Examen partiel d'ILP

Jacques Malenfant

2 novembre 2007

Directives

- Le contrôle dure 2h00.
- Tous les documents sont autorisés, et notament ceux du cours. Vous disposez à ce titre d'un ordinateur avec accès http aux documents de cours et avec ECLIPSE pour vous permettre de rédiger les parties code de l'examen.
- Outre l'ordinateur qui vous est fourni, tous les appareils électroniques sont **prohibés** (y compris les téléphones portables, les assistants numériques personnels et les agendas électroniques).
- Le barème total est fixé à 20.
- Votre copie sera formée d'un fichier texte (ISO-Latin 1 ou 15) `PartielILP.txt` et de fichiers de programme Java que vous laisserez dans votre répertoire principal à la fin de l'examen ; ces fichiers seront formés de lignes ne dépassant pas 78 caractères pour en faciliter la lecture.

Le problème : alternatives entières à trois voies.

Fortran IV, un langage qui n'offrait pas d'instructions structurées comme les alternatives et les boucles d'ILP1, affectionnait toutes les variantes de sauts de type *goto* à des étiquettes posées sur des instructions. Il proposait en particulier l'alternative arithmétique (« *arithmetic-IF* ») à trois voies suivante :

```
IF (<expression>) etiq1, etiq2, etiq3
```

L'expression est évaluée, et le programme saute à l'étiquette `etiq1` si le résultat de l'expression est négatif, à `etiq2` s'il est nul, et à `etiq3` s'il est positif.

Nous vous demandons d'introduire en ILP1 une version structurée de cette alternative arithmétique (il n'est bien entendu pas question de revenir à des « *gotos* »). Il s'agira donc d'une instruction d'alternative qui aura trois alternants : l'un si la valeur de la condition est négative, le second si elle est nulle et le troisième si elle est positive.

Question 1

(2 points)

Proposez quelques variantes de syntaxe concrète pour cette nouvelle instruction, telle que vous pourriez l'utiliser dans un langage comme Java. Vous en sélectionnerez une pour la suite, en argumentant votre choix.

Livraison

Un réponse, sous le titre « Question 1 », au format texte dans le fichier `PartielILP.txt`.

Réponse :

On peut partir de l'idée de l'alternative et obtenir :

```
si-arithmetique (<condition>)
si-negatif {
    <instructions>
} si-nul {
    <instructions>
} si-positif {
    <instructions>
}
```

En s'inspirant du `switch`, on pourrait aussi avoir :

```
calculer (<expression>) {
cas-negatif : <instructions>
cas-nul : <instructions>
cas-positif : <instructions>
}
```

Question 2

(3 points)

Proposez ensuite une syntaxe XML pour intégrer cette instruction à ILP1 et définissez l'extension nécessaire à la grammaire d'ILP1.

Livraison

Une réponse, sous le titre «Question 2», au format texte dans le fichier `PartielILP.txt` donnant et expliquant en quelques mots votre syntaxe XML et un fichier `grammar1-partiel.rnc`.

Réponse :

Syntaxe XML :

```
<si-arithmetique>
  <condition>...</condition>
  <si-negatif>...</si-negatif>
  <si-nul>...</si-nul>
  <si-positif>...</si-positif>
</si-arithmetique>
```

```
include "grammar1.rnc"

instruction |= si-arithmetique

si-arithmetique = element si-arithmetique {
element condition  { expression },
element si-negatif { instruction+ },
```

```
element si-nul     { instruction+ },
element si-positif { instruction+ }
}
```

Question 3

(3 points)

Décrivez les modifications à faire pour réaliser l'analyse des programmes utilisant cette nouvelle instruction.

Livraison

Une réponse, sous le titre « Question 3 », au format texte dans le fichier `PartielILP.txt`, en incluant lorsque nécessaire les lignes de code à ajouter pour traiter le problème, et un fichier contenant votre classe `EASTParser`. Le fichier de la classe d'`EAST` implantant le noeud d'arbre de syntaxe abstraite sera à rendre lors de la question suivante.

Réponse :

Il faut créer une nouvelle interface de fabrique et une classe implantant cette interface en ajoutant la méthode pour les noeuds de type `EASTsiArithmetique`.

```
package fr.upmc.ilp.partiel2007.eval;

import fr.upmc.ilp.ilp1.interfaces.IAST;

public interface IEASTFactory<Exc extends Exception> extends fr.upmc.ilp.ilp1.eval.IE

    IAST newSiArithmetique(IAST cond, IAST siNegatif, IAST siNul, IAST siPositif);

}
```

```
package fr.upmc.ilp.partiel2007.eval;

import fr.upmc.ilp.ilp1.eval.EASTException;
import fr.upmc.ilp.ilp1.interfaces.IAST;

public class EASTFactory extends fr.upmc.ilp.ilp1.eval.EASTFactory implements
        IEASTFactory<EASTException> {

    public IAST newSiArithmetique(IAST cond, IAST siNegatif, IAST siNul,
            IAST siPositif) {
        return new EASTsiArithmetique(cond, siNegatif, siNul, siPositif);
    }

}
```

```
package fr.upmc.ilp.partiel2007.eval;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
```

```
import org.w3c.dom.NodeList;

import fr.upmc.ilp.ilp1.eval.EASTException;
import fr.upmc.ilp.ilp1.interfaces.IAST;

public class EASTParser extends fr.upmc.ilp.ilp1.eval.EASTParser {

    protected IEASTFactory<EASTException> factory;

    public EASTParser(IEASTFactory<EASTException> factory) {
        super(factory);
        this.factory = factory ;
    }

    @Override
    public IAST parse(Node n) throws EASTException {
        try {
            switch ( n.getNodeType() ) {

            case Node.DOCUMENT_NODE: {
                final Document d = (Document) n;
                return this.parse(d.getDocumentElement());
            }

            case Node.ELEMENT_NODE: {
                final Element e = (Element) n;
                final NodeList nl = e.getChildNodes();
                final String name = e.getTagName();

                if ( "si-arithmetique".equals(name) ) {
                    final IAST cond = findThenParseChild(nl, "condition");
                    final IAST siNegatif = findThenParseChild(nl, "si-negatif");
                    final IAST siNul = findThenParseChild(nl, "si-nul");
                    final IAST siPositif = findThenParseChild(nl, "si-positif");
                    return factory.newSiArithmetique(cond, siNegatif, siNul, siPosit
                } else if ( "si-negatif".equals(name) ) {
                    return factory.newSequence(this.parseList(nl));
                } else if ( "si-nul".equals(name) ) {
                    return factory.newSequence(this.parseList(nl));
                } else if ( "si-positif".equals(name) ) {
                    return factory.newSequence(this.parseList(nl));
                } else {
                    return super.parse(n);
                }
            }

            default: {
                final String msg = "Unknown node type: " + n.getNodeName();
                return factory.throwParseException(msg);
            }
            }
        } catch (final EASTException e) {
            throw e;
        } catch (final Exception e) {
```

```
            throw new EASTException(e);
        }

    }

} // class fr.upmc.ilp.partiel2007.eval.EASTParser
```

Question 4

(5 points)

Définissez l'interprétation de la nouvelle instruction.

Livraison

Un fichier contenant la classe Java d'`EAST` complète implantant le noeud d'arbre de syntaxe abstraite, y compris sa méthode `eval` pleinement définie.

Réponse :

```
package fr.upmc.ilp.partiel2007.eval;

import java.math.BigInteger;

import fr.upmc.ilp.ilp1.eval.EAST;
import fr.upmc.ilp.ilp1.interfaces.IAST;
import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp1.runtime.ICommon;
import fr.upmc.ilp.ilp1.runtime.ILexicalEnvironment;
import fr.upmc.ilp.partiel2007.interfaces.IASTsiArithmetique;

public class EASTsiArithmetique extends EAST implements IASTsiArithmetique {

    protected EAST condition ;
    protected EAST siNegatif ;
    protected EAST siNul ;
    protected EAST siPositif ;

    public EASTsiArithmetique(IAST condition, IAST siNegatif, IAST siNul, IAST siPos
    {
        this.condition = (EAST) condition ;
        this.siNegatif = (EAST) siNegatif ;
        this.siNul = (EAST) siNul ;
        this.siPositif =  (EAST) siPositif ;
    }

    @Override
    public Object eval(ILexicalEnvironment lexenv, ICommon common)
            throws EvaluationException
    {
        Object res = null ;
        Object resCond = condition.eval(lexenv, common) ;
```

```
        if (resCond instanceof BigInteger) {
            switch (((BigInteger) resCond).compareTo(BigInteger.ZERO)) {
            case -1:
                res = siNegatif.eval(lexenv, common) ;
                break;
            case 0 :
                res = siNul.eval(lexenv, common) ;
                break ;
            case 1 :
                res = siPositif.eval(lexenv, common) ;
                break ;
            default:
                throw new EvaluationException(
                        "Résultat de comparaison différent de -1, 0, 1 dans un si ari
                        "ne devrait jamais se produire.") ;
            }
        } else {
            throw new EvaluationException(
                        "Expression non-entiere dans un si arithmétique : " + resCond
        }
        return res ;
    }

    public IAST getCondition() { return condition ; }
    public IAST getSiNegatif() { return siNegatif ; }
    public IAST getSiNul()     { return siNul ;     }
    public IAST getSiPositif() { return siPositif ; }

} // class fr.upmc.ilp.partiel2007.eval.EASTsiArtihmetique
```

Question 5

(5 points)

Donnez le schéma de compilation pour la nouvelle instruction (c'est-à-dire un schéma similaire à ceux que nous vous avons présentés en cours pour les instructions d'ILP1). Définissez ensuite la procédure pour générer le code C.

Vous pouvez supposer que la variable buffer dans fr.upmc.ilp.ilp1.cgen.Cgenerator est 'protected' plutôt que 'private'.

Livraison

Un fichier texte `q5.txt` donnant le schéma de compilation et la définition de la méthode `generate` pour la nouvelle instruction.

Réponse :

```
package fr.upmc.ilp.partiel2007.cgen;

import fr.upmc.ilp.ilp1.cgen.CgenerationException;
import fr.upmc.ilp.ilp1.cgen.ICgenEnvironment;
import fr.upmc.ilp.ilp1.cgen.ICgenLexicalEnvironment;
import fr.upmc.ilp.ilp1.interfaces.IAST;
```

```
import fr.upmc.ilp.partiel2007.interfaces.IASTsiArithmetique;

public class Cgenerator extends fr.upmc.ilp.ilp1.cgen.Cgenerator {

    public Cgenerator(ICgenEnvironment common) {
        super(common);
    }

    @Override
    protected void analyze(
        IAST iast,
        ICgenLexicalEnvironment lexenv,
        ICgenEnvironment common,
        String destination
        ) throws CgenerationException
    {
        if (iast instanceof IASTsiArithmetique) {
            this.generate((IASTsiArithmetique)iast, lexenv, common, destination);
        } else {
            super.analyze(iast, lexenv, common, destination);
        }
    }

    private void generate(
        IASTsiArithmetique iast,
        ICgenLexicalEnvironment lexenv,
        ICgenEnvironment common,
        String destination
        ) throws CgenerationException
    {
        // [ -> d] siArithmetique
        //
        // {
        //     int c = [ -> ] condition ;
        //     if (c < 0) {
        //         [ -> d ] siNegatif
        //     } else if (c == 0) {
        //         [ -> d ] siNul
        //     } else if (c > 0) {
        //         [ -> d ] siPositif
        //     } else {
        //         erreur
        //     }
        // }
        buffer.append("{ int c ;\n");
        buffer.append("  ILP_Object condition = ");
        analyzeExpression(iast.getCondition(), lexenv, common);
        buffer.append(" ;\n");
        buffer.append("  if (!ILP_isInteger(condition)) {\n");
        buffer.append("    printf(\"Condition is not an integer\") ;\n");
        buffer.append("    exit(1) ;\n");
        buffer.append("  }\n");
        buffer.append("  c = condition->_content.asInteger ;\n");
        buffer.append("  if (c < 0) {\n");
```

```
        analyzeInstruction(iast.getSiNegatif(), lexenv, common, destination) ;
        buffer.append("  } else if (c == 0) {\n") ;
        analyzeInstruction(iast.getSiNul(), lexenv, common, destination) ;
        buffer.append("  } else if (c > 0) {\n") ;
        analyzeInstruction(iast.getSiPositif(), lexenv, common, destination) ;
        buffer.append("  } else {") ;
        buffer.append("    printf(\"Not in the three way comparison, should never occ
        buffer.append("    exit(1) ;\n") ;
        buffer.append("  }\n") ;
        buffer.append("}\n") ;
    }

}
```

Question 6

(2 points)

Dans les questions précédentes, nous avons fait l'hypothèse que l'alternative arithmétique fonctionne sur une expression à résultat entier. Expliquez s'il est possible de généraliser cette instruction et si oui, comment on peut l'étendre pour traiter les autres types de base d'ILP1 (réel, booléen, chaîne de caractères). Faites ressortir les difficultés éventuelles selon les types de données.

Livraison

Une réponse, sous le titre « Question 6 », au format texte dans le fichier `PartielILP.txt`.

Réponse :

Pour les réels, il y a une interprétation immédiate, mais ce serait bien si les étudiants mentionnaient que le test égal à 0 pour les réels est fragilisé par les erreurs de calculs sur les réels.

Pour les chaînes de caractères, une comparaison à la chaîne nulle n'a guère de sens. Par contre, on peut trouver une interprétation un peu plus compliquée fondée sur la comparaison des chaînes de caractères donnant plus petit, égal et plus grand. Il se trouve que les fonctions de comparaison de chaînes à la C donne déjà un résultat qui est -1, 0 ou 1. Il ne s'agit pas de calcul cependant, mais bien de comparaison.

Pour les booléens, il n'y a guère que deux voies possibles, donc on peut s'en remettre à l'alternative classique.