concernées.

On ne se souciera toujours pas à ce niveau de vérifier que les instructions `next` et `last` sont bien incluses dans une boucle.

Livraison

– les fichiers *fr/upmc/ilp/ilp2escl/CEASTparser.java*

Notation sur 2 points

– 2 points Si votre analyseur répond à la question.

Question 7

On suppose que les classes des deux nouvelles instructions sont `CEASTlast` et `CEASTnext` (dans le paquetage `fr.upmc.ilp.ilp2escl`). L'implantation de la boucle sera modifiée et deviendra `CEASTwhileWithEscape` (dans le même paquetage).

Écrire les méthodes d'interprétation (méthode `eval`) pour ces trois classes.

Livraison

– les fichiers *fr/upmc/ilp/ilp2escl/CEASTlast.java* et *fr/upmc/ilp/ilp2escl/CEASTnext.java* et *fr/upmc/ilp/ilp2escl/CEA* et toute autre classe nécessaire.

Notation sur 1 point

– 1 point si vos classes permettent d'interpréter les exemples fournis.

Question 8

Écrire les méthodes de compilation vers C (méthode `compile_instruction`) nécessaires pour compiler ILP2escl. Pensez à décrire par un commentaire adapté la forme du code engendré (c'est plus compréhensible que le code Java brut).

On rappelle que les boucles nommées n'existent pas en C. Indice : on pourra utiliser l'instruction `goto` de C. Voici un petit exemple pour vous en rappeler la syntaxe :

```
    while ( 1 ) {
      if ( j>5 ) {
        goto FIN;
      };
      faire(quelquechose);
    }
FIN:
    printf("Je suis sorti!");
```

Livraison

– les fichiers *fr/upmc/ilp/ilp2escl/CEASTlast.java* et *fr/upmc/ilp/ilp2escl/CEASTnext.java* et *fr/upmc/ilp/ilp2escl/CEA* et toute autre classe nécessaire.

Notation sur 4 points

- 4 points si vos classes permettent de compiler les exemples fournis.