Ecole Polytechnique Fédérale de Lausanne

Projet de Semestre Bachelor

Emulateur de Machine de Turing

Auteur :
Ludovic Favre

Superviseur :
Mahdi Cheraghchi

5 juin 2009

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Résumé

Ce document vous présente mon projet de semestre dont le titre original est *Turing machine emulator*. Le but de ce projet a été d'implémenter un système capable d'émuler certaines abstractions des machines de Turing.
La première étape était de mettre en place un "langage de programmation simple" pour décrire le comportement d'une machine de Turing et ensuite implémenter un interpreteur pour ce langage.
Les étapes suivantes ont permis d'écrire des modules simples qui implémentent des opérations basiques sur les machines de Turing ainsi qu'un "compilateur" pour traduire un language de plus haut-niveau permettant d'écrire des programmes basiques en des modules fonctionnant sur des machines de Turing ainsi que de mettre en place une interface d'utilisateur permettant l'interaction avec les différents systèmes.

Table des matières

Chapitre 1

Introduction

Avant de plonger au coeur de mon projet, voici un rappel des notions théoriques requises pour une bonne compréhension des différentes parties du projet.

1.1 Définition générale d'une machine de Turing

Une machine de Turing est une machine théorique inventée par Alan Turing (1937) qui sert de modèle abstrait au fonctionnement des appareils mécaniques de calcul, tel un ordinateur et sa mémoire.
Cette machine théorique a été créée en vue de donner une définition précise au concept d'algorithme ou "procédure mécanique".
Une machine de Turing consiste en [1] :
- une ligne de cellule également appelée "bande" qui peut être parcourue de gauche à droite ou de droite à gauche,
- un élément actif appelé "tête de lecture" (ou simplement tête), qui possède une propriété appelée "état",
- un ensemble d'instructions qui permet de décrire comment la tête doit modifier la cellule active et dans quel sens elle doit se déplacer sur la bande.

1.2 Définition formelle d'une machine de Turing

Une machine de Turing est un septuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ [2] où :
- Q est un ensemble fini d'états,
- Σ est l'alphabet d'entrée, qui ne contient pas le *symbol blanc* [3] '␣',
- Γ est l'alphabet de la bande, où '␣' $\in \Gamma$ et $\Sigma \subseteq \Gamma$,
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ est la fonction de transition,
- $q_0 \in Q$ est l'état de départ,
- $q_{accept} \in Q$ est l'état acceptant, et
- $q_{reject} \in Q$ est l'état rejetant.

1. Source Wikipedia [1] et Wolfram [4]
2. Définition tirée de "Introduction to the Theory of Computation" [3, p.142]
3. Le symbol blanc peut être n'importe quel symbol particulier choisi pour cette occasion

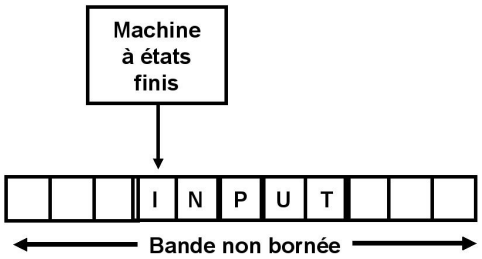

FIGURE 1.1 – Machine de Turing à bande doublement infinie

1.3 Machine de Turing à bande doublement infinie

Dans la définition de base d'une machine de Turing, la bande est bornée sur le côté gauche, c'est-à-dire qu'on ne peut plus effectuer de transition vers ce côté une fois la tête arrivée à l'extrémité gauche de la bande (par exemple, en début de bande, il n'est pas possible d'effectuer une transition vers la gauche). Une machine de turing à bande doublement infinie est en réalité sensiblement identique à une machine de Turing classique, à la seule différence que sa bande est infinie des deux côtés. On peut démontrer [4] que cette variante de machine de Turing est **équivalente** à une machine de Turing classique. La figure 1.1 illustre ce type de machines.

L'intérêt de considérer ce type de machine de Turing dans mon projet est principalement de rendre l'implémentations de modules plus aisée (dans la partie mettant en place le mini-langage de programmation) sans devoir se préoccuper des décalages de bande éventuellement nécessaires dans certains cas (décalages qui de toutes façons seraient réalisables en ajoutant des états et transitions spécifiques pour cette tâche), et ainsi rendre la description des machines de Turing beaucoup plus concise.

1.4 Machine de Turing à plusieurs bandes

Une machine de Turing à plusieurs bandes [5] est comme un machine de Turing ordinaire avec plusieurs bandes. Chacune de ces bandes a sa propre tête "indépendante" pour lire et écrire. La fonction de transition est modifiée pour permettre la lecture, l'écriture et le mouvement des têtes sur plusieurs ou toutes les bandes simultanément. Formellement, on a :

$$\delta : Q \times \Gamma^k \ \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$$

où k est le nombre de bandes.
A noter que le mouvement de la tête peut également être S, c'est à dire : ne rien faire. Une illustration de ce type de machines est montré en figure 1.2

4. Voir `http://www-users.itlabs.umn.edu/classes/Spring-2007/csci4011/lecture11.pps` [2]
5. Voir "Introduction to the Theory of Computation" [3, chapitre 3.2]

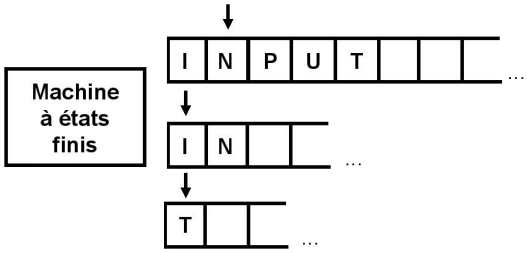

FIGURE 1.2 – Machine de Turing à plusieurs bandes

1.5 Equivalence des machines de Turing à plusieurs bandes et des machines de Turing à simple bande

Les machines de Turing à plusieurs bandes semblent être plus puissantes que les machines de Turing ordinaires. En réalité, on peut démontrer [6] que toute machine de Turing à plusieurs bandes a une machine de turing à simple bande équivalente (en utilisant le fait que deux machines sont équivalentes si elles reconnaissent le même langage).
L'avantage de considérer des machines de Turing à plusieurs bandes est de permettre une traduction plus lisible et aisée de certains problèmes travaillant sur plusieurs variables en machines de Turing (à plusieurs bandes).

6. Se référer à "Introduction to the Theory of Computation" [3, Théorème 3.13] pour la preuve complète

Chapitre 2

Le projet : Implémentation en Java

Le choix du langage de programmation s'est porté sur Java 1.6, étant donné que Java est le langage qui nous a été le plus enseigné durant ces trois années et qu'il possède la particularité d'être multiplateforme.

2.1 Implémentation d'un système d'émulation

La première phase de ce projet a été de programmer les différents éléments permettant d'abstraire une machine de Turing en Java. Pour ce faire, des classes telles que *Tape*,*State*,*Transition*,etc. ont été créées afin de séparer au mieux les parties de la machine représentant les attributs (ou l'état) de la machine de la partie simulation qui modifie l'état de la machine et procède aux transitions en fonction du contenu de la bande.

2.1.1 Machine de Turing Simple

Les classes principales pour ce type de machine sont :

- **Alphabet** : Une classe représentant un alphabet (pour la bande, ou la machine de turing),
- **SimpleTuringMachine** : Représente une machine de Turing ainsi que ses attributs,
- **State** : Représente un état (désigné par son label),
- **Tape** : Abstraction d'une bande,
- **Transition** : Représente un élément de la fonction de transition,
- **Direction** : Les directions possibles pour une transition,
- **SimpleTapeRunner** : Une classe permettant de simuler la machine de Turing une fois cette dernière initialisée.

2.1.2 Machine de Turing à plusieurs bandes

L'implémentation de la variante à plusieurs bandes reprend la majorité des éléments de la version simple. Les seules classes qui diffèrent sont :

- **MultiTapeDirection** : contient la direction complémentaire 'S',
- **MultiTapeRunner** : une classe permettant de simuler une machine à plusieurs bandes,
- **MultiTapeTransition** : transition ayant pour direction une MultiTapeDirection,

– **MultiTapeTuringMachine** : Représente la machine de Turing à plusieurs bandes ainsi que ses attributs.

2.2 Implémentation d'un système de sauvegarde

Une fois la simulation d'une machine de Turing simple implémentée, l'étape suivante a consisté à mettre en place un système de sauvegarde et de chargement pour ces machines de Turings. Le même genre de système a ensuite été repris pour la variante à plusieurs bandes.

2.2.1 Choix du langage

Afin de garantir une excellente flexibilité et compatibilité (avec d'autres langages de programmation que java par exemple) tout en conservant une lisibilité suffisante, mon choix s'est porté sur une description en XML des différents types de machine de Turing. Tout langage de programmation suffisament évolué proposant une API pour la lecture de données XML, ce choix permet donc l'utilisation des sources XML créées avec mon programme avec éventuellement d'autres langages comme le C++ sans avoir à implémenter un parseur complet.

2.2.2 Aperçu de la syntaxe générale

La syntaxe choisie a pour but de contenir un maximum d'informations tout en restant le plus lisible possible. L'arbre XML résultant peut être résumé comme suite :

- <turing> : racine avec plusieurs attributs
  - *type* : simple ou multitape
  - *variant* : bound ou unbound
  - *name* : un nom pour cette machine de Turing
  - *description* : Un description de cette machine
  - <input> avec un seul attribut : alphabet, contenant l'alphabet où les symboles sont séparés par une virgule
  - <tape> avec pour attributs :
    - *alphabet* : l'alphabet de la bande avec les symboles séparés par une virgule
    - *content* : le contenu par défaut de la bande
  - <states> : sous-arbre contenant les états
    - <state> état avec pour attribut un label permettant d'identifier cet état
  - <starting> avec pour seul attribut, le label de l'état de départ
  - <accepting> avec pour seul attribut, la liste des labels pour les états acceptant (séparés par une virgule)
  - <rejecting> avec pour seul attribut, la liste des labels pour les états non-acceptant (séparés par une virgule)
  - <transitions> : sous-arbre contenant les transitions
    - <transition> : une transition avec pour attributs :
      - $from\_state$: label de l'état de départ de la transition
      - *tapeSymbol* : symbol de la bande à lire pour cette transition
      - to_state : label de l'état final de la transition
      - $new\_tapeSymbol$: symbole de la bande à écrire pour cette transition

– *direction* : direction du mouvement de la tête de lecture ∈ {L,R} (ou ∈ {L,R,S} pour plusieurs bandes)

Syntaxe pour une machine de Turing Simple

Afin de rendre plus concrète la description de l'arbre XML faite ci-dessus, voici un exemple de fichier XML pour une machine de Turing simple :

```
<?xml version="1.0" encoding="UTF-8"?>
<turing type="simple" variant="bound" name="Tape-inverter" description="Une␣
    description␣de␣la␣machine␣de␣turing">
        <input alphabet="1,0,#"/>
        <tape alphabet="1,0,#" content="0110101001011#"/>
        <states>
                <state label="q0"/>
                <state label="q1"/>
                <state label="q2"/>
                ...
        </states>
        <starting state="q0"/>
        <accepting states="q2"/>
        <rejecting states="q1"/>
        <transitions>
                <transition from_state="q0" tapeSymbol="0" to_state="q1"
                    new_tapeSymbol="1" direction="R"/>
                <transition from_state="q0" tapeSymbol="1" to_state="q1"
                    new_tapeSymbol="0" direction="R"/>
                <transition from_state="q1" tapeSymbol="0" to_state="q1"
                    new_tapeSymbol="1" direction="R"/>
                ...
        </transitions>
</turing>
```

A noter que, dans mon implémentation, il peut y avoir plusieurs états acceptant et plusieurs états rejetant. Cette simplification se justifie par le fait qu'il est tout à fait possible de regrouper ces deux ensembles d'états en deux états en transformant légèrement la machine de Turing.

Syntaxe pour une machine de Turing à plusieurs bandes

La mise en place des machines de Turing à plusieurs bandes a nécessité plusieurs modifications de la syntaxe des fichiers de sauvegarde en plus des modifications requises dans le coeur du système.
Voici un exemple de fichier XML pour une machine de Turing à plusieurs bandes effectuant la comparaison du contenu de deux bandes :

```
<?xml version="1.0" encoding="UTF-8"?>
<turing type="multitape" variant="unbound" name="Compare two tape content"
    description="Simple comparator">
        <input alphabet="1,0,#"/>
        <states>
                <state label="q0"/>
                <state label="q1"/>
                <state label="qaccept"/>
                <state label="qreject"/>
        </states>
        <starting state="q0"/>
        <accepting states="qaccept"/>
        <rejecting states="qreject"/>
        <tapes>
          <tape indice="0" alphabet="1,0,#" content="1010010101010010101#">
            <transitions>
                    <transition from_state="q0" tapeSymbol="1" to_state="q1"
                        new_tapeSymbol="1" direction="R"/>
                    <transition from_state="q0" tapeSymbol="0" to_state="q0"
                        new_tapeSymbol="0" direction="R"/>
                    <transition from_state="q1" tapeSymbol="1" to_state="q1"
                        new_tapeSymbol="1" direction="R"/>
                    <transition from_state="q1" tapeSymbol="0" to_state="q0"
                        new_tapeSymbol="0" direction="R"/>
                    <transition from_state="q1" tapeSymbol="#" to_state="qaccept"
                        new_tapeSymbol="#" direction="R"/>
                    <transition from_state="q0" tapeSymbol="#" to_state="qaccept"
                        new_tapeSymbol="#" direction="R"/>
            </transitions>
          </tape>
          <tape indice="1" alphabet="1,0,#" content="1010010101010010101#">
            <transitions>
                    <transition from_state="q0" tapeSymbol="1" to_state="q1"
                        new_tapeSymbol="1" direction="R"/>
                    <transition from_state="q0" tapeSymbol="0" to_state="q0"
                        new_tapeSymbol="0" direction="R"/>
                    <transition from_state="q1" tapeSymbol="1" to_state="q1"
                        new_tapeSymbol="1" direction="R"/>
                    <transition from_state="q1" tapeSymbol="0" to_state="q0"
                        new_tapeSymbol="0" direction="R"/>
                    <transition from_state="q1" tapeSymbol="#" to_state="qaccept"
                        new_tapeSymbol="#" direction="R"/>
                    <transition from_state="q0" tapeSymbol="#" to_state="qaccept"
                        new_tapeSymbol="#" direction="R"/>
            </transitions>
          </tape>
        </tapes>
</turing>
```

Dans la version à plusieurs bandes du fichier XML, la principale différence se situe dans la gestion des transitions : chaque bande a maintenant sa fonction de transition.

2.3 Implémentation d'un langage utilisant le système d'émulation

Une fois les deux types de machines de Turing implémentés, la suite du projet a consisté à mettre en place une interface utilisateur ainsi qu'un "compilateur" pour traduire un language de plus haut-niveau permettant d'écrire des programmes basiques dans une abstraction sous forme de machines de Turing.

2.3.1 Spécification du langage

L'ensemble des Instructions

Pour ce langage simple, plusieurs instructions ont été choisies, dont voici la liste :

Instruction	Paramètres	Entrée(s)	Sortie(s)	Type	Exemple
ADD(*,*)	Variable,Variable	2	1	Simple-bande	ADD(X,Y)
ADDI(*,*)	Variable,Entier	2	1	Simple-bande	ADDI(X,12)
SUB(*,*)	Variable,Variable	2	1	Simple-bande	SUB(X,Y)
SUBI(*,*)	Variable,Entier	2	1	Simple-bande	SUBI(X,10)
ODD(*)	Variable	1	0	Simple-bande	ODD(X)
EVEN(*)	Variable	1	0	Simple-bande	EVEN(Z)
INC(*)	Variable	1	1	Simple-bande	INC(A)
DEC(*)	Variable	1	1	Simple-bande	DEC(B)
EQU(*,*)	Variable,Variable	2	0	Multi-bandes	EQU(A,B)
INIT(*,*)	Variable,Entier	2	0	-	INIT(X,100)
PRINT(*)	Variable	1	0	-	PRINT(X)
CALL_*	Variable	0	0	Mixte	CALL_XY
LABEL_*	Variable	0	0	-	LABEL_C
JUMPAR(*,*)	Variable	0	0	-	JUMPAR(E,F)
JUMPA(*)	Variable	0	0	-	JUMPA(L1)
JUMPR(*)	Variable	0	0	-	JUMPR(L2)
GOTO(*)	Variable	0	0	-	GOTO(L3)

Les instructions sont soit implémentées en utilisant une machine de Turing simple *(Simple-bande)*, soit en utilisant une machine de Turing à plusieurs bandes *(Multi-bandes)* ou alors implémentée "en dure" en java comme c'est le cas, par exemple, pour les sauts, les initialisations de variables ou encore l'affichage de la valeur des variables.

Sémantique des Instructions

Voici également la signification de ces instructions :

- ADD : Additionne le second paramètre du premier paramètre et mémorise le résultat dans le premier paramètre
- ADDI : Additionne au premier paramètre l'entier passé en second paramètre et mémorise le résultat dans le premier paramètre
- SUB : Soustrait le second paramètre du premier paramètre et mémorise le résultat dans le premier paramètre

– SUBI : Soustrait du premier paramètre l'entier passé en second paramètre et mémorise le résultat dans le premier paramètre
– ODD : Accepte si et seulemenent si la variable passée en paramètre est impaire, rejette dans le cas contraire
– EVEN : Accepte si et seulement si la variable passée en paramètre est paire, rejette dans le cas contraire
– INC : Incrémente la variable passée en paramètre
– DEC : Décrémente la variable passée en paramètre
– EQU : Accepte si et seulement si les deux variables ont les mêmes valeurs, rejette dans le cas contraire
– INIT : Initialise une variable passée en paramètre à la valeur donnée
– PRINT : Affiche la valeur de la variable passée en paramètre
– CALL : Exécute une procédure
– LABEL : Place un label dans le code
– JUMPAR : Saut à l'un des deux labels suivant le résultat de l'instruction précédente
– JUMPA : Saut au label passé en paramètre si l'instruction précédente a donné un "accept"
– JUMPR : Saut au label passé en paramètre si l'instruction précédente a donné un "reject"
– GOTO : Saut inconditionnel au label passé en paramètre

2.3.2 Syntaxe d'un programme

La syntaxe d'un programme se présente comme suit :

```
MAIN(){
 <instructions séparées par des points-virgules>
};
ROUTINE(<nom routine>){
 <instructions de la routine séparées par des points-virgules>
};
```

Le block principal (*MAIN()*) est l'entrée du programme. Il est également possible de définir des routines à appeler dans la fonction *main*, mais cette fonctionnalité n'a malheureusement pas pu être suffisament testée pour garantir une exécution sans problème.

2.3.3 Exemple de programme

Afin d'illustrer l'exécution ainsi que la syntaxe, voici le code d'un programme d'exemple qui affiche 1 si X (c'est-à-dire $9 + 1 = 10$) est impaire et 0 sinon :

```
MAIN(){
  INIT(X,9);
  INIT(A,1);
  INIT(B,0);
  INC(X);
  ODD(X);
  JUMPR(LB);
  PRINT(A);
  GOTO(END);
  LABEL_LB;
  PRINT(B);
  GOTO(END);
  LABEL_END;
};
```

L'exécution de ce code en utilisant l'interface graphique nous donne le résultat montré à la figure 2.1.

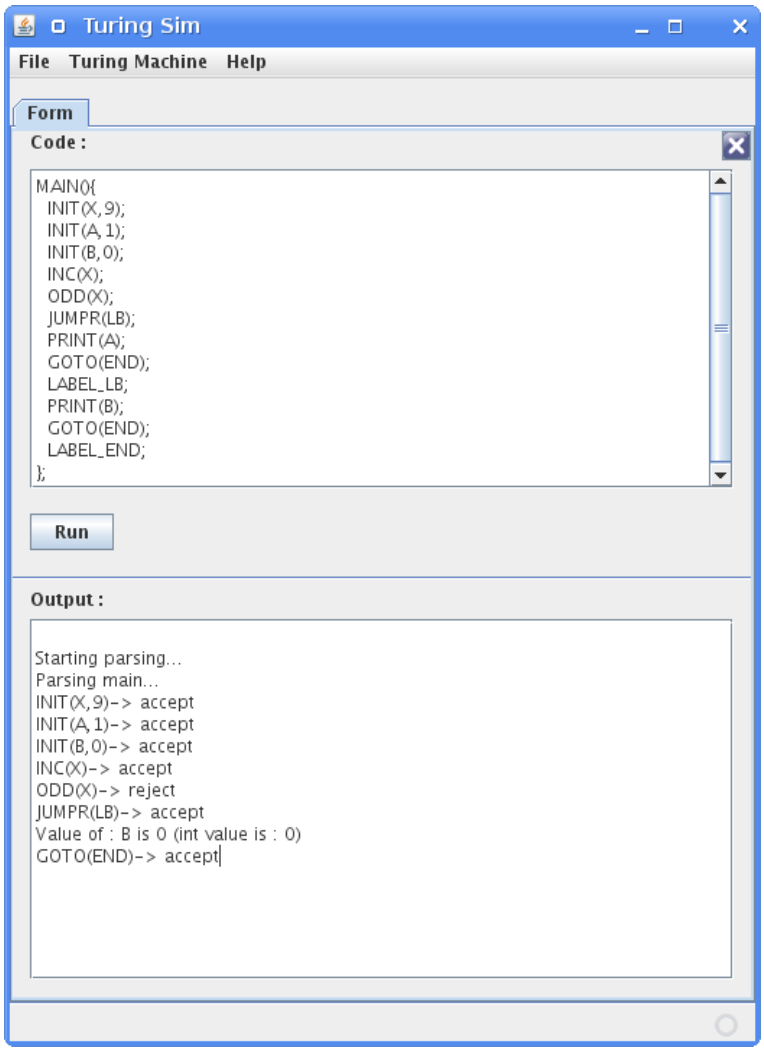

FIGURE 2.1 – Le résultat de l'exécution d'un "programme"

2.4 Spécificités de l'implémentation

Voici quelques spécificités liées au programme

Machines de Turing

Durant la mise en place de cet émulateur, certains choix on été faits pour permettre une plus grande flexibilité. Il faut notamment tenir compte des éléments suivants :

1. les émulateurs de machines de Turing sont considérés comme ayant rejeté l'entrée si leur état n'est pas acceptant. Ceci peut être vu comme une technique de *reject by default* afin de ne pas rester dans une situation incertaine sur l'état de l'émulateur,
2. il est possible de définir plusieurs états dans les ensembles acceptant ou rejetant afin de faciliter la construction d'une machine de Turing,
3. les machines de Turing à plusieurs bandes doivent avoir **toutes les transitions possibles** spécifiées pour chaque état, ceci afin de retirer tout ambiguité dans le programme,
4. l'ensemble des programme enregistrés sous forme XML sont représentés dans la variante *unbound* pour faciliter la simulation ainsi que la programmation des fonctions qu'ils incarnent.

Mini-Langage

A cause de la faible part qu'il a été possible de consacrer à la mise en place de ce mini-langage, il faut également tenir compte de certaines restrictions implicites :

1. Les paramètres fournis ne sont que très peu vérifiés : il faut prendre garde à ne fournir que des entiers positifs et des variables initialisées,
2. L'instruction *SUB* est en fait une variante de l'instruction *ADD* qui effectue la soustration en utilisant l'addition sur le complément à deux du nombre à soustraire. La valeur maximale possible pour un entier est de 512,
3. Les labels et sauts ne peuvent être utilisés que dans le bloque *main*.

Chapitre 3

Résultat

Le programme résultant de mon travail peut être consulté en téléchargeant le programme ainsi qu'au chapitre 4, où un guide vous permet de visualiser l'utilisation de l'interface du programme.

3.1 Idées d'ajouts futurs

Un travail lié au domaine aussi vaste que les machines de Turing m'a bien évidemment donné un grand nombre d'idées pour des ajouts qui seraient possibles sur ce programme ou en utilisant une partie de ce programme.
J'ai notamment retenu :

1. Affichage des machines de Turing sous forme de graphe d'état comme le permet JFLAP illustré à la figure 3.1
2. Edition de machines de Turing sous forme de graphe d'état
3. Implémenter un langage plus intermédiaire que celui implémenté dans ce projet comme par exemple : piloter la tête de lecture en lui demandant de scanner jusqu'au premier symbol #, lui faire faire un remplacement de toutes les occurences d'un symbole à partir d'une position, ...
4. Essayer d'implémenter un traducteur de machine à plusieurs bandes vers une machine à simple bande
5. Mettre en place d'autres variantes de machines de Turing

3.2 Licence

Etant donné que ce projet contient une part très importante de programmation et que dans ce cas, la question de la licence sous laquelle le code est fourni se pose, j'ai choisi, conformément à la règlementation en vigueur à l'EPFL [1], de publier le code sous licence GPLv3 [2]. Une des raisons principales est que le seul programme similaire (à ma connaissance) permettant d'émuler des machines de Turing est le logiciel JFLAP [3] dont les sources ne sont pas fournies ; je voulais donc permettre à des futurs développeurs d'éventuellement

1. `http://elle.epfl.ch/Publier-sous-licence-GPL-a-l-EPFL`
2. `http://gplv3.fsf.org/`
3. `http://www.jflap.org/`

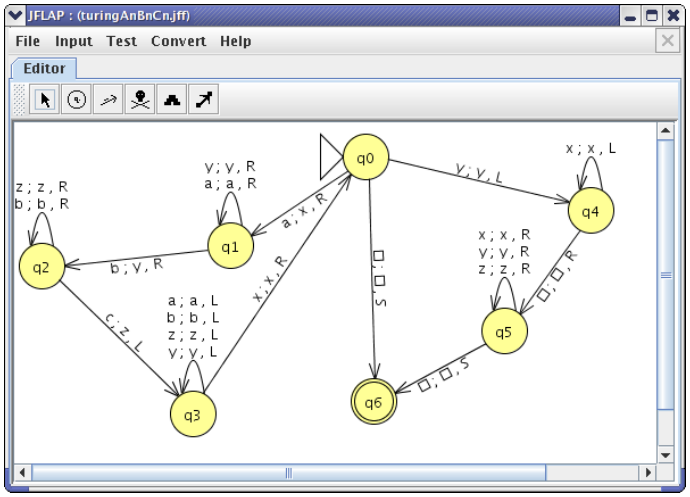

FIGURE 3.1 – Un aperçu de JFLAP

utiliser mon travail (sans prétention néanmoins de pouvoir concurrencer techniquement un logiciel développé depuis plus de 10 ans et impliquant plusieurs personnes dans son développement).

Chapitre 4

Guide d'utilisation du programme

Dans les pages qui suivent, figure un guide d'utilisation de l'interface graphique (GUI) ainsi que les procédures à suivre pour les trois principales fonctions de ce programme.
Le lancement du programme se fait en utilisant la commande

java -jar TuringSim.jar

4.1 L'interface générale

L'interface principale, une fois le programme lancé, consiste en une simple fenêtre permettant contenant des menus contextuels, telle qu'illustrée à la figure 4.1.
Une fois l'interface lancée, il est possible d'effectuer trois tâches principales :

1. Travailler avec une machine de Turing simple bande,
2. Travailler avec une machine de Turing à plusieurs bandes,
3. Utiliser le mini-langage de programmation.

Ces possibilités sont décrites dans les pages qui suivent.

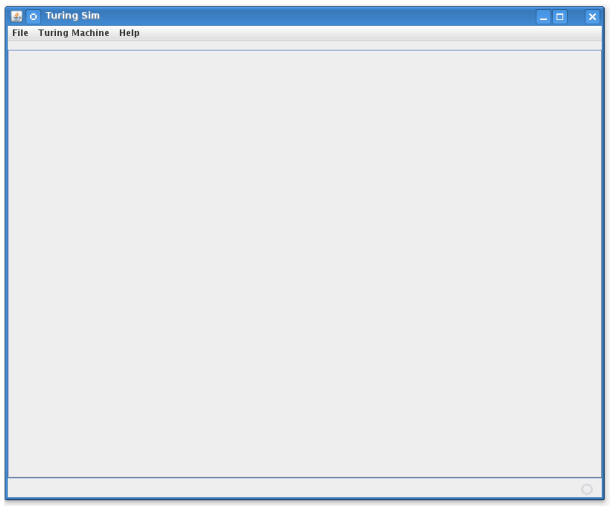

FIGURE 4.1 – La fenêtre principale du programme au lancement

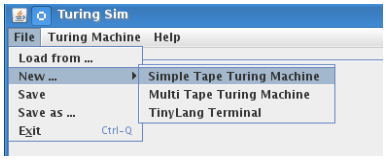

FIGURE 4.2 – Le menu "file"

4.2 Machine de Turing simple

4.2.1 Créer une nouvelle machine de Turing simple

Pour créer une nouvelle machine de Turing à simple bande, il suffit de choisir *"File→New→Simple Tape Turing Machine"* comme illustré à la figure 4.2.
Une fois la nouvelle machine de Turing créée, il suffit de l'éditer via l'interface graphique. Il faut cependant respecter un certain ordre dans l'édition pour pouvoir construire sa propre machine :

1. Il faut tout d'abord spécifier les deux alphabets via le menu *"Turing Machine → Edit Alphabet → ..."* comme montré à la figure 4.3
2. Une fois les alphabets choisis, il faut ajouter les états via le bouton *Add State* en spécifiant un état comme état de départ

Lorsque ces deux étapes ont été réalisées, il est dès lors possible de modifier le contenu de la bande ainsi que d'éditer les transitions.

4.2.2 Charger un fichier existant

Pour charger un fichier existant, il faut passer par le menu *"File→New→Simple Tape Turing Machine"* (cf figure 4.2). Une fenêtre apparaît alors pour permettre la sélection du fichier XML contenant la machine à charger (figure 4.4) et une fois un fichier choisi, ce dernier est chargé dans l'interface (figure 4.5).

FIGURE 4.3 – Le menu "édition de l'alphabet"

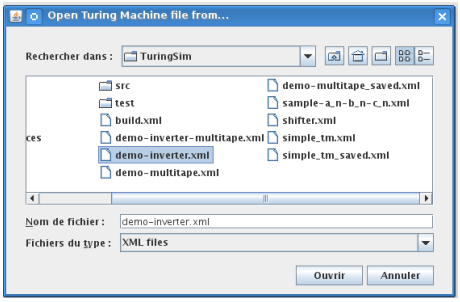

FIGURE 4.4 – Ouvrir une machine de Turing existante

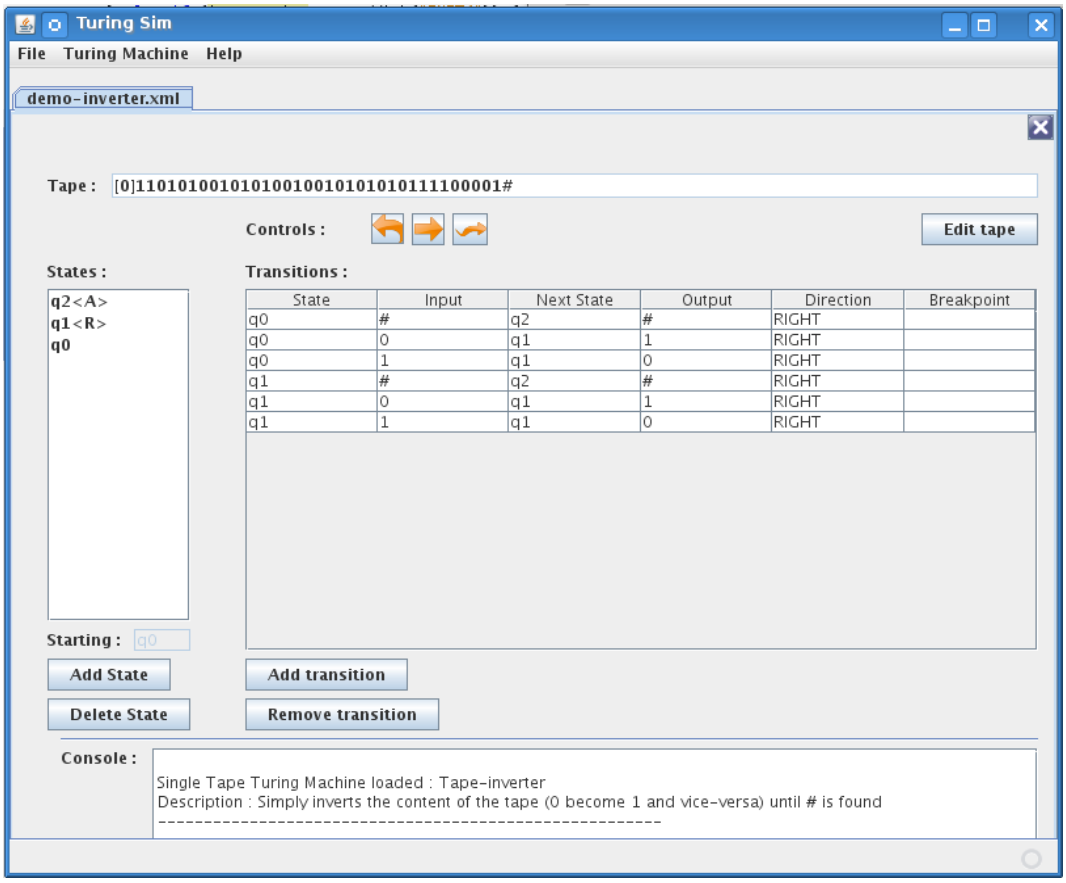

FIGURE 4.5 – Une machine de Turing simple une fois chargée

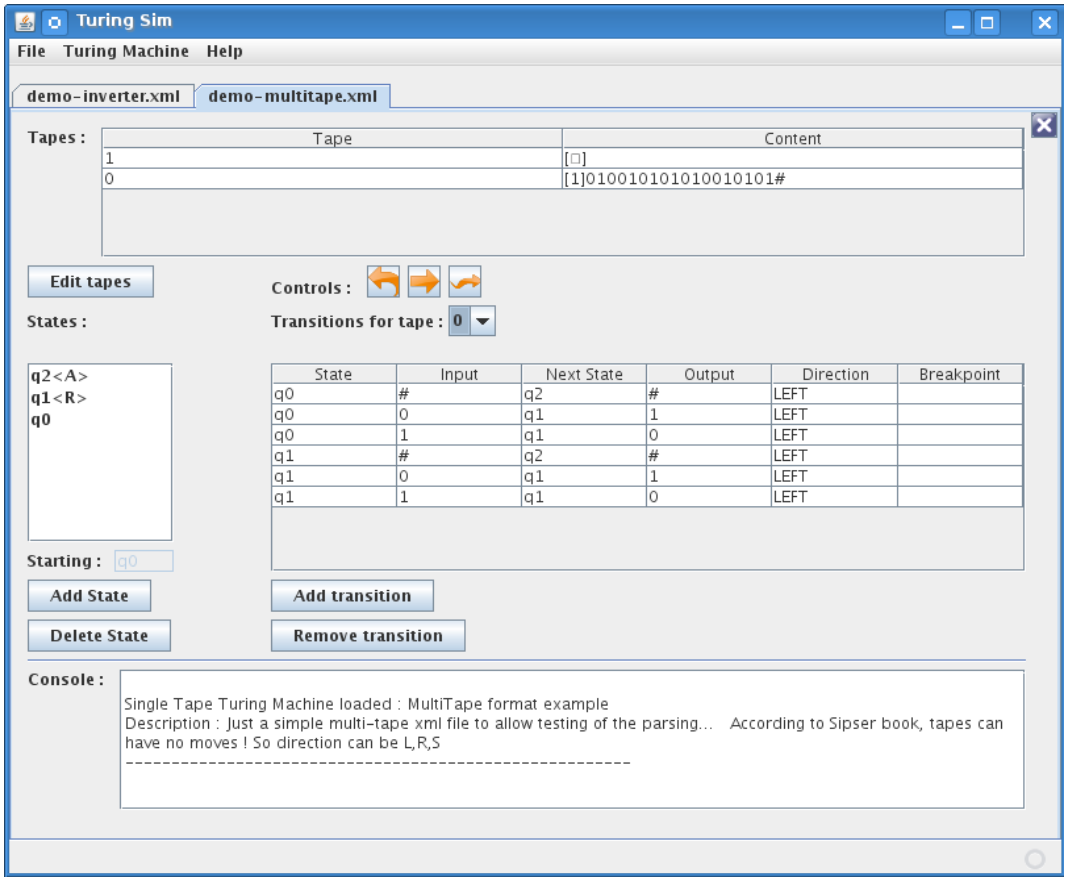

FIGURE 4.6 – Une machine à plusieurs bandes

4.3 Machine de Turing à plusieurs bandes

4.3.1 Créer une nouvelle machine de Turing à plusieurs bandes

La création d'une machine de Turing à plusieurs bandes suit le même protocole que pour une machine de Turing à simple bande, à la différence près qu'il faut spécifier, pour chaque bande, l'alphabet.
Une illustration d'une machine de Turing à plusieurs bandes une fois programmée est montrée à la figure 4.6

4.3.2 Charger un fichier existant

Le chargement de fichiers pour une machine à plusieurs bande s'effectue exactement de la même manière que pour une machine de Turing à simple bande.

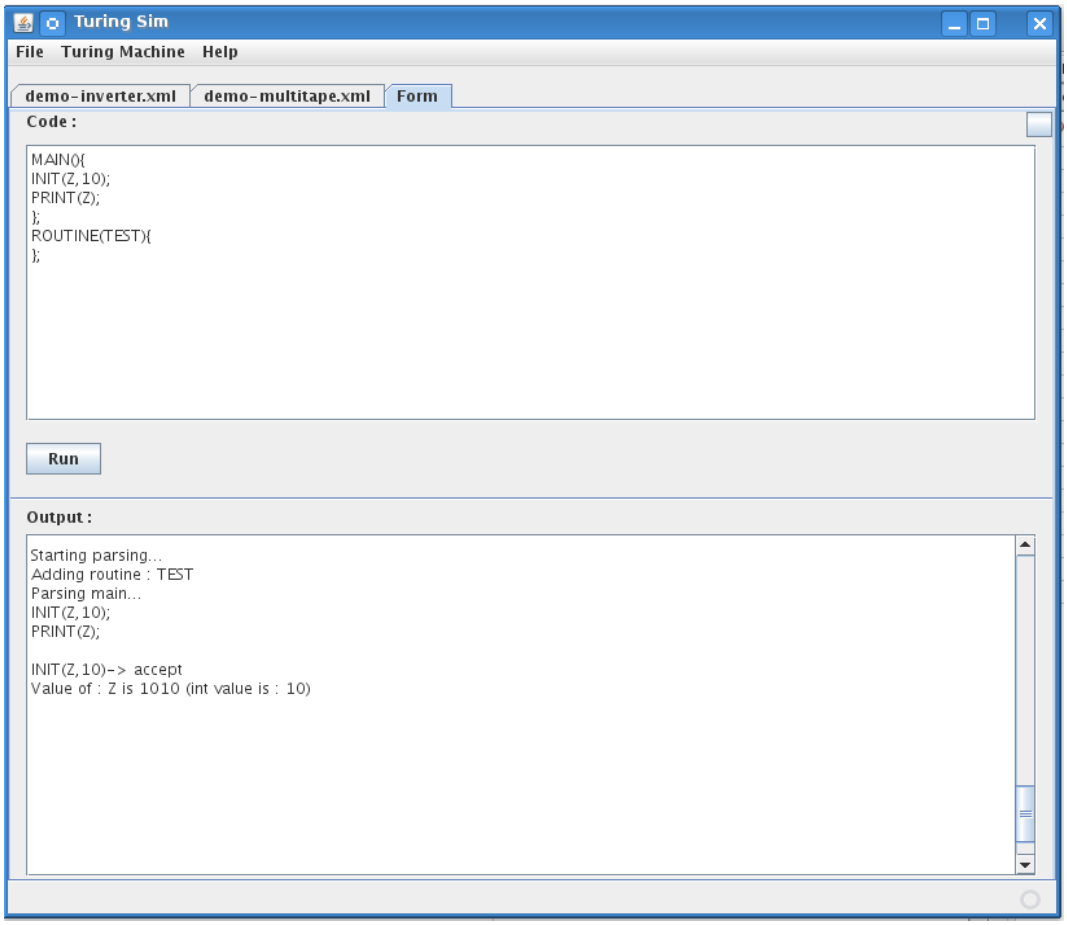

FIGURE 4.7 – Le fenêtre d'interaction avec le mini-langage

4.4 Mini-langage

L'utilisation de l'interface graphique pour le Mini-langage est très simple : Il suffit d'entrer le code dans la partie supérieure et de cliquer sur le bouton *Run* pour que le résultat soit affiché dans la zone de texte inférieure comme à la figure 4.7.

4.5 Utiliser les sources

Le projet a été réalisé en utilisant l'IDE NetBeans 6.5 [1] et le développement s'est effectué en sauvegardant les versions sur le service SVN fourni par polysvn [2]. Ils est donc fortement conseillé d'utiliser l'IDE NetBeans, en particulier pour la modification de l'interface graphique du programme.
A noter qu'il est également possible de n'utiliser que certains paquets comme par exemple le paquet *core* qui ne comprend par la GUI, dans le but de réaliser d'autres programmes utilisant des machines de Turing.

Aperçu des différents paquets de sources

Utilisation des sources

Récupérer le code source par SVN

Le code source du projet peut être récupéré à l'adresse :
`https://polysvn.ch/repos/TuringSim/trunk/TuringSim/`
Remarque : un accès est requis pour l'accès au dépot SVN.

Une fois les sources récupérées, il suffit d'ouvrir le projet NetBeans se trouvant dans le dossier */nbproject*.

Librairies requises

Les librairies requises pour le développement de ce projet sont :
– **log4j-1.1.15.jar** pour l'affichage de log
– **jdom.jar** pour la lecture des fichiers XML
– **junit-4.5.jar** pour les tests unitaires
A noter que la version de Java utilisée pour ce projet est la 1.6. Il se peut donc que les versions antérieures posent problèmes.

Les tests

Une batterie de tests est également disponible pour vérifier le bon fonctionnement du programme tout le long du développement et de l'ajout de fonctionnalités. La classe de test principale est *CoreImplementationTest.java* ; c'est elle qui se charge d'appeler tous les tests pour le coeur du projet.

1. `http://www.netbeans.org/`
2. `http://polysvn.ch/`

Chapitre 5

Conclusion

La réalisation de ce projet m'a permis de me confronter à un projet de taille relativement importante (principalement dû au fait que je l'ai réalisé seul à partir de rien) et ainsi de me mettre dans la situation où je devais faire des choix importants du point de vue de la conception (tels que le choix d'un langage, le choix d'un système de sauvegarde ou encore le choix d'une interface).
Je considère ce projet comme un excellent indicateur pour effectuer un bilan de mes trois ans de Bachelor. Indicateur qui m'a d'ailleurs permis de me rendre compte de l'importance capitale du bagage, aussi bien mathématique ou logique que lié au développement de logiciel, acquis durant ces années. Il m'a d'ailleurs fallu faire appel à plusieurs reprises à différentes techniques et astuces vues dans les cours que j'ai eu l'occasion de suivre jusqu'à maintenant.
J'ose espérer que le fait que le code source du programme soit mis à disposition sous licence GPLv3 permettra, peut-être, de servir de base à d'autres programmes similaires ou s'en inspirant.
Je tiens tout particulièrement à remercier M. Mahdi Cheraghchi pour m'avoir encadré durant ce projet.

Références

[1] http ://fr.wikipedia.org/wiki/machine_de_turing
(1er juin 2009).

[2] http ://www-users.itlabs.umn.edu/classes/spring-2007/csci4011/lecture11.pps
(2 juin 2009).

[3] Michael Sipser. *Introduction to the Theory of Computation, Second Edition.* Course Technology, February 2005.

[4] Wolfram. http ://mathworld.wolfram.com/turingmachine.html
(1er juin 2009).