UFR 919 Informatique – Master Informatique

Spécialité STL – UE MI016 – ILP

TME4 — Introduction d'un nouveau trait

Jacques Malenfant, Aurélien Moreau, Christian Queinnec

Objectif : Ajout du mot-clé `aMoinsQue`

Buts

— Ajouter un nouveau mot-clé
— Mesurer les conséquences des choix d'implantation

Pour ce TME, vous pouvez choisir d'étendre ILP1 ou ILP2.

1 Ajouter la construction aMoinsQue

La construction `aMoinsQue` *condition corps* a pour sémantique d'évaluer la condition. Si le résultat est faux, les instructions du corps sont évaluées comme une séquence. On trouve souvent cette construction, notamment en Perl, sous le nom de `unless`.

À côté de la technique suggérée dans le mémento en `http://www-master.ufr-info-p6.jussieu.fr/site-annuel-courant/Ext/queinnec/ILP/refcard.pdf`, il existe au moins trois autres façons d'opérer si l'on considère le processus menant du fichier XML à l'AST final. Ω

```
1 fichier XML -(1)-> DOM -(2)-> AST -(3)-> AST
```

La variante (1) est un hack moche. La variante (2) est plus propre mais tout comme la première variante, si l'on change de syntaxe d'entrée, il faudra la réécrire. La variante (3) est insensible aux changements de syntaxe et correspond plus aux techniques des compilateurs modernes (qui réécrivent les AST de façon à minimise le nombre de mot-clés restant à considérer (notion de *core syntax*)).

1.1 Travail à réaliser

Vous commencerez par écrire la nouvelle grammaire, nommée `grammar2-tme4.rnc`, définissant `aMoinsQue`. Vous écrirez alors au moins deux programmes de tests (dont le nom sera suffixé par `-2tme4.xml` et une classe `ProcessTest` pour tester votre extension d'ILP2 passe toujours tous les tests d'ILP2 et les nouveaux tests d'`aMoinsQue`.

***Pour enseignant*:**
— L'instruction `aMoinsQue` n'est qu'une alternative sans conséquent dont la condition est inversée. On doit donc pouvoir l'implanter en utilisant tout ce qui a été développé pour l'alternative.
— Les deux autres solutions sont :

1. L'interception de l'élément `aMoinsQue` dès le parcours de l'arbre DOM pour générer une alternative avec la condition inversée (grâce à l'opérateur `"!"`).
2. Le remplacement dans l'arbre de syntaxe abstraite de l'objet représentant l'instruction `aMoinsQue` par un objet instance de la classe `CEASTalternative` avant l'interprétation et la compilation. Ceci nécessite d'écrire un parcours de l'AST qui va transformer l'AST de telle manière à éliminer toutes les instances de `CEASTaMoinsQue`.

***Pour enseignant*:**

1. Pour la solution interception, seules les étapes 1, 2, 3, 7 et 11 seront nécessaires.
2. Pour la solution avec réécriture de l'AST, il faut ajouter les étapes 4, 5 et 6.
3. Pour la solution de l'implantation de bout-en-bout :
   — L'étape 8 : pas de modification du runtime.
   — L'étape 10 : pas de modification de la bibliothèque C, tout est là grâce à l'alternative.

***Pour enseignant*:** Étape 1 et 2 – On étend la grammaire pour ajouter la nouvelle instruction :

```
include "grammar2.rnc"

expression |= aMoinsQue

aMoinsQue = element aMoinsQue {
  element condition {expression},
  element corps {instruction+}
}
```

Étape 3 – On écrit un programme de test :

```
<?xml version='1.0' encoding='ISO-8859-15' ?>
<!--

;;; $Id$

(begin
  (unless #f
    (print "OK") )
  700 )

;;; end of u700-2tme5a.scm

 -->
<programme2
><sequence
><aMoinsQue
><condition
><booleen valeur='false'
/></condition
```

```
><corps
><invocationPrimitive fonction='print'
><chaine
>OK</chaine
></invocationPrimitive
></corps
></aMoinsQue
><entier valeur='700'
/></sequence
></programme2
>
```

Pour la première solution, il suffit d'étendre la classe `CEASTParser` pour traiter le nouvel élément `aMoinsQue` selon l'étape 7 :

```
package fr.upmc.ilp.ilp2tme4.ast;

import org.w3c.dom.Element;
import org.w3c.dom.Node;
import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.IAST2;
import fr.upmc.ilp.ilp2tme4.ast.CEASTaMoinsQue;

public class CEASTParser extends fr.upmc.ilp.ilp2.ast.CEASTParser {

    public CEASTParser(CEASTFactory factory) {
        super(factory);
    }

    @Override
    public IAST2<CEASTparseException> parse(Node n) throws CEASTparseException {
        if (n.getNodeType() == Node.ELEMENT_NODE) {
            Element e = (Element) n;
            String name = e.getTagName();
            if ("aMoinsQue".equals(name)) {
                return CEASTaMoinsQue.parse(e, this);
            } else {
                return super.parse(n);
            }
        } else {
            return super.parse(n);
        }
    }

}
```

Pour la seconde solution, on garde ce qui a été fait aux étapes 1 à 3 de la solution précédente et on introduit à l'étape 4 l'interface `IAST2aMoinsQue` pour le nouvelle instruction :

```
package fr.upmc.ilp.ilp2tme4.interfaces;

import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.IAST2;
import fr.upmc.ilp.ilp2.interfaces.IAST2instruction;
```

```
6
7  public interface IAST2aMoinsQue extends IAST2instruction<CEASTparseException> {
8      public IAST2<CEASTparseException> getCondition();
9      public IAST2<CEASTparseException> getCorps();
10 }
```

Nous ajoutons également l'interface `IAST2transformeAMQ` (pour transforme A Moins Que) pour définir la signature de la méthode de parcours de l'AST qui va remplacer l'instance de `CEASTaMoinsQue` par une instance de `CEASTalternative` :

```
1  package fr.upmc.ilp.ilp2tme4b.interfaces;
2
3  import fr.upmc.ilp.ilp2.ast.CEASTparseException;
4  import fr.upmc.ilp.ilp2.interfaces.IAST2;
5
6  public interface IAST2transformeAMQ {
7
8      IAST2<CEASTparseException> transformeAMQ() ;
9
10 }
```

À l'étape 5, on introduit d'abord la classe `CEASTaMoinsQue` pour la nouvelle instruction :

```
1  package fr.upmc.ilp.ilp2tme4b.ast;
2
3  import java.util.Set;
4
5  import org.w3c.dom.Element;
6  import org.w3c.dom.NodeList;
7
8  import fr.upmc.ilp.ilp1.cgen.CgenerationException;
9  import fr.upmc.ilp.ilp1.runtime.EvaluationException;
10 import fr.upmc.ilp.ilp2.ast.CEASTexpression;
11 import fr.upmc.ilp.ilp2.ast.CEASTinstruction;
12 import fr.upmc.ilp.ilp2.ast.CEASTparseException;
13 import fr.upmc.ilp.ilp2.interfaces.IAST2;
14 import fr.upmc.ilp.ilp2.interfaces.IAST2expression;
15 import fr.upmc.ilp.ilp2.interfaces.IAST2instruction;
16 import fr.upmc.ilp.ilp2.interfaces.IAST2variable;
17 import fr.upmc.ilp.ilp2.interfaces.ICgenEnvironment;
18 import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
19 import fr.upmc.ilp.ilp2.interfaces.ICommon;
20 import fr.upmc.ilp.ilp2.interfaces.IDestination;
21 import fr.upmc.ilp.ilp2.interfaces.ILexicalEnvironment;
22 import fr.upmc.ilp.ilp2.interfaces.IParser;
23 import fr.upmc.ilp.ilp2tme4b.interfaces.IAST2aMoinsQue;
24 import fr.upmc.ilp.ilp2tme4b.interfaces.IAST2transformeAMQ;
25
26 public class CEASTaMoinsQue extends CEASTinstruction
27                             implements IAST2aMoinsQue, IAST2transformeAMQ {
28
29     protected CEASTexpression condition;
30     protected CEASTinstruction corps;
31
32     public CEASTaMoinsQue(
33             IAST2expression<CEASTparseException> condition,
34             IAST2instruction<CEASTparseException> corps)
35     {
36         this.condition = (CEASTexpression) condition ;
```

```
37          this.corps = (CEASTinstruction) corps ;
38      }
39
40      public static IAST2aMoinsQue parse(
41                  final Element e, final IParser<CEASTparseException> parser)
42      throws CEASTparseException
43      {
44          final NodeList nl = e.getChildNodes();
45          IAST2expression<CEASTparseException> condition =
46              (IAST2expression<CEASTparseException>)
47                  parser.findThenParseChildAsUnique(nl, "condition");
48          IAST2instruction<CEASTparseException> corps =
49              (IAST2instruction<CEASTparseException>)
50                  parser.findThenParseChildAsSequence(nl, "corps");
51          return new CEASTaMoinsQue(condition, corps) ;
52      }
53
54      public Object eval(ILexicalEnvironment lexenv, ICommon common)
55              throws EvaluationException {
56          // Ne doit jamais être utilise
57          throw new EvaluationException("ne doit jamais etre appele");
58      }
59
60      public IAST2<CEASTparseException> transformeAMQ() {
61          CEASTunaryOperation newCond =
62              new CEASTunaryOperation("!",
63                      (IAST2expression<CEASTparseException>) this.getCondition()) ;
64          IAST2<CEASTparseException> newCorps =
65              ((IAST2transformeAMQ)this.getCorps()).transformeAMQ() ;
66          return new CEASTalternative(newCond,
67                          (IAST2instruction<CEASTparseException>) newCorps);
68      }
69
70      public IAST2<CEASTparseException> getCondition() {
71          return this.condition ;
72      }
73
74      public IAST2<CEASTparseException> getCorps() {
75          return this.corps ;
76      }
77
78      public void compileInstruction(StringBuffer buffer,
79              ICgenLexicalEnvironment lexenv, ICgenEnvironment common,
80              IDestination destination) throws CgenerationException {
81          throw new CgenerationException("ne doit jamais etre appelee") ;
82      }
83
84      @Override
85      public void
86      findGlobalVariables (final Set<IAST2variable> globalvars,
87                           final ICgenLexicalEnvironment lexenv ) {
88          getCondition().findGlobalVariables(globalvars, lexenv);
89          getCorps().findGlobalVariables(globalvars, lexenv);
90      }
91
92  }
```

Comme on peut le voir, cette classe implante les deux interfaces précédentes, dont celle pour la transformation de l'AST remplaçant l'instance de `CEASTaMoinsQue` en ins-

tance de CEASTalternative. On doit étendre ensuite toutes les autres classes de CEAST servant à instancier les objets faisant partie de l'AST pour implanter le parcours de transformation de l'AST de teller manière à propager la transformation au sous-nœuds et qui retourne le nœud lui-même ensuite. Par exemple, pour les contantes comme CEASTentier, il suffit de retourner le nœud lui-même :

```
1  package fr.upmc.ilp.ilp2tme4b.ast;
2
3  import fr.upmc.ilp.ilp2.ast.CEASTparseException;
4  import fr.upmc.ilp.ilp2.interfaces.IAST2;
5  import fr.upmc.ilp.ilp2tme4b.interfaces.IAST2transformeAMQ;
6
7  public class CEASTinteger extends fr.upmc.ilp.ilp2.ast.CEASTinteger implements
8          IAST2transformeAMQ {
9
10     public CEASTinteger(String valeur) {
11         super(valeur);
12     }
13
14     public IAST2<CEASTparseException> transformeAMQ() {
15         return this;
16     }
17
18 }
```

et pour un nœud complexe comme une alternative :

```
1  package fr.upmc.ilp.ilp2tme4b.ast;
2
3  import fr.upmc.ilp.ilp2.ast.CEASTparseException;
4  import fr.upmc.ilp.ilp2.interfaces.IAST2;
5  import fr.upmc.ilp.ilp2.interfaces.IAST2expression;
6  import fr.upmc.ilp.ilp2.interfaces.IAST2instruction;
7  import fr.upmc.ilp.ilp2tme4b.interfaces.IAST2transformeAMQ;
8
9  public class CEASTalternative extends fr.upmc.ilp.ilp2.ast.CEASTalternative
10 implements IAST2transformeAMQ {
11
12     public CEASTalternative(
13             IAST2expression<CEASTparseException> condition,
14             IAST2instruction<CEASTparseException> consequence,
15             IAST2instruction<CEASTparseException> alternant) {
16         super(condition, consequence, alternant);
17     }
18
19     public CEASTalternative(
20             IAST2expression<CEASTparseException> condition,
21             IAST2instruction<CEASTparseException> consequence) {
22         super(condition, consequence);
23     }
24
25     public IAST2<CEASTparseException> transformeAMQ() {
26         IAST2expression<CEASTparseException> condition;
27         IAST2instruction<CEASTparseException> consequence;
28         IAST2instruction<CEASTparseException> alternant;
29         condition =
30             (IAST2expression<CEASTparseException>)
31                 ((IAST2transformeAMQ)this.getCondition()).transformeAMQ();
32         consequence =
33             (IAST2instruction<CEASTparseException>)
```

```
            ((IAST2transformeAMQ)this.getConsequent()).transformeAMQ();
        if ( isTernary() ) {
            alternant =
                (IAST2instruction<CEASTparseException>)
                    ((IAST2transformeAMQ)this.getAlternant()).transformeAMQ();
            return new CEASTalternative(condition, consequence, alternant);
        } else {
            return new CEASTalternative(condition, consequence);
        }
    }

}
```

À l'étape 6, on introduit la nouvelle interface de fabrique :

```
package fr.upmc.ilp.ilp2tme4b.interfaces;

import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.IAST2expression;
import fr.upmc.ilp.ilp2.interfaces.IAST2instruction;

public interface IAST2Factory<Exc extends Exception>
extends fr.upmc.ilp.ilp2.interfaces.IAST2Factory<Exc> {

    public IAST2aMoinsQue newAMoinsQue(
            IAST2expression<CEASTparseException> condition,
            IAST2instruction<CEASTparseException> corps) ;

}
```

et la fabrique reprend elle tous les nœuds redéfinis dans le package :

```
package fr.upmc.ilp.ilp2tme4b.ast;

import java.util.List;

import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.IAST2alternative;
import fr.upmc.ilp.ilp2.interfaces.IAST2assignment;
import fr.upmc.ilp.ilp2.interfaces.IAST2binaryOperation;
import fr.upmc.ilp.ilp2.interfaces.IAST2boolean;
import fr.upmc.ilp.ilp2.interfaces.IAST2expression;
import fr.upmc.ilp.ilp2.interfaces.IAST2float;
import fr.upmc.ilp.ilp2.interfaces.IAST2functionDefinition;
import fr.upmc.ilp.ilp2.interfaces.IAST2instruction;
import fr.upmc.ilp.ilp2.interfaces.IAST2integer;
import fr.upmc.ilp.ilp2.interfaces.IAST2invocation;
import fr.upmc.ilp.ilp2.interfaces.IAST2localBlock;
import fr.upmc.ilp.ilp2.interfaces.IAST2primitiveInvocation;
import fr.upmc.ilp.ilp2.interfaces.IAST2program;
import fr.upmc.ilp.ilp2.interfaces.IAST2reference;
import fr.upmc.ilp.ilp2.interfaces.IAST2sequence;
import fr.upmc.ilp.ilp2.interfaces.IAST2string;
import fr.upmc.ilp.ilp2.interfaces.IAST2unaryBlock;
import fr.upmc.ilp.ilp2.interfaces.IAST2unaryOperation;
import fr.upmc.ilp.ilp2.interfaces.IAST2variable;
import fr.upmc.ilp.ilp2.interfaces.IAST2while;
import fr.upmc.ilp.ilp2tme4b.interfaces.IAST2Factory;
import fr.upmc.ilp.ilp2tme4b.interfaces.IAST2aMoinsQue;

```

```java
public class CEASTFactory extends fr.upmc.ilp.ilp2.ast.CEASTFactory
implements IAST2Factory<CEASTparseException>{

    @Override
    public IAST2program<CEASTparseException> newProgram(
            IAST2functionDefinition<CEASTparseException>[] functions,
            IAST2instruction<CEASTparseException> instruction) {
        return new CEASTprogram(functions, instruction);
    }

    @Override
    public IAST2alternative<CEASTparseException> newAlternative(
            IAST2expression<CEASTparseException> condition,
            IAST2instruction<CEASTparseException> consequent,
            IAST2instruction<CEASTparseException> alternant) {
        return new CEASTalternative(condition, consequent, alternant);
    }

    @Override
    public IAST2alternative<CEASTparseException> newAlternative(
            IAST2expression<CEASTparseException> condition,
            IAST2instruction<CEASTparseException> consequent) {
        return new CEASTalternative(condition, consequent);
    }

    @Override
    public IAST2assignment<CEASTparseException> newAssignment(
            IAST2variable variable, IAST2expression<CEASTparseException> value) {
        return new CEASTassignment(variable, value);
    }

    @Override
    public IAST2binaryOperation<CEASTparseException> newBinaryOperation(
            String operatorName,
            IAST2expression<CEASTparseException> leftOperand,
            IAST2expression<CEASTparseException> rightOperand) {
        return new CEASTbinaryOperation(operatorName, leftOperand, rightOperand);
    }

    @Override
    public IAST2boolean<CEASTparseException> newBooleanConstant(String value) {
        return new CEASTboolean(value);
    }

    @Override
    public IAST2float<CEASTparseException> newFloatConstant(String value) {
        return new CEASTfloat(value);
    }

    @Override
    public IAST2functionDefinition<CEASTparseException> newFunctionDefinition(
            String functionName, IAST2variable[] variables,
            IAST2instruction<CEASTparseException> body) {
        return new CEASTfunctionDefinition(functionName, variables, body);
    }

    @Override
    public IAST2integer<CEASTparseException> newIntegerConstant(String value) {
```

```
87          return new CEASTinteger(value);
88      }
89
90      @Override
91      public IAST2invocation<CEASTparseException> newInvocation(
92              IAST2expression<CEASTparseException> function,
93              IAST2expression<CEASTparseException>[] arguments) {
94          return new CEASTinvocation(function, arguments);
95      }
96
97      @Override
98      public IAST2localBlock<CEASTparseException> newLocalBlock(
99              IAST2variable[] variables,
100             IAST2expression<CEASTparseException>[] initializations,
101             IAST2instruction<CEASTparseException> body) {
102         return new CEASTlocalBlock(variables, initializations, body);
103     }
104
105     @Override
106     public IAST2primitiveInvocation<CEASTparseException> newPrimitiveInvocation(
107             String primitiveName,
108             IAST2expression<CEASTparseException>[] arguments) {
109         return new CEASTprimitiveInvocation(primitiveName, arguments);
110     }
111
112     @Override
113     public IAST2reference<CEASTparseException> newReference(
114             IAST2variable variable) {
115         return new CEASTreference(variable);
116     }
117
118     @Override
119     public IAST2sequence<CEASTparseException> newSequence(
120             List<IAST2instruction<CEASTparseException>> asts) {
121         return new CEASTsequence(asts);
122     }
123
124     @Override
125     public IAST2string<CEASTparseException> newStringConstant(String value) {
126         return new CEASTstring(value);
127     }
128
129     @Override
130     public IAST2unaryBlock<CEASTparseException> newUnaryBlock(
131             IAST2variable variable,
132             IAST2expression<CEASTparseException> initialization,
133             IAST2instruction<CEASTparseException> body) {
134         return new CEASTunaryBlock(variable, initialization, body);
135     }
136
137     @Override
138     public IAST2unaryOperation<CEASTparseException> newUnaryOperation(
139             String operatorName, IAST2expression<CEASTparseException> operand) {
140         return new CEASTunaryOperation(operatorName, operand);
141     }
142
143     @Override
144     public IAST2while<CEASTparseException> newWhile(
```

```
            IAST2expression<CEASTparseException> condition,
            IAST2instruction<CEASTparseException> body) {
        return new CEASTwhile(condition, body);
    }

    public IAST2aMoinsQue newAMoinsQue(
            IAST2expression<CEASTparseException> condition,
            IAST2instruction<CEASTparseException> corps) {
        throw new RuntimeException("should not occurr!");
    }

}
```

La nouvelle classe pour l'analyse syntaxique étend celle d'ILP et traite la création du nœud `CEASTaMoinsQue` :

```
package fr.upmc.ilp.ilp2tme4b.ast;

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

import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.IAST2;
import fr.upmc.ilp.ilp2.interfaces.IAST2program;

public class CEASTParser extends fr.upmc.ilp.ilp2.ast.CEASTParser {

    public CEASTParser(
            fr.upmc.ilp.ilp2.interfaces.IAST2Factory<CEASTparseException> factory) {
        super(factory);
    }

    @Override
    public IAST2program<CEASTparseException> parse (final Document d)
    throws CEASTparseException {
        final Element e = d.getDocumentElement();
        return CEASTprogram.parse(e, this);
    }

    @Override
    public IAST2<CEASTparseException> parse(Node n) throws CEASTparseException {
        try {
            switch ( n.getNodeType() ) {
            case Node.ELEMENT_NODE: {
                final Element e = (Element) n;
                final String name = e.getTagName();

                if ( "aMoinsQue".equals(name) ) {
                    return CEASTaMoinsQue.parse(e, this);
                } else {
                    return super.parse(n);
                }
            }
            default: {
                return super.parse(n);
            }
            }
        } catch (final CEASTparseException e) {
```

```
            throw e;
        } catch (final Exception e) {
            throw new CEASTparseException(e);
        }
    }
}
```

Il faut prêter une attention particulière à la classes `CEASTprogram` dont les constructeurs créent une instane de `CEASTsequence` sans passer par la fabrique, d"où le code ajouté aux constructeurs de la nouvelle classe pour réinstancier la sequence formant le coprs du programme :

```
package fr.upmc.ilp.ilp2tme4b.ast;

import org.w3c.dom.Element;

import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.IAST2;
import fr.upmc.ilp.ilp2.interfaces.IAST2functionDefinition;
import fr.upmc.ilp.ilp2.interfaces.IAST2instruction;
import fr.upmc.ilp.ilp2.interfaces.IAST2program;
import fr.upmc.ilp.ilp2.interfaces.IParser;
import fr.upmc.ilp.ilp2tme4b.interfaces.IAST2transformeAMQ;

public class CEASTprogram extends fr.upmc.ilp.ilp2.ast.CEASTprogram
implements IAST2transformeAMQ {

    public CEASTprogram(
            IAST2functionDefinition<CEASTparseException>[] definitions,
            IAST2instruction<CEASTparseException> body) {
        super(definitions, body);
    }

    public static IAST2program<CEASTparseException> parse (
            final Element e, final IParser<CEASTparseException> parser)
    throws CEASTparseException {
        return fr.upmc.ilp.ilp2.ast.CEASTprogram.parse(e, parser);
    }

    public IAST2<CEASTparseException> transformeAMQ() {
        IAST2functionDefinition<CEASTparseException>[] functionDefinitions =
            this.getFunctionDefinitions();
        for (int i = 0 ; i > functionDefinitions.length ; i++) {
            functionDefinitions[i] =
                (IAST2functionDefinition<CEASTparseException>)
                    ((IAST2transformeAMQ)functionDefinitions[i]).transformeAMQ();
        }
        IAST2instruction<CEASTparseException> body2 =
            (IAST2instruction<CEASTparseException>)
                ((IAST2transformeAMQ)this.getBody()).transformeAMQ();
        return new CEASTprogram(functionDefinitions, body2);
    }

}
```

La classe `Process` reprend la préparation pour faire exécuter la transformation de l'AST avant de passer la main à la suite du traitement (interprétation et compilation) :

```
package fr.upmc.ilp.ilp2tme4b;

```

```java
import java.io.IOException;

import org.w3c.dom.Document;

import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.IAST2program;
import fr.upmc.ilp.ilp2tme4b.ast.CEASTFactory;
import fr.upmc.ilp.ilp2tme4b.ast.CEASTParser;
import fr.upmc.ilp.ilp2tme4b.interfaces.IAST2Factory;
import fr.upmc.ilp.ilp2tme4b.interfaces.IAST2transformeAMQ;
import fr.upmc.ilp.tool.IFinder;

public class Process extends fr.upmc.ilp.ilp2.Process {

    public Process (IFinder finder) throws IOException {
        super(finder); // pour mémoire!
        getFinder().addPossiblePath("Java/src/fr/upmc/ilp/ilp2tme4");
        setGrammar(getFinder().findFile("grammar2-tme4.rng"));
        IAST2Factory<CEASTparseException> factory = new CEASTFactory();
        setFactory(factory);
        setParser(new CEASTParser(factory));
    }

    @Override
    public void prepare() {
        try {
            final Document d = getDocument(this.rngFile);

            final CEASTFactory factory = new CEASTFactory();
            final CEASTParser parser = new CEASTParser(factory);
            this.setCEAST((IAST2program<CEASTparseException>) parser.parse(d));
            this.setCEAST((IAST2program<CEASTparseException>)
                          ((IAST2transformeAMQ)this.ceast).transformeAMQ()) ;

            this.prepared = true;

        } catch (Throwable e) {
            this.preparationFailure = e;
        }
    }

}
```

et la classe `ProcessTest` utilise cette classe `Process` et applique le test aux jeux d'essai du TD-TME5 :

Pour la troisième solution, on conserve la même interface `IASTaMoinsQue` que pour la seconde solution, et la même interface de fabrique. L'implantation d'`CEASTaMoinsQue` devient :

```java
package fr.upmc.ilp.ilp2tme4b.ast;

import java.util.Set;

import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

import fr.upmc.ilp.ilp1.cgen.CgenerationException;
```

```java
import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp2.ast.CEASTexpression;
import fr.upmc.ilp.ilp2.ast.CEASTinstruction;
import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.IAST2;
import fr.upmc.ilp.ilp2.interfaces.IAST2expression;
import fr.upmc.ilp.ilp2.interfaces.IAST2instruction;
import fr.upmc.ilp.ilp2.interfaces.IAST2variable;
import fr.upmc.ilp.ilp2.interfaces.ICgenEnvironment;
import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
import fr.upmc.ilp.ilp2.interfaces.ICommon;
import fr.upmc.ilp.ilp2.interfaces.IDestination;
import fr.upmc.ilp.ilp2.interfaces.ILexicalEnvironment;
import fr.upmc.ilp.ilp2.interfaces.IParser;
import fr.upmc.ilp.ilp2tme4b.interfaces.IAST2aMoinsQue;
import fr.upmc.ilp.ilp2tme4b.interfaces.IAST2transformeAMQ;

public class CEASTaMoinsQue extends CEASTinstruction
                            implements IAST2aMoinsQue, IAST2transformeAMQ {

    protected CEASTexpression condition;
    protected CEASTinstruction corps;

    public CEASTaMoinsQue(
            IAST2expression<CEASTparseException> condition,
            IAST2instruction<CEASTparseException> corps)
    {
        this.condition = (CEASTexpression) condition ;
        this.corps = (CEASTinstruction) corps ;
    }

    public static IAST2aMoinsQue parse(
              final Element e, final IParser<CEASTparseException> parser)
    throws CEASTparseException
    {
        final NodeList nl = e.getChildNodes();
        IAST2expression<CEASTparseException> condition =
            (IAST2expression<CEASTparseException>)
                parser.findThenParseChildAsUnique(nl, "condition");
        IAST2instruction<CEASTparseException> corps =
            (IAST2instruction<CEASTparseException>)
                parser.findThenParseChildAsSequence(nl, "corps");
        return new CEASTaMoinsQue(condition, corps) ;
    }

    public Object eval(ILexicalEnvironment lexenv, ICommon common)
            throws EvaluationException {
        // Ne doit jamais être utilise
        throw new EvaluationException("ne doit jamais etre appele");
    }

    public IAST2<CEASTparseException> transformeAMQ() {
        CEASTunaryOperation newCond =
            new CEASTunaryOperation("!",
                    (IAST2expression<CEASTparseException>) this.getCondition()) ;
        IAST2<CEASTparseException> newCorps =
            ((IAST2transformeAMQ)this.getCorps()).transformeAMQ() ;
        return new CEASTalternative(newCond,
```

```
                        (IAST2instruction<CEASTparseException>) newCorps);
    }

    public IAST2<CEASTparseException> getCondition() {
        return this.condition ;
    }

    public IAST2<CEASTparseException> getCorps() {
        return this.corps ;
    }

    public void compileInstruction(StringBuffer buffer,
            ICgenLexicalEnvironment lexenv, ICgenEnvironment common,
            IDestination destination) throws CgenerationException {
        throw new CgenerationException("ne doit jamais etre appelee") ;
    }

    @Override
    public void
    findGlobalVariables (final Set<IAST2variable> globalvars,
                         final ICgenLexicalEnvironment lexenv ) {
        getCondition().findGlobalVariables(globalvars, lexenv);
        getCorps().findGlobalVariables(globalvars, lexenv);
    }

}
```

La fabrique devient :

```
package fr.upmc.ilp.ilp2tme4b.ast;

import java.util.List;

import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.IAST2alternative;
import fr.upmc.ilp.ilp2.interfaces.IAST2assignment;
import fr.upmc.ilp.ilp2.interfaces.IAST2binaryOperation;
import fr.upmc.ilp.ilp2.interfaces.IAST2boolean;
import fr.upmc.ilp.ilp2.interfaces.IAST2expression;
import fr.upmc.ilp.ilp2.interfaces.IAST2float;
import fr.upmc.ilp.ilp2.interfaces.IAST2functionDefinition;
import fr.upmc.ilp.ilp2.interfaces.IAST2instruction;
import fr.upmc.ilp.ilp2.interfaces.IAST2integer;
import fr.upmc.ilp.ilp2.interfaces.IAST2invocation;
import fr.upmc.ilp.ilp2.interfaces.IAST2localBlock;
import fr.upmc.ilp.ilp2.interfaces.IAST2primitiveInvocation;
import fr.upmc.ilp.ilp2.interfaces.IAST2program;
import fr.upmc.ilp.ilp2.interfaces.IAST2reference;
import fr.upmc.ilp.ilp2.interfaces.IAST2sequence;
import fr.upmc.ilp.ilp2.interfaces.IAST2string;
import fr.upmc.ilp.ilp2.interfaces.IAST2unaryBlock;
import fr.upmc.ilp.ilp2.interfaces.IAST2unaryOperation;
import fr.upmc.ilp.ilp2.interfaces.IAST2variable;
import fr.upmc.ilp.ilp2.interfaces.IAST2while;
import fr.upmc.ilp.ilp2tme4b.interfaces.IAST2Factory;
import fr.upmc.ilp.ilp2tme4b.interfaces.IAST2aMoinsQue;

public class CEASTFactory extends fr.upmc.ilp.ilp2.ast.CEASTFactory
implements IAST2Factory<CEASTparseException>{

```

```
    @Override
    public IAST2program<CEASTparseException> newProgram(
            IAST2functionDefinition<CEASTparseException>[] functions,
            IAST2instruction<CEASTparseException> instruction) {
        return new CEASTprogram(functions, instruction);
    }

    @Override
    public IAST2alternative<CEASTparseException> newAlternative(
            IAST2expression<CEASTparseException> condition,
            IAST2instruction<CEASTparseException> consequent,
            IAST2instruction<CEASTparseException> alternant) {
        return new CEASTalternative(condition, consequent, alternant);
    }

    @Override
    public IAST2alternative<CEASTparseException> newAlternative(
            IAST2expression<CEASTparseException> condition,
            IAST2instruction<CEASTparseException> consequent) {
        return new CEASTalternative(condition, consequent);
    }

    @Override
    public IAST2assignment<CEASTparseException> newAssignment(
            IAST2variable variable, IAST2expression<CEASTparseException> value) {
        return new CEASTassignment(variable, value);
    }

    @Override
    public IAST2binaryOperation<CEASTparseException> newBinaryOperation(
            String operatorName,
            IAST2expression<CEASTparseException> leftOperand,
            IAST2expression<CEASTparseException> rightOperand) {
        return new CEASTbinaryOperation(operatorName, leftOperand, rightOperand);
    }

    @Override
    public IAST2boolean<CEASTparseException> newBooleanConstant(String value) {
        return new CEASTboolean(value);
    }

    @Override
    public IAST2float<CEASTparseException> newFloatConstant(String value) {
        return new CEASTfloat(value);
    }

    @Override
    public IAST2functionDefinition<CEASTparseException> newFunctionDefinition(
            String functionName, IAST2variable[] variables,
            IAST2instruction<CEASTparseException> body) {
        return new CEASTfunctionDefinition(functionName, variables, body);
    }

    @Override
    public IAST2integer<CEASTparseException> newIntegerConstant(String value) {
        return new CEASTinteger(value);
    }

```

```
90      @Override
91      public IAST2invocation<CEASTparseException> newInvocation(
92              IAST2expression<CEASTparseException> function,
93              IAST2expression<CEASTparseException>[] arguments) {
94          return new CEASTinvocation(function, arguments);
95      }
96
97      @Override
98      public IAST2localBlock<CEASTparseException> newLocalBlock(
99              IAST2variable[] variables,
100             IAST2expression<CEASTparseException>[] initializations,
101             IAST2instruction<CEASTparseException> body) {
102         return new CEASTlocalBlock(variables, initializations, body);
103     }
104
105     @Override
106     public IAST2primitiveInvocation<CEASTparseException> newPrimitiveInvocation(
107             String primitiveName,
108             IAST2expression<CEASTparseException>[] arguments) {
109         return new CEASTprimitiveInvocation(primitiveName, arguments);
110     }
111
112     @Override
113     public IAST2reference<CEASTparseException> newReference(
114             IAST2variable variable) {
115         return new CEASTreference(variable);
116     }
117
118     @Override
119     public IAST2sequence<CEASTparseException> newSequence(
120             List<IAST2instruction<CEASTparseException>> asts) {
121         return new CEASTsequence(asts);
122     }
123
124     @Override
125     public IAST2string<CEASTparseException> newStringConstant(String value) {
126         return new CEASTstring(value);
127     }
128
129     @Override
130     public IAST2unaryBlock<CEASTparseException> newUnaryBlock(
131             IAST2variable variable,
132             IAST2expression<CEASTparseException> initialization,
133             IAST2instruction<CEASTparseException> body) {
134         return new CEASTunaryBlock(variable, initialization, body);
135     }
136
137     @Override
138     public IAST2unaryOperation<CEASTparseException> newUnaryOperation(
139             String operatorName, IAST2expression<CEASTparseException> operand) {
140         return new CEASTunaryOperation(operatorName, operand);
141     }
142
143     @Override
144     public IAST2while<CEASTparseException> newWhile(
145             IAST2expression<CEASTparseException> condition,
146             IAST2instruction<CEASTparseException> body) {
147         return new CEASTwhile(condition, body);
```

```
148     }
149
150     public IAST2aMoinsQue newAMoinsQue(
151             IAST2expression<CEASTparseException> condition,
152             IAST2instruction<CEASTparseException> corps) {
153         throw new RuntimeException("should not occurr!");
154     }
155
156 }
```

et l'analyseur syntaxique :

```
1  package fr.upmc.ilp.ilp2tme4b.ast;
2
3  import org.w3c.dom.Document;
4  import org.w3c.dom.Element;
5  import org.w3c.dom.Node;
6
7  import fr.upmc.ilp.ilp2.ast.CEASTparseException;
8  import fr.upmc.ilp.ilp2.interfaces.IAST2;
9  import fr.upmc.ilp.ilp2.interfaces.IAST2program;
10
11 public class CEASTParser extends fr.upmc.ilp.ilp2.ast.CEASTParser {
12
13     public CEASTParser(
14             fr.upmc.ilp.ilp2.interfaces.IAST2Factory<CEASTparseException> factory) {
15         super(factory);
16     }
17
18     @Override
19     public IAST2program<CEASTparseException> parse (final Document d)
20     throws CEASTparseException {
21         final Element e = d.getDocumentElement();
22         return CEASTprogram.parse(e, this);
23     }
24
25     @Override
26     public IAST2<CEASTparseException> parse(Node n) throws CEASTparseException {
27         try {
28             switch ( n.getNodeType() ) {
29             case Node.ELEMENT_NODE: {
30                 final Element e = (Element) n;
31                 final String name = e.getTagName();
32
33                 if ( "aMoinsQue".equals(name) ) {
34                     return CEASTaMoinsQue.parse(e, this);
35                 } else {
36                     return super.parse(n);
37                 }
38             }
39             default: {
40                 return super.parse(n);
41             }
42             }
43         } catch (final CEASTparseException e) {
44             throw e;
45         } catch (final Exception e) {
46             throw new CEASTparseException(e);
47         }
48     }
```

```
}
```

À l'étape 9, on introduit le générateur de code C dans la classe `CEASTaMoinsQue`, qui apparaît dans le code donné précédemment.

Les classes `Process` et `ProcessTest` sont similaires à celles de la solution 2, sauf qu'on ne fait pas la transformation de l'AST :

```
package fr.upmc.ilp.ilp2tme4b;

import java.io.IOException;

import org.w3c.dom.Document;

import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.IAST2program;
import fr.upmc.ilp.ilp2tme4b.ast.CEASTFactory;
import fr.upmc.ilp.ilp2tme4b.ast.CEASTParser;
import fr.upmc.ilp.ilp2tme4b.interfaces.IAST2Factory;
import fr.upmc.ilp.ilp2tme4b.interfaces.IAST2transformeAMQ;
import fr.upmc.ilp.tool.IFinder;

public class Process extends fr.upmc.ilp.ilp2.Process {

    public Process (IFinder finder) throws IOException {
        super(finder); // pour mémoire!
        getFinder().addPossiblePath("Java/src/fr/upmc/ilp/ilp2tme4");
        setGrammar(getFinder().findFile("grammar2-tme4.rng"));
        IAST2Factory<CEASTparseException> factory = new CEASTFactory();
        setFactory(factory);
        setParser(new CEASTParser(factory));
    }

    @Override
    public void prepare() {
        try {
            final Document d = getDocument(this.rngFile);

            final CEASTFactory factory = new CEASTFactory();
            final CEASTParser parser = new CEASTParser(factory);
            this.setCEAST((IAST2program<CEASTparseException>) parser.parse(d));
            this.setCEAST((IAST2program<CEASTparseException>)
                          ((IAST2transformeAMQ)this.ceast).transformeAMQ()) ;

            this.prepared = true;

        } catch (Throwable e) {
            this.preparationFailure = e;
        }
    }

}
```