UFR 919 Informatique – Master Informatique

Spécialité STL – UE MI016 – ILP

TME1 — Installation et programmation en ILP1

Christian Queinnec

La séance est divisée en deux parties. Dans la première, vous installerez les outils nécessaires afin de pouvoir les utiliser dans la seconde partie.

1 Structure des documents de TME

Les documents de TME débutent par un « objectif » qui indique, en termes généraux le thème du TME. Les « buts » décrivent des compétences qui devraient être acquises après le TME. « Savoir » décrit une compétence qui est mécaniquement vérifiable tandis que « Comprendre » désigne un processus intellectuel qui n'est pas directement enseignable et souvent peu vérifiable. Les travaux relevant de cette dernière catégorie sont marqués avec le symbole Ω en marge et nécessitent de se concentrer !

2 Environnement de travail

Objectif : Se donner un environnement de travail incorporant les outils et les programmes du cours.

Buts

— Identifier et localiser les outils
— Installer son environnement de travail (à l'UPMC et chez soi)
— Écrire puis valider un programme XML
— Valider puis évaluer un programme ILP
— Tester un ensemble de programmes ILP
— Écrire puis exécuter un test avec JUnit

Les liens

Les répertoires dans lesquels vous pourrez trouver informations et outils pour le cours ILP se trouvent en :

```
file:///Infos/lmd/2013/master/ue/ilp-2013oct/
http://www-master.ufr-info-p6.jussieu.fr/site-annuel-courant/
```

Les sources du système ILP se trouvent sous :

```
file:///Infos/lmd/2013/master/ue/ilp-2013oct/
```

Par ailleurs, indiquez, si besoin, dans votre navigateur, que votre proxy est `proxy.ufr-info-p6.jussieu.fr` port 3128 (c'est utile pour charger des greffons supplementaires dans Eclipse). Par la même occasion, placez un signet sur le site du master car, lors des examens, vous n'aurez pas accès à Internet pour retrouver ce site.

Les examens sont individuels, ils ne s'effectuent pas en binôme ! Ainsi, chacun doit avoir installé ces outils dans son répertoire personnel afin de pouvoir répondre aux examens qui se passent sur machine.

Le travail à réaliser suppose que vous avez quelques connaissances de bash et de Java.

2.1 Travail à réaliser

Lancez Eclipse avec la commande `/usr/local/eclipse/eclipse`. Cette commande lance la toute dernière version (3.9 Kepler) d'Eclipse (qui est stockée en `/usr/local/eclipse/`).

Java7 est installé sur les machines de TME et ILP utilise Java7. La commande `java` par défaut démarre Java7.

Pour chez vous, Eclipse peut être trouvé sur son site en `http://www.eclipse.org/`. Prenez une version *Eclipse IDE for Java EE Developers* adaptée à votre machine.

Une fois lancé, Eclipse veut savoir où ranger son espace de travail (pour *workspace*), dites oui à ce qu'il vous propose. C'est, par défaut, `~/workspace/`, ne changez rien ! Cochez aussi la case lui demandant de mémoriser cette réponse afin qu'il ne vous la pose plus. Ne changez pas d'espace de travail, c'est là où nous récupérerons vos textes lors des examens.

Ajustez Eclipse de la manière suivante :

1. Menu *Window*, *Preferences*, *Java*, *Compiler*, réglez *Compliance level* à 1.7 (si ce n'est déjà fait). Si Eclipse ne sait pas où se trouve java 1.7, il faut lui indiquer où se situe cette JVM ainsi : Menu *Window*, *Preferences*, *Java*, *Installed JRE*, bouton *Add* et indiquer le chemin menant à cette JVM : `/usr/lib/jvm/jdk1.7.0_07`.
2. Menu *Window*, *Preferences*, *General*, *Editors*, *Text Editors*, cochez la case *Show Print Margin* qui montre la marge droite située à 80 colonnes : ne dépassez pas cette colonne, nous risquons de ne pas corriger vos programmes (notamment lors des examens) si leurs lignes dépassent cette limite.
3. Menu *Window*, *Preferences*, *General*, *Editors*, *Text Editors*, cochez la case *insert spaces for tabs* (car il ne faut jamais utiliser de tabulations dans les programmes (sauf si c'est une convention imposée comme pour les Makefile par exemple)).
4. Menu *Window*, *Preferences*, *General*, *Workspace*, *Text File Encoding* : utf8.
5. Menu *Window*, *Preferences*, *General*, *Workspace*, *New text file line delimiter* : unix.

Parmi les problèmes qui parfois surviennent, il y a

— pas assez de place sur votre compte (utilisez la commande `quota` pour le déterminer)

— Eclipse est sorti sans avoir effacé le fichier `~/workspace/.metadata/.lock` : effacez-le puis relancer Eclipse

— Eclipse ne se relance plus et l'erreur figurant dans `~/workspace/.metadata/.log` mentionne des classes SWT en relation avec le navigateur (*browser*) : supprimez le répertoire `~/.mozilla/eclipse`

— Plein de classes d'ILP sont en rouge : Vous n'êtes probablement pas en Java 7 : il y a Java/Compiler/ComplianceLevel mais il y a aussi JavaCompiler/ComplianceLevel et quelques fois les deux ne sont pas d'accord.

2.1.1 Installation sources ILP

Pour installer les sources comme un projet dans Eclipse, voici une procédure :

1. dans un shell (vous avez bien lancé Eclipse en arrière plan ?), faites ce qui suit (cela va créer quelques répertoires dont Java, Grammars et C) :

```
% /Infos/lmd/2013/master/ue/ilp-2013oct/Scripts/install.sh
```

2. Dans l'écran initial d'Eclipse, cliquez sur la flèche *workbench* (tout en haut à droite)
3. Créez un nouveau projet de type Java (*File*, *New java project*) ; dans la case nom du projet, tapez ILP. Au moment où vous finissez de taper P, Eclipse détecte qu'un tel répertoire existe et vous propose de le récupérer tel quel : cliquez *Finish*. Si ce n'était pas le cas, cliquez sur *create new project from sources* et choisissez le répertoire `workspace/ILP`, enfin, cliquez *Finish*.
4. Le projet est presque prêt. Eclipse va peut-être vous demander s'il faut ouvrir la perspective Java, répondez affirmativement.

Voici ce que l'on peut voir dans l'explorateur de paquetages :
— `Java/src` : les sources d'ILP en Java.
— `*.jar` : les archives des outils utilisés dans le projet.
— `JRE System Library [JVM 1.7]` : l'exécutif Java utilisé par ce projet.
— `C` : des bibliothèques pour la compilation d'ILP vers C.
— `Grammars` : les grammaires des différentes versions ILP* et des programmes dans le sous-répertoire `Sample`.
— `Java` : le répertoire contenant `Java/src`, d'autres utilitaires et la documentation Javadoc du projet.
— et quelques autres fichiers ou répertoires comme le fichier `build.xml` qui est le fichier Ant de construction des binaires et autre produits du projet ou le fichier `LISEZ.MOI` dont le nom indique l'usage qu'il faut en faire (il contient notamment la documentation d'emploi du greffon).

Cette année, il y a encore une différence entre mon Eclipse et celui de la PPTI, un point d'exclamation jaune apparaît sur la classe `fr.upmc.ilp.ilp1.test.AbstractMainForProcessTest`, vous pouvez l'ignorer ou réparer comme suggéré.

Vous pouvez alors tester un peu l'installation : effectuez ceci

1. Pour tester un peu plus, dépliez `Java/src` puis le paquetage `fr.upmc.ilp.ilp1.test`, clic droit sur le fichier `WholeTestSuite`, *Run as JUnit test*. Normalement, tous les tests doivent passer (ce que montre une barre totalement verte).

Eclipse cherche normalement à compiler tous les sources du projet automatiquement, et donc lors de la création d'un projet à partir de sources. la fenêtre du bas sert à produire les messages de compilation. Si cette fenêtre fait apparaître des erreurs, il se peut que certaines configurations du projet ne soient pas correctes.

Pensez à étudier le tutoriel d'Eclipse intégré dans l'aide en ligne d'Eclipse.

2.1.2 Installation greffon ILP

Le greffon est **déjà installé** sur les machines de l'ARI. Pour détecter si le greffon est installé, ouvrez le menu contextuel du fichier `build.xml` et regardez si une entrée ILP y figure. Si le greffon n'est pas installé (ce sera le cas chez vous), voici comment faire :

1. Menu *Help*, item *Install New Software*,
2. cliquez sur le bouton *Add*. Eclipse ouvre une fenêtre, indiquez ILP comme nom puis tapez l'URL suivante dans le champ *Location* :

```
http://www-master.ufr-info-p6.jussieu.fr/2013/Ext/queinnec/ILP/
```

   Cliquez OK, Eclipse va alors chercher les greffons disponibles avec les numéros de version.

3. Cochez ILP pour demander son installation puis cliquez sur le bouton *Next*
4. Acceptez la licence d'utilisation et passez, avec le bouton *Next*, à l'écran suivant.
5. Quelque *Valider*, *OK* et *Finish* plus tard, le greffon s'installe. Il faut juste confirmer que je ne l'ai pas signé et que vous me faites confiance en cliquant sur *Install all*.
6. Eclipse se relance alors et le greffon est normalement installé.

Le greffon ILP installe de nouveaux menus contextuels sur les fichiers de suffixe `.xml`, `.rnc` et `.rng`. Pour vérifier qu'il est bien installé, faites comme indiqué au début de cette section.

À propos, si vous souhaitez installer d'autres greffons dans Eclipse, vous aurez besoin d'indiquer à Eclipse d'utiliser un relais (pour *proxy*) pour accéder à Internet. C'est dans les préférences d'Eclipse, menu *Window*, *Preferences...*, item *General*, *Network Connection*. Indiquez alors les coordonnées du proxy conseillé par l'ARI, fort probablement `proxy.ufr-info-p6.jussieu.fr` port 3128.

Parmi les greffons que j'utilise régulièrement : svn (pour utiliser Subversion), findbugs, pmd, checkstyle.

2.1.3 Deux ou trois trucs pour Eclipse

Si vous modifiez des fichiers en dehors d'Eclipse, la touche fonction F5 indique à Eclipse de se resynchroniser avec le système de fichiers.

Lorsque le curseur est sur un nom, F3 permet d'ouvrir le fichier définissant ce nom. F4 permet de voir la hiérarchie des classes ou interfaces du nom concerné.

CTRL-shift-O permet d'importer les classes ou interfaces qui manquent. Attention cependant en cas de noms dupliqués à importer la bonne version (`File` par exemple existe dans `java.io` et aussi dans `fr.upmc.ilp.tools`)!

CTRL-space propose des complétions. Le survol par la souris d'une zone erronée propose souvent de bonnes solutions (*Quick Fix*).

Augmentez votre espace vital en déplaçant les vues, souvent inutiles, *Outline*, *Tasks* sous la vue *Package explorer*. Pour cela faites glisser l'onglet depuis sa position actuelle à celle souhaitée. Vous pouvez aussi les fermer.

Vous pouvez voir plusieurs fichiers côte-à-côte en prenant l'onglet de l'un et en le faisant glisser sur le bord horizontal (ou vertical (essayez les deux!)) de la fenêtre. On peut même maintenant ouvrir 2 fenêtres sur le même fichier (menu contextuel 'new Editor' sur l'onglet contenant le nom du fichier).

Il peut être aussi bien utile d'associer leur source et documentation aux archives `.jar` qui vous sont proposés. Pour ce faire, Menu Project, Properties, Java Build Path, onglet Libraries, choisir puis déplier la description d'un .jar, cliquer sur Source attachment : et choisir le .zip contenant les sources.

2.1.4 Deux ou trois trucs pour Mercurial

Les diverses variantes d'ILP vous seront fournies via le gestionnaire de version Mercurial `http://mercurial.selenic.com/`. Le script `install.sh` précédemment utilisé a cloné le dépôt central (sous `/Infos/...`, lire le code d'`install.sh` pour connaître l'endroit précis) qui sera régulièrement enrichi des nouvelles versions. Voici comment mettre à jour votre dépôt pour la version ilpX :

```
1 cd ; cd workspace/ILP
2 hg pull -u -r ilpX
```

La commande `hg pull` prend connaissance de ce qui a évolué dans le dépôt central par rapport à votre dépôt personnel, l'option `-u` importe les modifications correspondant à la version (option `-r`) nommée ilp*X*. Attention, si vous avez modifié certains des programmes d'ILP (ce que vous ne devez pas faire), il vous faudra gérer les fusions.

La structure du dépôt central ressemblera à :

```
                 /--> ilp1tme2
                /
ilp1 +----> ilp2 +----> ilp3
     \
      \--> ilp1tme1
```

Sur ce schéma (en ASCII-art) apparaissent les labels des différentes versions et leurs relations historiques (l'utilitaire `hgview` permet de visualiser ces relations).

Vous êtes fortement encouragés à gérer votre dépôt lors des TME, créer une branche par TME (nommez-les `tmeY` afin de ne pas entrer en conflit avec les branches `ilpXtmeY` qui seront utilisées dans le dépôt central), archiver souvent afin de pouvoir revenir en arrière ou même, développer de manière collaborative.

Un livre entier (le hgbook) est disponible sur le site de Mercurial.

Comme les sources d'ILP proviennent d'un dépôt Mercurial, Eclipse le détecte et vous permet de gérer directement votre propre dépôt. Lisez la documentation de Mercurial pour gérer votre dépôt, le partager, faire des essais aventureux et pouvoir revenir en arrière, etc.

3 Cycle d'exécution d'un programme ILP

Objectif : Apprendre à réaliser toutes les étapes permettant d'exécuter un programme ILP et vérifier la bonne installation de l'environnement de travail.

Buts

— Comprendre les exemples de programmes ILP1.
— Savoir vérifier la syntaxe d'un programme en validant le document XML correspondant par rapport à sa grammaire.
— Savoir exécuter un programme.

Les liens

XML	`http://www.w3.org/XML/Core/`
RelaxNG	`http://www.oasis-open.org/committees/relax-ng/`

Documents sur RelaxNG

Tutoriel	`http://www.oasis-open.org/committees/relax-ng/tutorial-20011203.html`
Syntaxe compacte	`http://www.oasis-open.org/committees/relax-ng/compact-20021121.html`
Livre	`http://books.xmlschemata.org/relaxng/`

Outils spécifiques

Jing	`http://www.thaiopensource.com/`

3.1 Comment valider un document avec Jing ?

Jing est le validateur de document XML par rapport aux schémas RelaxNG que nous utilisons pour définir nos grammaires de langages. Voici les principales étapes permettant de le mettre en œuvre :

— On peut, dans Eclipse, grâce au greffon ILP, utiliser le menu contextuel sur un fichier XML, menu *ILP* puis *Validate*. Les résultats apparaissent dans l'onglet console (dans la sous-console intitulée *Jing Output*.

— Un greffon spécialisé en XML procure la possibilité de vérifier si un fichier est bien-formé. C'est l'entrée *Validate* du menu contextuel des fichiers XML. La validation par le greffon ILP vérifie en plus la conformité à une grammaire ILP.

— en termes de ligne de commande, pour valider un fichier XML par rapport à une grammaire RelaxNG, il faut faire (où JING_JAR est le fichier approprié (dans `Java/jars/`) :

```
1  java -jar ${JING_JAR} <schema>.rng <fichier>.xml
```

3.2 Travail à réaliser

— Valider à l'aide du greffon quelques fichiers XML (dans `Grammars/Samples/`). Modifiez un de ces fichiers (de manière réversible bien sûr) afin de voir comment s'affichent les problèmes (balises manquantes, attribut mal orthographié, etc.)

— Indiquer au greffon la grammaire ILP1 à utiliser par défaut et valider quelques fichiers XML (utiliser le menu contextuel que procure le greffon sur une grammaire RelaxNG).

— Valider un fichier XML à partir de la ligne de commande.

3.3 Comment exécuter un programme ILP1 ?

Pour exécuter un programme ILP1 avec l'interprète, nous vous fournissons une classe assez sommaire appelée `EASTFileTest.java` disponible dans le paquetage `fr.upmc.ilp.ilp1.eval.test`. Pour exécuter l'échantillon des programmes ILP1 du répertoire `Grammars/Samples`, il faut cliquer sur le nom de la classe (`EASTFileTest.java`) avec le bouton droit et sélectionner *Run as* puis *JUnit Test*. L'exécution de cette classe de test produit une trace dans la fenêtre du bas où vous pourrez voir le nom de chaque programme exécuté, le résultat et les impressions produites ainsi que ce qui était attendu.

Les programmes ILP1 du répertoire `Samples` ont tous un nom de la forme `u<d>+-1.xml` où `<d>+` représente une suite de chiffres. Une manière simple de faire exécuter un programme ILP1 consiste donc à l'écrire dans un fichier ayant un nom de cette forme du répertoire `Samples` et de relancer les tests de la classe `EASTFileTest.java`.

Nous passons effectivement par des tests unitaires écrits avec l'outil JUnit pour exécuter les programmes. Les liens vous permettront de vous documenter sur JUnit (versions 3 et 4 (les deux sont utilisées)) :

JUnit `http://www.junit.org/`

JUnit `http://junit.sourceforge.net/`

Une meilleure description du traitement d'un programme se trouve dans la classe `fr.upmc.ilp.ilp1.Process` (cette classe sait non seulement interpréter les programmes mais aussi les compiler). L'ensemble des tests d'ILP1 est déclenchable avec la classe `fr.upmc.ilp.ilp1.test.WholeTestSuite`. Il est d'ailleurs bon de lire les classes de tests car elles montrent comment synthétiser, lire, plus généralement traiter des programmes ILP.

La classe `EASTFileTest.java` contient également un point d'entrée permettant de la considérer comme une application Java à part entière. On peut lancer une application Java avec le menu contextuel *Run as*, puis *Java application*. Il faut toutefois, avant de

faire cela, paramétrer le lanceur d'application avec *Run as*, puis *Open Run Dialog...* afin d'indiquer que l'argument de l'application est le fichier que vous voulez tester individuellement.

Pour indiquer le fichier à évaluer, vous devez le nommer avec un chemin correct, ce qui implique de connaître le répertoire où se lance le programme (par défaut `~/workspace/`). Vous pouvez aussi écrire `${file_prompt}` ce qui vous permettra de choisir interactivement le fichier à traiter.

On peut aussi lancer ce test à partir de la ligne de commande avec (si l'on est en `~/workspace/ILP`) :

```
1  java -cp Java/bin:Java/jars/jing.jar:Java/jars/junit-4.10.jar \
2    fr.upmc.ilp.ilp1.eval.test.EASTFileTest <fichier>.xml
```

Noter que le *classpath* d'Eclipse n'est pas celui de cette ligne de commande.

3.4 Travail à réaliser

— Exécuter les tests d'ILP1 et comparer les résultats obtenus avec ceux que vous attendiez en lisant quelques uns des programmes exécutés.
— Recopier en le modifiant un peu un des fichiers XML de test d'ILP1 pour le tester avec la classe `EASTFileTest.java` en ligne de commande (prendre, par exemple, comme base `u10-1.xml`)
— Modifier le fichier XML afin de voir ce que produisent les anomalies (XML mal formé, invalide, résultat non attendu, etc.)
— Noter (ou mémoriser dans un script) ce que vous avez retenu.
— Tester ce même nouveau programme ILP1 au sein de tous les autres (dans le répertoire `Grammars/Samples`) avec les tests JUnit lancés depuis Eclipse. N'oubliez pas les fichiers `.result` et `.print` !

4 Programmer en ILP1

Objectif : Comprendre toutes les étapes permettant d'écrire et d'exécuter un programme ILP1.

Buts

— Écrire un programme ILP1.
— Le mettre au point syntaxiquement par validation.
— Le mettre au point sémantiquement par son exécution.

Vous pouvez choisir les outils qui vous plaisent mais il est bon de voir ce que savent faire les uns et les autres.

4.1 Édition avec Eclipse d'un document XML

Créer un fichier XML et ouvrez-le avec le menu contextuel avec un éditeur de texte ou un éditeur structuré. Comparer les deux visions, structurée ou textuelle.

4.2 Édition sous Emacs documents XML avec nxml-mode et de schémas avec rnc-mode

Les liens

nxml-mode	`http://www.thaiopensource.com/nxml-mode/`
rnc-mode	`http://www.pantor.com/`

4.2.1 Mise en place du mode nxml dans Emacs

Pour utiliser nXML sous emacs, placez d'abord les lignes suivantes dans votre fichier `~/.emacs` (**attention**, si vous copiez-collez ce qui suit depuis le document PDF vers un éditeur de textes, il se peut que vous ayiez à convertir les apostrophes en des simples guillemets (une petite barre verticale) et à retirer des blancs superflus) :

```
(setq load-path
      (append load-path
              '("/Infos/lmd/2013/master/ue/ilp-2013oct/ELISP/nxml-mode-20041004/")))

(load "/Infos/lmd/2013/master/ue/ilp-2013oct/ELISP/nxml-mode-20041004/rng-auto.el")

(setq auto-mode-alist
      (cons '("\\.\\(xml\\|xsl\\|rng\\|xhtml\\)$" . nxml-mode)
            auto-mode-alist))
```

Ceci fait, dès que vous lancerez Emacs sur un fichier de suffixe xml, xsl, rng ou xhtml, ce mode sera automatiquement activé. Vous pouvez aussi copier ces fichiers (par exemple, dans un répertoire `.emacs.d/`) et les y compiler.

4.2.2 Mise en place du mode rnc

Pour utiliser rnc-mode sous emacs, placez d'abord les lignes suivantes dans votre fichier `~/.emacs` :

```
(autoload 'rnc-mode
          "/Infos/lmd/2013/master/ue/ilp-2013oct/ELISP/rnc-mode")
(setq auto-mode-alist
      (cons '("\\.rnc$" . rnc-mode) auto-mode-alist))
```

Ceci fait, dès que vous lancerez Emacs sur un fichier de suffixe rnc, ce mode sera automatiquement activé. Vous pouvez aussi copier ces fichiers (par exemple, dans un répertoire `.emacs.d/`) et les y compiler.

4.2.3 Utilisation des modes

Pour les documents XML, éditez ensuite un nouveau fichier avec l'extension `xml` puis jeter un coup d'œil à la liste des liaisons des clés (*key bindings* ou en raccourci par défaut `^Hm`) du mode pour voir les principales commandes. Pour compléter, regarder sur le site donné ci-avant dans la partie "*Further information about nXML mode*".

Le mode nXML vous donne quelques outils pour faciliter l'écriture de fichiers XML comme la validation au fur et à mesure des documents et la complétion des noms de balise. Pour que la validation fonctionne, il faut associer un schéma au fichier en cours d'édition. Pour cela, on va dans le menu XML et on choisit "Set schema" puis "File..." et on sélectionne le fichier de grammaire en format rnc. Pour les programmes ILP1, il s'agit du fichier `grammar1.rnc` du répertoire `Grammars`. Quand la validation est activée, un message `nXML-valid` ou `nXML-invalid` apparaît dans la barre d'état d'emacs en bas de la fenêtre.

Les deux commandes les plus utiles sont Ctrl-enter pour compléter le nom d'une balise ou d'un attribut et Ctrl-c Ctrl-f pour générer la balise fermante de l'élément englobant ouvert à l'endroit où se situe le curseur (en fait, j'utilise plutôt </ qui suffit à fermer un élément).

Pour les schémas RelaxNG (en syntaxe compacte), il suffit d'éditer un fichier avec l'extension `rnc`. Le mode rnc est cependant très limité. Il fait une légère coloration syntaxique, introduit les crochets et les accolades électriques et permet de commenter et décommenter rapidement des parties de fichier. Faire un "List key bindings" pour en savoir (un peu) plus.

4.3 Travail à réaliser

Écrire un programme ILP1 qui calcule le discriminant d'une équation du second degré étant donnés les coefficients a, b, c. Votre programme doit d'abord lier les variables a, b et c aux valeurs choisies pour exécuter le calcul (avec des blocs unaires enchâssés). Ensuite, le programme calcule le discriminant et doit retourner l'une des chaînes suivantes :

— "discriminant negatif: aucune racine"
— "discriminant positif: deux racines"
— "discriminant nul: une seule racine"

Votre programme doit bien sûr être bien formé et valide. Essayez aussi de ne calculer le discriminant qu'au plus une fois. L'évaluation de ce programme fait l'objet d'une section ci-après.

***Pour enseignant*:** discriminant.xml

```
<programme1>
  <!-- solution de l'exercice correspondant du TP1 -->
  <!-- $Id: discriminant.xml,v 1.3 2004/09/29 11:56:41 malenfant Exp $ -->
  <blocUnaire>
    <variable nom='a'/><valeur><flottant valeur='1.0'/></valeur>
    <corps>
      <blocUnaire>
        <variable nom='b'/><valeur><flottant valeur='2.0'/></valeur>
        <corps>
          <blocUnaire>
            <variable nom='c'/><valeur><flottant valeur='1.0'/></valeur>
            <corps>
              <blocUnaire>
                <variable nom='discriminant'/>
                <valeur>
                  <operationBinaire operateur='-'>
                    <operandeGauche>
                      <operationBinaire operateur='*'>
                        <operandeGauche>
                          <variable nom='b'/>
                        </operandeGauche>
                        <operandeDroit>
                          <variable nom='b'/>
                        </operandeDroit>
                      </operationBinaire>
                    </operandeGauche>
                    <operandeDroit>
                      <operationBinaire operateur='*'>
                        <operandeGauche>
                          <flottant valeur='4.0'/>
                        </operandeGauche>
                        <operandeDroit>
                          <operationBinaire operateur='*'>
                            <operandeGauche>
                              <variable nom='a'/>
                            </operandeGauche>
                            <operandeDroit>
                              <variable nom='c'/>
                            </operandeDroit>
                          </operationBinaire>
                        </operandeDroit>
                      </operationBinaire>
```

```
                    </operandeDroit>
                  </operationBinaire>
                </valeur>
                <corps>
                  <alternative>
                    <condition>
                      <operationBinaire operateur='&lt;'>
                        <operandeGauche>
                          <variable nom='discriminant'/>
                        </operandeGauche>
                        <operandeDroit>
                          <flottant valeur='0.0'/>
                        </operandeDroit>
                      </operationBinaire>
                    </condition>
                    <consequence>
                      <chaine>discriminant negatif : aucune racine</chaine>
                    </consequence>
                    <alternant>
                      <alternative>
                        <condition>
                          <operationBinaire operateur='&gt;'>
                            <operandeGauche>
                              <variable nom='discriminant'/>
                            </operandeGauche>
                            <operandeDroit>
                              <flottant valeur='0.0'/>
                            </operandeDroit>
                          </operationBinaire>
                        </condition>
                        <consequence>
                          <chaine>discriminant positif : deux racines</chaine>
                        </consequence>
                        <alternant>
                          <chaine>discriminant nul : une seule racine</chaine>
                        </alternant>
                      </alternative>
                    </alternant>
                  </alternative>
                </corps>
              </blocUnaire>
            </corps>
          </blocUnaire>
        </corps>
      </blocUnaire>
    </corps>
  </blocUnaire>
</programme1>
```

Lorsque vous saurez écrire des tests JUnit (voir plus bas), vous pourrez écrire les tests appropriés pour votre programme de discriminant : voir section 6.

5 Test JUnit

Objectif : Écrire un test selon les conventions de JUnit 4 et d'ILP.

Buts

— Comprendre les tests et suites de test en ILP1
— Écrire un test JUnit4 pour ILP1

Les liens

Java	`http://java.sun.com/`
JUnit	`http://www.junit.org/`
XMLUnit	`http://xmlunit.sourceforge.net/`

Les documents

JUnit Test Infected	`http://junit.sourceforge.net/doc/testinfected/testing.htm`
JUnit Cookbook	`http://junit.sourceforge.net/doc/coobook/cookbook.htm`

Fidèles à la devise du cours « on ne modifie pas le code précédemment donné », les extensions à ILP respecteront toujours les conventions suivantes. Nous vous demandons de vous y conformer (notamment pour les examens).

— toute extension est effectuée au sein d'un nouveau paquetage (généralement nommé `fr.upmc.ilp.ilpItmeN` où N est le numéro du TME et I le niveau du langage ILP que vous étendez. Pour ce premier TME, ce sera donc `ilp1tme1`.
— aucune modification (sauf stipulation contraire explicite) n'est permise dans les sources d'ILP.

L'interface enrichie `fr.upmc.ilp.ilp1tme1.IASTsequence` vous est fournie.

```
1 package fr.upmc.ilp.ilp1tme1;
2
3 import fr.upmc.ilp.ilp1.fromxml.ASTException;
4 import fr.upmc.ilp.ilp1.interfaces.IAST;
5
6 public interface IASTsequence extends fr.upmc.ilp.ilp1.interfaces.IASTsequence {
7     IAST[] getAllButLastInstructions() throws ASTException;
8 }
```

L'extension proposée est d'étendre l'`ASTsequence` pour implanter cette interface enrichie afin de procurer une méthode `getAllButLastInstructions` utile à certains endroits de l'interprète ou du compilateur. Il faut donc créer un paquetage nommé `fr.upmc.ilp.ilp1tme1` et une telle classe. Pour pouvoir utiliser cette classe, il faut écrire un analyseur syntaxique à partir d'`ASTParser`. Pour tester le tout, il faut une classe de test mettant à l'épreuve votre nouvelle méthode `getAllButLastInstructions` et vérifiant que vous n'avez rien cassé de ce qui était auparavant fait. Cette classe de tests sera nommée `fr.upmc.ilp.ilp1tme1.ASTsequenceTe`

5.1 Travail à réaliser

— Recenser toutes les classes de test présentes dans les sources d'ILP1
— Pour chacun des ces tests, déterminer si c'est du JUnit3 ou JUnit4, déterminer ce qui est testé, puis lancer le test depuis Eclipse.
— Écrire la classe `ASTsequenceTest` (en JUnit4) puis la classe `ASTsequence` répondant minimalement à ce test. Indice : XMLUnit est bien utile pour comparer des documents XML.
— Vérifier ensuite que si vous substitutez votre classe `ASTsequence` à la mienne, tous les tests d'ILP1 fonctionnent toujours. Pour ce faire, il faut s'intéresser à la classe `ASTParser`.

***Pour enseignant*:** Voir sous svn, `fr.upmc.ilp.ilp1tme1`. Les points importants sont :
— utiliser l'héritage : à la fois pour ASTsequence, ASTParser et ASTParserTest.

— La méthode doit échouer sur les séquences vides : il faut donc écrire le test qui vérifie que cela échoue.
— Pas de croix rouge ni de triangle jaune dans les sources.

fr/upmc/ilp/ilp1tme1/ASTsequence.java

```
package fr.upmc.ilp.ilp1tme1;

import java.util.List;

import fr.upmc.ilp.ilp1.fromxml.AST;
import fr.upmc.ilp.ilp1.fromxml.ASTException;
import fr.upmc.ilp.ilp1.interfaces.IAST;

public class ASTsequence
extends fr.upmc.ilp.ilp1.fromxml.ASTsequence
implements fr.upmc.ilp.ilp1tme1.IASTsequence {

    public ASTsequence(List<AST> instructions) {
        super(instructions);
    }

    public IAST[] getAllButLastInstructions() throws ASTException {
        int length = this.getInstructionsLength();
        try {
            IAST[] _instructions = new IAST[length-1];
            for (int i = 0; i<length-1; i++) {
                _instructions[i] = this.getInstruction(i);
            }
            return _instructions;
        } catch (Exception e) {
            throw new ASTException(e.getMessage());
        }
    }
}
```

fr/upmc/ilp/ilp1tme1/ASTParser.java

```
package fr.upmc.ilp.ilp1tme1;

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

import fr.upmc.ilp.ilp1.fromxml.AST;
import fr.upmc.ilp.ilp1.fromxml.ASTException;

public class ASTParser extends fr.upmc.ilp.ilp1.fromxml.ASTParser {

      /** Analyser le DOM pour convertir les sequences et laisser l'analyseur
       * dont on herite traiter tout le reste. */

    @Override
    public AST parse (Node n) throws ASTException {
        switch ( n.getNodeType() ) {

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

```
23
24          if ( "sequence".equals(name) ) {
25              return new ASTsequence(this.parseList(nl));
26          } else {
27              return super.parse(n);
28          }
29        }
30
31        default: {
32            return super.parse(n);
33        }
34        }
35    }
36 }
```

fr/upmc/ilp/ilp1tme1/test/ASTsequenceTest.java

```
1  package fr.upmc.ilp.ilp1tme1.test;
2
3  import java.io.StringReader;
4
5  import javax.xml.parsers.DocumentBuilder;
6  import javax.xml.parsers.DocumentBuilderFactory;
7
8  import org.junit.Before;
9  import org.w3c.dom.Document;
10 import org.w3c.dom.Element;
11 import org.w3c.dom.Node;
12 import org.w3c.dom.NodeList;
13 import org.xml.sax.InputSource;
14
15 import fr.upmc.ilp.ilp1.fromxml.AST;
16 import fr.upmc.ilp.ilp1.fromxml.ASTException;
17 import fr.upmc.ilp.ilp1.fromxml.test.ASTParserTest;
18 import fr.upmc.ilp.ilp1.interfaces.IAST;
19 import fr.upmc.ilp.ilp1tme1.ASTsequence;
20 import fr.upmc.ilp.ilp1tme1.IASTsequence;
21
22 public class ASTsequenceTest extends ASTParserTest {
23
24        /* NOTA: On recopie mon ASTParser afin de n'avoir, pour l'exercice,
25         * qu'une unique dependance sur l'ASTsequence des etudiants. */
26
27    static class ASTParser extends fr.upmc.ilp.ilp1.fromxml.ASTParser {
28
29        /** Analyser le DOM pour convertir les sequences et laisser l'analyseur
30         * dont on herite traiter tout le reste. */
31
32        @Override
33        public AST parse (Node n) throws ASTException {
34            switch ( n.getNodeType() ) {
35
36            case Node.ELEMENT_NODE: {
37                Element e = (Element) n;
38                NodeList nl = e.getChildNodes();
39                String name = e.getTagName();
40
41                if ( "sequence".equals(name) ) {
42                    return new ASTsequence(this.parseList(nl));
```

```
                } else {
                    return super.parse(n);
                }
            }

            default: {
                return super.parse(n);
            }
            }
        }
    }

    /* Utilitaire pour convertir une chaine XML en AST. */
    public static AST toAST(ASTParser parser, String xml)
    throws ASTException {
        try {
            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            DocumentBuilder db = dbf.newDocumentBuilder();
            StringReader sr = new StringReader(xml);
            InputSource is = new InputSource(sr);
            Document d = db.parse(is);
            AST ast = parser.parse(d);
            return ast;
        } catch (ASTException exc) {
            throw exc;
        } catch (Throwable cause) {
            throw new ASTException(cause);
        }
    }

    @Override
    @Before
    public void setUp () {
        this.parser = new ASTParser();
    }
    private ASTParser parser;

    /* Test sur une sequence de 2 termes */
    @Override
    public void testSequence () throws Exception {
        String program = "<sequence>"
              + "<booleen valeur='true'/>"
              + "<entier valeur='1'/>"
              + "</sequence>";
        AST a = toAST(parser, program);
        assertEquals(program, a.toXML());
        assertTrue(a instanceof IASTsequence);
        IASTsequence iast = (IASTsequence) a;
        IAST[] iastlist = iast.getInstructions();
        assertNotNull(iastlist);
        assertTrue(iastlist.length == 2);
        IAST e1 = iastlist[0];
        assertXMLEqual("<booleen valeur='true'/>", ((AST)e1).toXML());
        IAST e2 = iastlist[1];
        assertXMLEqual("<entier valeur='1'/>", ((AST)e2).toXML());
        // Les tests supplementaires:
        IAST[] iastbl = iast.getAllButLastInstructions();
        assertEquals(1, iastbl.length);
```

```
101         assertXMLEqual("<booleen valeur='true'/>", ((AST)iastbl[0]).toXML());
102     }
103 
104     /* Test sur une sequence de 3 termes pour verifier que les termes
105      * restant sont bien dans le bon ordre. */
106     public void testSequenceThree () throws Exception {
107         String program = "<sequence>"
108               + "<booleen valeur='true'/>"
109               + "<entier valeur='1'/>"
110               + "<entier valeur='11'/>"
111               + "</sequence>";
112         AST a = toAST(parser, program);
113         assertEquals(program, a.toXML());
114         assertTrue(a instanceof IASTsequence);
115         IASTsequence iast = (IASTsequence) a;
116         IAST[] iastlist = iast.getInstructions();
117         assertNotNull(iastlist);
118         assertTrue(iastlist.length == 3);
119         IAST e1 = iastlist[0];
120         assertXMLEqual("<booleen valeur='true'/>", ((AST)e1).toXML());
121         IAST e2 = iastlist[1];
122         assertXMLEqual("<entier valeur='1'/>", ((AST)e2).toXML());
123         // Les tests supplementaires:
124         IAST[] iastbl = iast.getAllButLastInstructions();
125         assertEquals(2, iastbl.length);
126         assertXMLEqual("<booleen valeur='true'/>", ((AST)iastbl[0]).toXML());
127         assertXMLEqual("<entier valeur='1'/>", ((AST)iastbl[1]).toXML());
128     }
129 
130     /* Test sur une sequence d'un terme seulement */
131     @Override
132     public void testSequenceOne () throws Exception {
133         String program = "<sequence>"
134             + "<entier valeur='1'/>"
135             + "</sequence>";
136         AST a = toAST(parser, program);
137         assertEquals(program, a.toXML());
138         assertTrue(a instanceof fr.upmc.ilp.ilp1.fromxml.ASTsequence);
139         IASTsequence iast = (IASTsequence) a;
140         IAST[] iastlist = iast.getInstructions();
141         assertNotNull(iastlist);
142         assertTrue(iastlist.length == 1);
143         IAST e1 = iastlist[0];
144         assertXMLEqual("<entier valeur='1'/>", ((AST)e1).toXML());
145         // Les tests supplementaires:
146         IAST[] iastbl = iast.getAllButLastInstructions();
147         assertEquals(0, iastbl.length);
148     }
149 
150     /* Test sur une sequence vide: exception attendue! */
151     //@Override
152     public void no_testSequenceEmpty () throws ASTException {
153         String program = "<sequence>"
154             + "</sequence>";
155         AST a = toAST(parser, program);
156         assertEquals(program, a.toXML());
157         assertTrue(a instanceof fr.upmc.ilp.ilp1.fromxml.ASTsequence);
158         IASTsequence iast = (IASTsequence) a;
```

```
159         IAST[] iastlist = iast.getInstructions();
160         assertNotNull(iastlist);
161         assertTrue(iastlist.length == 0);
162         // Les tests supplementaires:
163         try {
164             iast.getAllButLastInstructions();
165             fail("Pas de getAllButLastInstructions(sequence vide)!");
166         } catch (ASTException e) {
167             // OK
168         }
169     }
170 }
```

6 Base de tests

Objectif : Enrichir la base de tests d'ILP.

Buts

— Comprendre les suites de test en ILP1
— Enrichir la base des programmes ILP de test

6.1 Travail à réaliser

— Ajouter quelques programmes ILP1 à la base de tests pour le programme calculant le discriminant que vous avez écrit plus haut en indiquant les résultats et/ou impressions attendus.
— Tester que l'ensemble ainsi enrichi de programmes continue à passer `WholeTestSuite`.

***Pour enseignant*:** Il faut nommer ces programmes `u*.xml`, les inclure dans `Grammars/Samples/` et ajouter les fichiers `.result` et `.print` associés.

On peut également les placer dans le paquetage `ilp1tme1` mais alors il faut hériter de `ProcessTest` en spécifiant le répertoire et la regexp ciblant ces fichiers.

7 Annexes diverses

7.1 Commandes

Si vous souhaitez utiliser la ligne de commande, il peut être utile de définir quelques variables d'environnement.

— Mettre les archives des outils en Java dans le chemin des classes :
Sous bash :

```
1 export ILPDIR=$HOME/workspace/ILP
2 export JING=$ILPDIR/Java/jars/jing.jar ;
3 for arch in $ILPDIR/Java/jars/*.jar ; do
4     CLASSPATH=$arch:$CLASSPATH ;
5 done
6 export CLASSPATH
```

7.2 Initialisation automatique de l'interprète de commandes

Les initialisations de variables globales de l'interprète de commandes (*shell*) que vous venez de faire sont valables uniquement dans l'instance d'interprète de commandes dans lequel vous les avez exécutées. Si vous lancez un nouvel interprète de commandes (par exemple, en ouvrant une nouvelle fenêtre terminal), il faut refaire les opérations pour initiliser les variables de cet interprète (modulo les interprètes dépendants qui reçoivent les liaisons de variables exportées).

Pour éviter de refaire ces commandes et plutôt initialiser automatiquement toutes les instances d'interprète lancées, il faut utiliser le fichier d'initialisations `.bashrc`. Ce fichier doit se trouver dans votre répertoire de base ("*HOME directory*"). Vous pouvez donc y ajouter les lignes précédentes. Faites cependant attention, il y a probablement déjà un tel fichier qui y a été placé par défaut lors de la création de votre compte. Éditez donc ce fichier plutôt que d'en créer un autre en l'écrasant.

7.3 Chemin de classes explicite en Java

Le fait de mettre les archives Java dans le chemin des classes défini par l'interprète de commandes (*shell*) fait en sorte que lors de l'exécution de la plupart des programmes de la galaxie Java (`javac`, `java`, ...) qui vont suivre, ces outils sauront retrouver les classes correspondantes pour les exécuter. Le problème est que d'autres outils auront aussi besoin de ces classes mais avec d'autres versions. Une bien meilleure solution consiste à passer explicitement ce chemin de classes à Java lors de son appel avec l'option `-cp` ou `-classpath`, sous la forme :

```
java -cp <chemin-de-classes> MaClasseAExecuter
```

Dans certains cas, les archives sont auto-exécutables, ce qui évite de préciser la classe à exécuter. L'inconvénient de cette approche est de devoir repréciser le chemin des classes à chaque appel à Java. On peut circonvenir cela en se définissant soi-même une variable pour contenir le chemin des classes et l'utiliser explicitement (comme le fait par exemple le script `ilprun`) :

```
ILP_CP=<chemin-des-classes-pour-ILP>
java -cp ${ILP_CP} -jar ${FICHIER_JAR}
```

Cela peut paraître équivalent à l'utilisation du chemin de classes standard (spécifié par la variable (globale) CLASSPATH), mais l'avantage est de pouvoir gérer indépendamment plusieurs chemins des classes selon le projet Java sur lequel on travaille. Lorsqu'on fait beaucoup de Java, cela permet d'éviter les inévitables conflits entre les chemins de classes des différents projets. De même réduire la taille du chemin de classes accélère la recherche des classes par Java.