# 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.
- À l'exception des clés USB en lecture seule, 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, avec en plus une question bonus.
- Votre travail pour l'examen sera fait dans l'espace de travail (*workspace*) Eclipse ILP1 dans lequel vous créerez le paquetage `fr.upmc.ilp.ilp1partiel2008` c'est-à-dire le répertoire `$HOME/workspace/ILP1/Java/src/fr/upmc/ilp/ilp1partiel2008` qui contiendra les programmes (non nécessairement compilables) que vous aurez écrit ainsi que dans le fichier textuel `$HOME/workspace/ILP1/Java/src/fr/upmc/ilp/ilp1partiel2008/partiel.txt` qui contiendra le reste. Des consignes plus précises seront énoncées pour chaque question. Plus généralement, tout fichier ou répertoire dont le nom contiendra `partiel2008` sera ramassé pour constituer votre copie. Rien d'autre ne sera ramassé !
- Vos fichiers textuels seront codés en ISO-Latin 1 ou 15, ils seront formés de lignes ne dépassant pas 78 caractères pour en faciliter la lecture.

## Le problème : les commandes gardées

Le concept de commande gardée est dû à Dijkstra dans le contexte de la programmation parallèle où il est intimement lié à l'idée d'exécution non-déterministe. Une alternative gardée est une instruction de la forme générale :

$$\textbf{if } G_1 \rightarrow C_1 \,||\, ... \,||\, G_n \rightarrow C_n \textbf{ fi}$$

où les $G_i$ sont des expressions booléennes appelées *gardes* et les $C_i$ sont des commandes (instructions). Dans l'esprit de Dijkstra, la sémantique de cette alternative consiste à évaluer les gardes, puis à choisir de manière non-déterministe une instruction parmi celles dont la garde est vraie pour l'exécuter.

Par exemple, en Ada, si trois tâches `A1`, `A2` et `A3` peuvent appeler une tâche `T` sur des entrées `Ready1`, `Ready2` et `Ready3` respectivement, l'alternative gardée suivante permet à `T` d'accepter l'un des appels selon que les tâches sont dans l'état `"OK"` :

```
select when A1status = OK => accept Ready1 do ... end; // Exemple Ada
       when A2status = OK => accept Ready2 do ... end;
       when A3status = OK => accept Ready3 do ... end;
```

Le non-déterminisme permet à ce programme Ada de choisir uniquement l'un des appels si plusieurs des trois tâches sont dans l'état `"OK"` et que plusieurs appels sont faits simultanément. La réalisation du non-déterminisme n'est pas chose simple. Aucun ordinateur à ce jour est capable d'un choix non-déterministe, car nous ne connaissons pas de phénomène naturel simple qui exhibe du non-déterminisme (on songe actuellement à l'émission de particules par certains matériaux sous l'effet de l'absorption de photons, mais ce n'est pas facile à connecter à un processeur...). De plus, toute implantation du non-déterminsme devrait assurer l'équité entre les choix possibles (ceux dont la garde est vraie).

Dans cet examen, nous vous demandons d'introduire une instruction d'alternative gardée à ILP1 selon une sémantique très simple qui consiste à toujours choisir la première (ou la dernière) instruction gardée dont la garde est vraie. Le choix sera décidé selon une variable booléenne dans l'interprète/compilateur. L'alternative gardée qui choisit toujours la première instruction est similaire au `cond` du langage Scheme.

# Question 1

(3 points)

Proposez une syntaxe concrète XML pour cette nouvelle instruction en ILP1 sous la forme d'un exemple qui sera exécutable après la réalisation de cette extension à ILP1. Vous ferez apparaître en commentaire en tête du fichier le pseudo-code equivalent avec la syntaxe que vous souhaitez.

## Livraison

Un fichier de programme exemple `$HOME/workspace/ILP1/Grammars/Samples/u01-partiel2008.xml`.

# Question 2

(3 points)

Définissez l'extension nécessaire à la grammaire d'ILP. Vous pouvez ajouter les commentaires qui vous semblent pertinents.

## Livraison

Un fichier `$HOME/workspace/ILP1/Grammars/grammar1-partiel2008.rnc`.

# Question 3

(3 points)

Décrivez les modifications à faire (les interfaces ou classes à créer, les méthodes ou données appropriées, etc.) pour réaliser l'analyse syntaxique des programmes utilisant cette nouvelle instruction. Ces modifications peuvent figurer soit dans le code (non nécessairement compilable), soit dans le fichier textuel `$HOME/workspace/ILP1/Java/src/fr/upmc/ilp/ilp1partiel2008/partiel.txt` (en prenant soin d'indiquer le numéro de la question à laquelle vous répondez et de référencer les fichiers annexes.

## Livraison

Le fichier
`$HOME/workspace/ILP1/Java/src/fr/upmc/ilp/ilp1partiel2008/eval/EASTParser.java` conte-

nant votre classe d'analyseur ainsi que les fichiers
`$HOME/workspace/ILP1/Java/src/fr/upmc/ilp/ilp1partiel2008/eval/IEASTFactory.java` et
`$HOME/workspace/ILP1/Java/src/fr/upmc/ilp/ilp1partiel2008/eval/EASTFactory.java`.

## Question 4

(4 points)

Définissez l'interprétation de la nouvelle instruction. Vous pouvez ajouter les commentaires qui vous semblent pertinents.

### Livraison

Un fichier
d'`$HOME/workspace/ILP1/Java/src/fr/upmc/ilp/ilp1partiel2008/eval/EASTAlternativeGardee.java`
contenant la classe Java complète implantant le noeud d'arbre de syntaxe abstraite. Vous pouvez vous contenter de ne donner que la définition de la méthode `eval`.

## Question 5

(4 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 méthode pour engendrer le code C.

### Livraison

Votre fichier
d'`$HOME/workspace/ILP1/Java/src/fr/upmc/ilp/ilp1partiel2008/cgen/Cgenerator.java` avec en commentaire dans ce fichier le schéma de compilation pour votre alternative gardée. Vous n'oublierez pas qu'un zeste d'ASCII-art peut épargner un long discours ou même un code manquant. Ce commentaire est bien plus important que le code associé qui ne sera lu qu'en dernier recours.

## Question 6

(3 points)

L'alternative gardée, réalisée sous la sémantique du `cond` de Scheme n'est en réalité que du sucre syntaxique pour un ensemble d'alternatives imbriquées l'une dans l'autre. Cela veut dire qu'il n'est pas nécessaire de savoir interpréter ou compiler une alternative gardée ; il suffirait, en transformant l'arbre de syntaxe abstraite, immédiatement après sa création (dans la méthode `prepare` de `Process`), de remplacer les nœuds d'alternative gardée par des sous-arbres qui représentent les alternatives corespondantes.

En vous inspirant de la classe `Cgenerator`, écrire une classe `TransformationAlternativeGardee` qui réalise cette transformation. Pensez à expliquer votre stratégie de transformation.

### Livraison

Votre fichier
d'`$HOME/workspace/ILP1/Java/src/fr/upmc/ilp/ilp1partiel2008/cgen/TransformationAlternativeGardee.`

## Question bonus

La transformation de l'arbre de syntaxe abstraite n'est que l'une des façons de se débarasser de l'alternative gardée avant d'en arriver à l'interprétation et la compilation vers C. Pourriez-vous suggérer une autre façon de se débarrasser de cette instruction entre le programme en XML (où elle apparaît) et ces deux phases de traitement du programme ? Expliquez.

### Livraison

Rédigez votre réponse (du texte, pas besoin de code) avec des explications claires et précises dans le fichier `$HOME/workspace/ILP1/Java/src/fr/upmc/ilp/ilp1partiel2008/partiel.txt`.