Méthodes de Conception de Programmes
Projet “Machines de Turing”

Baudouin Le Charlier – 2013

1 Objectifs du projet

Ce projet de groupe poursuit deux objectifs principaux.

1. Vous faire réfléchir à l'utilité de l'approche “méthodologique” proposée par le cours de MCP dans la réalisation d'un projet impliquant plusieurs personnes et nécessitant de se répartir le travail.

2. Faire un lien avec le cours de Calculabilité, pour montrer notamment que la construction rigoureuse de programmes a son utilité dans toute les branches de l'“informatique”, les plus abstraites comme les plus terre à terre.

2 Résumé du projet

Le thème général du projet est la notion de *Machine de Turing*. Cette notion est définie et largement utilisée dans le cours de Calculabilité. Toutefois, la programmation ou construction “concrète” de machines de Turing n'y est abordée qu'“en passant”. Au cours de ce projet, vous constaterez qu'une machine de Turing peut être construite en appliquant les mêmes notions que pour les algorithmes au sens habituel : spécifications, décomposition en sous-problèmes, invariants, variants.

Une machine de Turing est en fait un algorithme manipulant une structure de donnée unique : son ruban. Il vous sera donc demandé de réaliser une classe Java représentant le “type logique” *Ruban d'une machine de Turing*. En parallèle, vous construirez un ensemble de machines de Turing abstraites que vous pourrez “coder” en Java en utilisant la classe implémentant le ruban.

Le projet sera réalisé par groupes d'environ six étudiants, divisés en deux sous-groupes de trois étudiants, appelés les sous-groupes A et B. Chaque sous-groupe aura trois tâches à réaliser : une tâche de spécification, une tâche d'implémentation et une tâche de vérification. Ainsi, le sous-groupe B implémentera ce qui aura été spécifié par le sous-groupe A et celui-ci vérifiera la correction et la qualité de cette implémentation. Et réciproquement, en échangeant A et B.

3 Tâches à réaliser

3.1 Spécification de l'implémentation du ruban (SGR A)

Cette tâche consiste à spécifier l'implémentation du type logique *Ruban d'une machine de Turing*. La spécification de ce type logique est donnée dans l'annexe A sous forme d'un interface Java. Les sous-tâches suivantes sont à réaliser.

1. Écrire le "squelette" d'une classe Java `TapeA_B` implémentant (au sens Java) l'interface Java `Tape` de l'annexe A. Ce squelette doit contenir uniquement les variables d'instances nécessaires à l'énoncé des conventions de représentation.

2. Énoncer de manière précise et complète *les conventions de représentation* permettant de représenter un ruban quelconque par une instance de la classe `TapeA_B`. Ces conventions doivent être aussi adéquates que possible pour permettre une implémentation efficace des opérations prévues sur le ruban sans utiliser une quantité excessive de mémoire.

   Remarque super importante : Les conventions de représentation doivent être énoncé de façon "déclarative", purement "logique", excluant toute forme d'explication "algorithmique". Les algorithmes seront écrits par le sous-groupe B, en se basant sur les conventions de représentations données. Toute infraction à cette consigne enlève tout intérêt et toute valeur à la réalisation de cette tâche.

3. Énoncer *l'invariant de classe* devant être respecté par toute instance de la classe `TapeA_B` pour représenter correctement un ruban de machine de Turing (notion logique, "abstraite"). En cas de doute à ce sujet, vous pouvez consulter le livre : *Barbara Liskov et John Guttag : Program Develoment in Java, Abstraction, Specification, and Object-Oriented Design*, pages 102–107. À nouveau, cette description de l'invariant de classe ne doit pas être algorithmique mais bien déclarative, logique.

4. Construire (avec rigueur, selon les méthodes du cours) le code Java d'une méthode `boolean repOk()` capable de tester si une instance quelconque, non nécessairement "correcte", de la classe `TapeA_B` respecte l'invariant de classe. (Voir le livre, ci-dessus, pages 105–106.) Cette méthode **ne doit pas** être communiquée au sous-groupe B, chargé de l'implémentation de la classe. Elle pourra servir au sous-groupe A à tester l'implémentation du groupe B.

5. Rédiger un rapport reprenant les points 1. à 3., ci-dessus. Ce rapport sera envoyé par mail, en format pdf, par le sous-groupe A, à chaque membre du sous-groupe B, accompagné du fichier `TapeA_B.java` contenant le "squelette" de la classe. Attention, ce fichier ne doit pas contenir le code de la méthode `repOk()`.

6. Les mêmes documents seront transmis aux quatre enseignants du cours, avec la modification que le squelette de la classe contiendra le code (documenté) de la méthode `repOk()`.

3.2 Découpe de quelques machines de Turing (SGR B)

Cette tâche consiste à réaliser une découpe en sous-problèmes pour la construction de quatre machines de Turing réalisant les opérations arithmétiques d'addition, soustraction, multiplication et division sur les nombres entiers naturels, représentés dans le système binaire. Des spécifications de ces quatre machine de Turing sont données à l'annexe B.

De manière plus explicite, on demande de *préparer un rapport*, à destination du sous-groupe A, chargé de l'implémentation des machines, contenant les informations suivantes.

1. Une vue générale de la découpe en sous-problèmes, présentant un ensemble de sous-problèmes (sous-machines de Turing) et expliquant leur rôle dans la résolution du problèmes général. Par exemple, on pourra songer à utiliser des sous-machines qui réalisent les quatre opérations dans le système unaire (on représente un nombre n par une suite de n symboles identiques). Il faudra alors prévoir des algorithmes de conversion entre le système binaire et le système unaire. D'une manière générale, il faut que la découpe en sous-problèmes soit suffisamment fine pour que chaque sous-machine soit très simple et comporte idéalement une seule boucle.

2. Une spécification précise de chaque sous-machine de Turing identifiée au point précédent. On supposera que chaque sous-machine de Turing est réalisée en Java sous forme d'une méthode ayant une en-tête comme la suivante :

   ```
   static void nomDeMachine(Tape t) throws Exception
   ```

   Une telle méthode possède donc un et un seul paramètre de type `Tape`, rien d'autre. Elle ne renvoie pas de valeur. Le code de la méthode ne peut utiliser que les instructions suivantes :

   - des appels aux méthodes de l'interface `Tape` ;
   - des instructions `if` et `while` ;
   - des appels à d'autres (sous)-machines de Turing.

   La spécification d'une telle méthode se réduit nécessairement à une précondition et à une postcondition sur l'état du ruban. Il faudra donc faire attention à ce que ces préconditions et postconditions soient suffisamment générales pour que la méthode puisse être utilisée comme un sous-problème, c'est-à-dire que le ruban puisse contenir des données qui n'ont pas de lien direct avec celles de la méthode et qui restent inchangées après l'appel de celle-ci.

Comme précisé plus haut, une machine de Turing n'utilise aucune autre structure de données que son ruban. Il est donc interdit de déclarer et d'utiliser des variables locales pour maintenir de l'information sur l'exécution de la machine de Turing. Toute l'information doit être placée sur le ruban !! Néanmoins, pour faciliter la programmation, on autorisera l'utilisation (en petit nombre) de variables booléennes. Celles-ci peuvent être utilisées dans les conditions d'un `if` ou d'un `while`. On peut seulement leur affecter des constantes (`false` ou `true`).

3.3 Construction du code des machines de Turing (SGR A)

Cette tâche consiste à construire les différentes méthodes (machines de Turing) spécifiées par le sous-groupe B (voir paragraphe 3.2). Chaque méthode doit, en principe, être construite pour être *rigoureusement* conforme à la spécification donnée par le sous-groupe B. Toutefois, on ne peut exclure que ces spécifications contiennent des erreurs, des imprécisions ou qu'elles ne soient pas adéquates pour que la méthode soit utilisée comme sous-problème d'une autre méthode ("l'erreur est humaine"). Si cela se produit, il est demandé que le sous-groupe A contacte les membres du sous-groupe B pour que ceux-ci "corrigent le problème". Il faudra alors que le sous-groupe A mentionne clairement la liste des modifications effectuées aux spécifications du sous-groupe B (en principe par le sous-groupe B, lui-même).

À l'issue de cette étape, le sous-groupe A fournira ce qui suit :

1. Un fichier `MTA_B.java` contenant le code commenté de toutes les machines de Turing réalisées. Les commentaires à fournir seront (a) les spécifications précises et complètes de chaque méthode, (b) un invariant précis et complet pour chaque boucle while, ainsi qu'un variant. On pourra compléter les spécifications et les invariants avec des commentaires plus intuitifs expliquant le "but" de chaque méthode et de chaque boucle mais ces commentaires doivent être brefs et ne peuvent remplacer les spécifications ou les invariants.

   Le fichier `MTA_B.java` doit être envoyé aux membres du sous-groupe B pour qu'il en vérifie la correction (voir tâche 3.6). Il doit également être envoyé aux enseignants du cours.

2. Un bref rapport, en format `pdf`, indiquant les problèmes éventuels rencontrés par le sous-groupe A dans la compréhension ou l'implémentation des spécifications du sous-groupe B, ainsi que les solutions apportées à ces problèmes. Ce rapport sera envoyé uniquement aux enseignants du cours.

3.4 Implémentation du ruban (SGR B)

Cette tâche est tout à fait semblable et comparable à la tâche 3.3. Je la décrirai donc plus succinctement (voir le paragraphe précédent pour plus de précisions).

La tâche consiste à implémenter la classe `TapeA_B` en respectant les conventions de représentation et l'invariant de classe définis par le sous-groupe A. En cas de problèmes rencontrés dans la compréhension des éléments fournis par le sous-groupe A, une "concertation" avec celui-ci sera organisée.

À l'issue de cette étape, le sous-groupe B fournira ce qui suit :

1. Le fichier `TapeA_B.java` complété et contenant désormais le code commenté complet de la classe. Les commentaires à fournir seront du même genre que précisé dans la description de la tâche 3.3.

Le fichier `TapeA_B.java` doit être envoyé aux membres du sous-groupe A pour qu'il en vérifie la correction (voir tâche 3.5). Il doit également être envoyé aux enseignants du cours.

2. Un bref rapport, en format `pdf`, indiquant les problèmes éventuels rencontrés par le sous-groupe B dans la compréhension ou l'implémentation des spécifications du sous-groupe A, ainsi que les solutions apportées à ces problèmes. Ce rapport sera envoyé uniquement aux enseignants du cours.

3.5 Vérification de l'implémentation du ruban (SGR A)

Cette tâche consiste à analyser et à vérifier la correction du code final de la classe `TapeA_B.java` fourni par le sous-groupe B. L'essentiel du travail sera d'analyser le code Java fourni, du point de vue de sa conformité aux conventions de représentation et à l'invariant de classe. Des tests d'exécution seront réalisés uniquement dans le but d'illustrer les erreurs découvertes dans l'inspection du code commenté. En particulier, le sous-groupe A pourra ajouter dans le fichier `TapeA_B.java` le code de sa propre méthode `repOk()` et des appels à cette méthode afin de vérifier que l'invariant de classe est respecté à l'entrée et à la sortie de chaque méthode publique de la classe (et à la sortie du constructeur).
À l'issue de cette tâche, le sous-groupe A produira un rapport contenant les éléments suivants.

1. Pour chaque méthode publique, suppose-t-elle bien que l'invariant de classe est vérifié en entrée ? Assure-t-elle bien qu'il est respecté en sortie (constructeur aussi) ? Sinon, donnez des contre-exemples concrets sous forme de tests à l'exécution dans lesquels la méthode `repOk()` renvoie `false`.

2. Pour chaque méthode publique (ou constructeur), en supposant l'invariant de classe respecté, donne-t-elle un résultat correct, traite-t-elle correctement les cas limites éventuels ? Sinon donnez des contre-exemples sous forme de tests concret d'exécution.

3. Pour chaque méthode interne, privée, sa spécification est-t-elle précise et complète ? Est-elle correcte par rapport à cette spécification ? Sinon, expliquez pourquoi et donnez des contre-exemples.

4. Pour chaque boucle, vérifiez la présence d'un invariant. Est-il précis et complet ? Est-il respecté par le code ? Sinon, donnez des contre-exemples.

5. Pour chaque boucle, vérifiez la présence d'un variant. Est-il correct ?

6. Une liste des problèmes observés qui ne seraient pas repris dans la liste des points précédents.

Le rapport sera envoyé aux enseignants du cours.

3.6 Vérification du code des machines de Turing (SGR B)

À nouveau, cette tâche est semblable à la précédente (3.5). Il s'agit d'analyser le contenu du fichier `MTA_B.java` fourni par le sous-groupe A et de produire un rapport contenant les éléments suivants.

1. Pour chaque (sous)-machine de Turing donne-t-elle un résultat correct, traite-t-elle correctement les cas limites éventuels ? Est-elle utilisable comme sous-problème ? Sinon donnez des contre-exemples sous forme de tests concrets d'exécution.

2. Pour chaque boucle, vérifiez la présence d'un invariant. Est-il précis et complet ? Est-il respecté par le code ? Sinon, donnez des contre-exemples.

3. Pour chaque boucle, vérifiez la présence d'un variant. Est-il correct ?

4. Une liste des problèmes observés qui ne seraient pas repris dans la liste des points précédents.

Le rapport sera envoyé aux enseignants du cours.

4 Encadrement et évaluation

Chacun des assistants du cours prendra en charge et évaluera un certain nombre de groupes formés chacun d'un sous-groupe A et d'un sous-groupe B. L'assistant est à votre disposition pour vous fournir toutes les explications nécessaires et vous devez recourir à lui pour tout problème de compréhension. Il évaluera de son côté chacun de vos rapports en accordant la plus grande importance aux points suivants. Avez-vous respecté toutes les consignes pour la tâche considérée ? Votre rapport est-il complet, clair et précis ? Avez-vous traité le problème en vous conformant à la philosophie du cours ? Dans certains cas, vous devez utiliser ou évaluer le travail réalisé par un autre sous-groupe dans une tâche précédente. Cette tâche précédente aura déjà été évaluée par l'assistant en charge de votre groupe. Il examinera donc si votre utilisation du travail de l'autre sous-groupe ou vos critiques éventuelles de ce travail sont pertinentes. Les conclusions de cet examen seront prises en compte pour l'évaluation de votre propre sous-groupe.

5 Échéances

Date	Activité
Jeudi 25 avril (10h45)	TP “machines de Turing” et séance de questions/réponses sur les tâches 3.1 et 3.2 pour les groupes 1, 2, 4, 5, 7, 10.
Jeudi 25 avril (14h)	TP “machines de Turing” et séance de questions/réponses sur les tâches 3.1 et 3.2 pour les groupes 8, 9, 12, 14, 15, 16.
Vendredi 26 avril (14h)	TP “machines de Turing” et séance de questions/réponses sur les tâches 3.1 et 3.2 pour les groupes 3, 6, 11, 13, 17.
Dimanche 28 avril (18h)	Envoi des rapports concernant les tâches 3.1 et 3.2.
Lundi 29 avril (10h45)	Remise d’une “version papier” des rapports concernant les tâches 3.1 et 3.2, au début du cours.
Jeudi 2 mai (10h45)	Séance de questions/réponses sur les tâches 3.3 et 3.4 pour les groupes 1, 2, 4, 5, 7, 10.
Jeudi 2 mai (14h)	Séance de questions/réponses sur les tâches 3.3 et 3.4 pour les groupes 8, 9, 12, 14, 15, 16.
Vendredi 3 mai (14h)	Séance de questions/réponses sur les tâches 3.3 et 3.4 pour les groupes 3, 6, 11, 13, 17.
Dimanche 5 mai (18h)	Envoi des rapports concernant les tâches 3.3 et 3.4.
Lundi 6 mai (10h45)	Remise d’une “version papier” des rapports concernant les tâches 3.3 et 3.4, au début du cours.
Jeudi 9 mai (10h45)	Séance de questions/réponses sur les tâches 3.5 et 3.6 pour les groupes 1, 2, 4, 5, 7, 10.
Jeudi 9 mai (14h)	Séance de questions/réponses sur les tâches 3.5 et 3.6 pour les groupes 8, 9, 12, 14, 15, 16.
Vendredi 10 mai (14h)	Séance de questions/réponses sur les tâches 3.5 et 3.6 pour les groupes 3, 6, 11, 13, 17.
Dimanche 12 mai (18h)	Envoi des rapports concernant les tâches 3.5 et 3.6.
Lundi 13 mai (10h45)	Remise d’une “version papier” des rapports concernant les tâches 3.5 et 3.6, au début du cours.

A Spécification du type logique Ruban (Tape)

```
public interface Tape{

/* Spécification :
   -------------
 Une instance de (toute classe implémentant l'interface) Tape représente
 un ruban de machine de Turing moyennant les conventions, précisions et
 restrictions ci-dessous.

 * Un ruban est une suite de cases infinie "dans les deux sens", munie
   d'une tête de lecture qui désigne une case particulière, appelée la "case
   sous la tête de lecture".

 * Chaque case du ruban contient un (unique) symbole appartenant à un certain
   alphabet Gamma.

 * Nous imposons l'utilisation d'un alphabet Gamma particulier : Gamma est
   l'ensemble des caractères imprimables du code ASCII, personnifiés en Java
   par les valeurs de type char comprises entre 32 (code du caractère "espace")
   et 126 (code du caractère ~), incluses.

 * L'alphabet contient un caractère spécial appelé caractère "blanc" et noté B.
   Nous imposons le caractère "espace" (code 32) comme caractère blanc.

 * Toutes les cases du ruban sont blanches sauf un nombre fini d'entre elles.

 Les opérations disponibles sur un ruban sont spécifiées ci-dessous.
 Outre les opérations "classiques" d'écriture, lecture et déplacement de
 la tête de lecture, on dispose d'une opération pour vérifier la cohérence
 de l'implémentation du ruban et une autre pour afficher son contenu de façon
 "lisible".
 */

 public boolean repOk() ;
 // Résultat : true si la structure de données vérifie l'invariant de
 // représentation ; false, sinon.

 public boolean isSymbol(char s) throws Exception ;
 // Résultat : true si s est le symbole sous la tête de lecture,
 //   false si s est un symbole permis différent de celui sous la tête de lecture.
 // Une exception est lancée si s n'est pas un symbole permis.
 // Le ruban n'est pas modifié (en aucun cas).
```

```
 public void putSymbol(char s) throws Exception ;
 // Une exception est lancée si s n'est pas un symbole permis ;
 // dans ce cas, le ruban n'est pas modifié.
 // Le caractère s est écrit sous la tête de lecture si c'est un symbole permis ;
 // les autres cases du ruban sont inchangées, dans ce cas.

 public void leftMove() ;
 // Déplace la tête de lecture d'une case vers la gauche.

 public void rightMove() ;
 // Déplace la tête de lecture d'une case vers la droite.

 public String toString() ;
 // résultat : une représentation du ruban sous la forme
 //
 // alpha[s]beta
 //
 // où s est le symbole sous la tête de lecture,
 // alpha est un suffixe de la suite des symboles placés avant la tête de lecture,
 // beta  est un préfixe de la suite des symboles placés après la tête de lecture,
 // alpha et beta contiennent tous les symboles non blancs du ruban (sauf s bien sûr).
 // NB. Cette représentation est ambiguë dans certains cas.

 /* Constructeur. */
 /* ------------- */
 // Toutes les classes qui implémentent cet interface doivent contenir
 // un unique constructeur vérifiant la spécification ci-dessous :
 //
 // public TapeA_B(String init) throws Exception
 //
 // Si init ne contient que des caractères qui sont des symboles valides,
 // ce constructeur crée un ruban contenant la suite de caractères init,
 // complétée à gauche et à droite par des caractères blancs.
 // En outre, la tête de lecture est placée sur la case située
 // exactement après le dernier symbole de init.
 //
 // Une exception est lancée si init contient au moins un caractère qui n'est pas
 // un symbole valide (ou si init == null).

}
```

B Spécifications des machines de Turing à construire

```
Note préliminaire. Dans les spécifications ci-dessous, on note B
le caractère "blanc" de l'alphabet du ruban. B désigne donc, en pratique,
selon nos conventions pour les classes Tape, le caractère ASCII de code 32,
c'est-à-dire l'espace, et non la lettre B majuscule, de code 66.
-------------------------------------------------------------------------

static void add(Tape t) throws Exception
// But : Additionner deux nombres binaires
// Pré : le ruban contient deux suites binaires de même longueur
//   séparées par un B ; la tête de lecture est positionnée
//   à droite du dernier bit.
//   Exemple  : 00100 10101[ ]
// Post : le ruban contient une suite binaire représentant la somme
//   des deux nombres naturels représentés par les suites binaires
//   placées initialement sur le ruban ; la tête de lecture est placée
//   à droite du dernier bit.
//   Exemples : 11001[ ]
//   Remarque : les suites placées sur le ruban au départ sont effacées,
//   le ruban ne contient que le résultat de l'addition.

static void subtract(Tape t) throws Exception
// But : Soustraire un nombre binaire d'un autre nombre binaire
// La spécification est identique à celle de add() sauf qu'au lieu
// d'additionner on soustrait le deuxième nombre (le plus à droite)
// du premier. Si le second nombre est plus grand que le premier,
// on renvoie zéro (une suite de 0).

static void multiply(Tape t) throws Exception
// But : Multiplier deux nombres binaires
// La spécification est identique à celle de add() sauf qu'au lieu
// d'additionner deux nombres, on les multiplie.

static void divide(Tape t) throws Exception
// But : Diviser un nombre binaire par un autre nombre binaire
// La spécification est semblable à celle de add() sauf qu'on divise
// le premier nombre (le plus à gauche) par le second. Après exécution,
// le ruban contient deux suites binaires représentant le quotient et
// le reste de la division (quotient à gauche, reste à droite), séparés
// par un B. La tête de lecture est placée à droite du dernier bit
// du reste.
```