UNIVERSITÉ
de Cergy-Pontoise
Département des Sciences Informatiques

Maîtrise d'informatique

Calculabilité, complexité & approximation

Projet – J.-F. Rey – 2004-2005

Introduction

Le but du projet est de réaliser en Java un simulateur de machines de Turing et de machines à registres comportant :

- une interface homme-machine soignée ;
- une composante d'édition des machines ;
- une visualisation graphique des machines ;
- une simulation graphique de l'exécution de ces machines.

Ce projet devra être réalisé par **groupe de quatre**, il comporte la réalisation du logiciel, le rapport d'analyse-programmation et la soutenance publique. Les soutenances de ce projet auront lieu dans la **semaine du 24 au 28 janvier 2005**. (Les partiels auront lieu dans la semaine du 17 au 21 janvier.)

Calendrier.

- La liste des groupes devra être constituée pour le vendredi 5 novembre 2004.
- Le rapport définitif sous forme papier ainsi qu'une version préliminaire du logiciel doivent être rendus au plus tard le vendredi 7 janvier.
- La version définitive du logiciel sur un CD-ROM doit être fournie au plus tard le jour de la soutenance.

Modalités d'évaluation.

- Le logiciel sera noté sur 10 points. Il devra comporter une procédure automatique de compilation et de génération de la documentation (Makefile, Javadoc). Il devra être indépendant de la plateforme de test (Unix, MacOs, Windows). Les sources seront encodés suivant le système ISO-8859-1 avec séparateur Unix. Tout logiciel présentant un défaut lors de la compilation sera très sévèrement sanctionné, il en sera de même de tout « emprunt »... L'interface homme machine doit être simple et conviviale pour l'utilisateur. La partie principale de la note portera sur les performances des algorithmes mis en œuvre.
- Le rapport écrit sera noté sur 5 points. Il doit expliquer dans le détail le travail que vous aurez accompli en précisant la méthode que vous avez employée, la répartition des tâches entre vous et les outils logiciels utilisés pour développer le projet. Il devra comporter un compte rendu du travail bibliographique accompli, des réunions d'harmonisation de l'équipe et une description argumentée des algorithmes choisis pour les points cruciaux du projet. Le listing des programmes peut faire l'objet d'une annexe éventuelle mais ne compte pas dans la notation du rapport.
- La soutenance sera notée sur 5 points. La note de soutenance est individuelle. Chaque participant du groupe devra soutenir la partie du projet sur laquelle il est intervenu principalement. La note attribuée prendra en compte le travail accompli lors des séances de TD tout au long du semestre. La soutenance est publique. Vous disposerez d'un rétro-projecteur et d'un vidéo-projecteur relié à une machine Linux comportant OpenOffice et les SDK Java-1.4 et Java-1.5.

Conseils. Ce projet nécessite de votre part un effort important vous devez y travailler régulièrement et dès maintenant. Répartissez vous les tâches, planifiez les étapes de développement, de rédaction du rapport. La maîtrise que vous allez acquérir dans la programmation en Java et dans l'interfaçage et l'utilisation de composants logiciels sera un plus important dans votre CV et dans votre recherche de stage. En effet, il y a peu de gens compétents à l'heure actuelle dans la réalisation d'interfaces réellement dynamiques et la demande augmente...

Édition des machines

Les machines devront pouvoir être saisies directement dans un éditeur de textes intégré au logiciel **et** lues directement à partir d'un fichier. Les textes seront encodés suivant le système ISO-8859-1 avec séparateur Unix. La saisie doit se faire « au kilomètre », mais un mode dirigé par la syntaxe peut être ajouté pour aider l'utilisateur en option. Pour réaliser l'éditeur de texte vous pouvez utiliser les API de SWING (par exemple JTextPane offre de nombreuses possibilités d'édition qui peuvent être utilisées pour la mise en sur-brillance des erreurs syntaxiques). Pour tester les algorithmes de placement de graphe qui sont au coeur du projet, il faut que votre logiciel puisse lire un fichier décrivant des machines ayant plusieurs milliers d'états. La validation syntaxique de la machine doit être faite avant le lancement de la simulation. Il n'est pas demandé de faire une saisie graphique des machines, ce sera à votre logiciel d'afficher sous forme graphique le graphe de la machine. Pour saisir les machines vous utiliserez obligatoirement des sur-langages des langages imposés pour le projet.

Pour faire l'analyse syntaxique et la génération de votre modèle interne de machine, vous pouvez utiliser les outils disponibles (JLex, JFlex, CUP, BYacc/J, ANTR,...).

Langage de description des machines à registres. La description d'une machine à registres est composées de deux parties : un ensemble de directives suivi d'un ensemble d'instructions.

Les fichiers pour les machines à registres auront pour extension `.reg`.

Les commentaires. La syntaxe sera identique à la syntaxe en java.

Les directives.

Directives	
Syntaxe	Signification
#Ri= BigNum	Valeur initiale du registre i. La valeur doit être un entier de taille quelconque non-signé.
#vitesse = entier	Indique un niveau de vitesse pour la simulation de 1 à 5.
#stop = $label_1, \cdots, label_n$	Indique des états où arrêter la machine (mode mise au point).
#nom = string	Nom de la machine à registres.

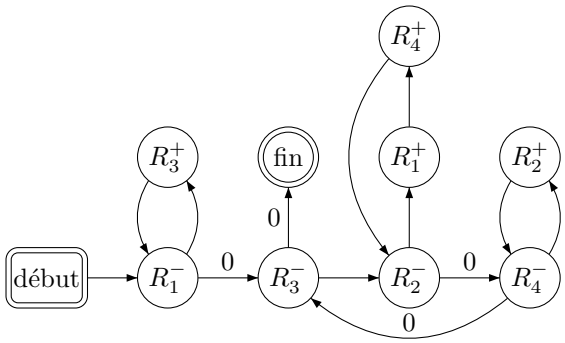
Les instructions.

Instructions	
Syntaxe	Signification
label :	Définit un sommet du graphe de la machine.
inc *entier* [*label*]	Incrémente le registre de numéro *entier* et branche au sommet de nom *label* ou à la ligne suivante si le label est omis.
dec *entier* [$label_0$] [, $label_p$]	Branche au sommet de nom $label_0$ (ou à la ligne suivante si le label est omis) si le contenu du registre de numéro *entier* est vide ; sinon décrémente le contenu du registre de numéro *entier* et branche au sommet de nom $label_p$ (ou à la ligne suivante si le label est omis).

Les noms de sommets obéissent à la syntaxe standard des identificateurs en Java. Les numéros de registres sont des entiers ordinaires (int positifs). Par contre les entiers contenus dans les registres peuvent être de taille arbitraire.

Vous pouvez étendre ce langage minimal, pour autoriser des appels à des sous-machines.

Exemple. Une machine à registres qui calcule la multiplication dont le graphe est :



sera saisie sous la forme :

```
/*
  Cette machine calcul le produit de deux nombres entiers x et y en utilisant
  la récursion :
     0.x = 0 et x.y = (x-1).y + y si x >0
*/

#nom = "calcul du produit"
#R1  = 123
#R2  = 13
debut :
   dec 1 boucle  //affecte  le registre R1 au registre R3 et vide R1
   inc 3 debut   //omission du second label
boucle:
   dec 3 fin    //omission du second label identique à  dec 3 fin, suite1
suite1 :
   dec 2 suite2
   inc 1
   inc 4 suite1
suite2 :
   dec 4 boucle
   inc 2 suite2
fin :
```

Langage de description des machines de Turing. La description d'une machine de Turing est composée de deux parties : un ensemble de directives suivi d'un ensemble d'instructions. Les bandes sont numérotées à partir de 0. La bande d'entrée est la bande numéro 0. La bande de sortie est la bande 1. Toutes les autres bandes sont des bandes de travail. Toute machine de Turing possède au minimum la bande 0.

Les fichiers pour les machines de Turing auront pour extension `.tur`.

Les commentaires. La syntaxe sera identique à la syntaxe en java.

Les directives.

Directives	
Syntaxe	Signification
#nbBandes = entier	Indique le nombre de bandes utilisées par la machine de Turing.
#alphabet = chaîne	Précise l'alphabet global utilisé.
#alphabet = entier, chaîne	Précise l'alphabet utilisé pour la bande de numéro entier.
#bande = chaîne	Indique le contenu de la bande d'entrée de la machine de Turing.
#vitesse = entier	Indique un niveau de vitesse pour la simulation de 1 à 5.
#stop = $etat_1, \cdots, etat_n$	Indique des états où arrêter la machine (mode mise au point).
#accepte = $etat_1, \cdots, etat_n$	Indique les états d'acceptation de la machine.
#rejette = $etat_1, \cdots, etat_n$	Indique les états de rejet de la machine.
#nom = string	nom de la machine de Turing

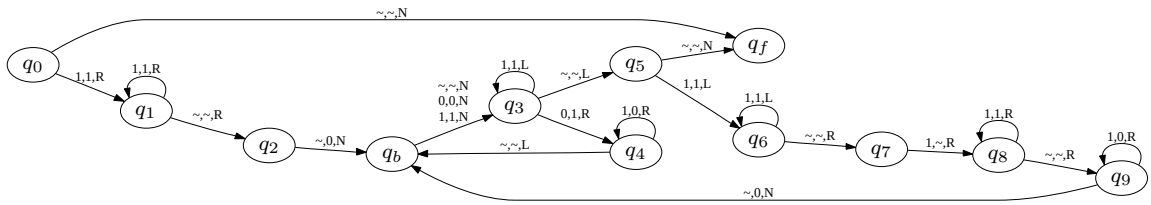
Seules les directives #nbBandes et #alphabet sont obligatoires dans le bloc de déclaration. Les entiers, chaînes et booléens obéissent à la syntaxe usuelle. Les noms d'états obéissent à la syntaxe usuelle des identificateurs en Java.

Les instructions.

Instructions	
Syntaxe	Signification
($etat_1$,b_1,l_1,e_1,d_1,$\cdots$,b_n,l_n,e_n,d_n,$etat_2$)	Si la machine est dans l'état $etat_1$ et si, pour tout i ($1 \leq i \leq n$), elle lit sur la bande b_i la lettre l_i, alors elle écrit le symbole e_i, elle effectue le déplacement d_i et elle passe dans l'état $etat_2$

où
- b_i est un entier représentant un numéro de bande ;
- l_i et e_i sont des caractères autorisés pour la bande b_i. On acceptera aussi le caractère blanc (l'espace) et le caractère * qui est une abréviation pour représenter n'importe quel autre caractère. La configuration $*,*$ signifie que le caractère lu est quelconque et n'est pas modifié. Le caractère , (virgule) sera saisi sous la forme \,. Vous pouvez ajouter d'autres séquences comme par exemple [a-z] pour représenter toutes les lettres comprises entre a et z ;
- d_i est le déplacement (L,R,N) les minuscules et majuscules seront indifférentes ici.

Exemple. La machine de Turing suivante :



sera décrite par le programme :

```
/*
Cette machine génère la suite de mots 0,1,00,01,10,11,000,001,010,011,...
pour l'ordre militaire elle s'arrête quand le dernier mot de taille n (donné
en unaire sur la bande initiale) est inscrit sur la bande.
*/
```

```
#nom      =  "Génération des suites de 2 symboles par ordre militaire"
#alphabet = "01" //L'espace fait toujours partie de l'alphabet
#nbBandes = 1
#stop     = q_b
#vitesse  = 2
#bande    = "11111"


/*
On déplace la tête derrière le compteur et on met le premier symbole
*/
(q_0, , ,N,q_f)      //cas n= 0
(q_0,1,1,R,q_1)      //saute le compteur
(q_1,1,1,R,q_1)
(q_1, , ,R,q_2)      //laisse un espace
(q_2, ,0,N,q_b)      //premier symbole généré 0

/*
Hypothèse la tête est sur le dernier symbole
Boucle principale on génère le mot suivant
*/

(q_b, , ,N,q_3)      //redémarrage de la boucle, permet d'insérer
(q_b,0,0,N,q_3)      //une machine auxiliaire entre q_b et q_3
(q_b,1,1,N,q_3)

(q_3,1,1,L,q_3)      //on recherche le zero le plus à gauche
(q_3, , ,L,q_5)      //on trouve que des 1 on passe à la longueur suivante
(q_3,0,1,R,q_4)      //on trouve un 0 on le remplace par 1
(q_4,1,0,R,q_4)      //on remplace tous les 1
(q_4, , ,L,q_b)      //on se place sur la dernière lettre

(q_5, , ,N,q_f)      //fin de la génération
(q_5,1,1,L,q_6)      //on revient au début du compteur
(q_6,1,1,L,q_6)
(q_6, , ,R,q_7)
(q_7,1, ,R,q_8)      //efface un 1 (décompte)
(q_8,1,1,R,q_8)      //saute les chiffres restants du compteur
(q_8, , ,R,q_9)      //saute l'espace séparateur
(q_9,1,0,R,q_9)      //génère le plus petit mot de la longueur suivante
(q_9, ,0,N,q_b)
```

L'analyse syntaxique de la description de la machine devra être faite avant la phase de simulation. Une aide sera fournie à l'utilisateur pour l'aider à corriger ses erreurs. Les machines peuvent être non déterministes dans ce cas, votre logiciel doit proposer une simulation déterministe **et** une simulation probabiliste.

La Simulation

La phase de simulation consiste à lancer l'exécution de la machine. Les simulations devront se faire dans des threads indépendants des threads de l'interface utilisateur pour ne pas la bloquer. La vitesse de simulation et le mode pas à pas devront pouvoir être activés à tout moment par l'utilisateur.

Les machines de Turing. Dans le cas des machines de Turing, la simulation consiste à :
– visualiser en temps réel le contenu des bandes et les mouvements des têtes de lecture/écriture ;
– mettre à jour les compteurs d'espace (nombre de cases visitées sur chacune des bandes) et de temps (nombre de transitions effectuées) ;
– indiquer l'état courant et la transition future ;
– indiquer sur le graphe de la machine l'état courant.

Les machines à registres. Dans le cas des machines à registres, la simulation consiste à :
– visualiser en temps réel le contenu des registres ;
– visualiser dans le texte du programme, l'instruction qui vient d'être exécutée et/ou celle qui va être exécutée ;
– mettre à jour les compteurs d'espace (plus grand nombre utilisé dans les registres) et de temps (nombres d'instructions *inc* ou *dec* effectuées) ;
– indiquer sur le graphe de la machine l'instruction en cours.

L'interface utilisateur

L'interface doit être soignée. Le choix de Java comme langage support du projet vous permet de réaliser une interface extrêmement conviviale. Elle doit entre autre autoriser l'utilisateur à
– mettre au point la machine : mode pas à pas, reprise après un arrêt, ajout/suppression de point d'arrêt, modification de la vitesse...
– modification du contenu des bandes pour les machines de Turing.
– modification du contenu des registres pour les machines à registres.
– les fenêtres de simulation et de visualisation doivent être disjointes pour permettre à l'utilisateur de les cacher et de les redimensionner suivant son goût.
– il est souhaitable, mais non obligatoire que l'utilisateur puisse lancer la simulation et/ou la visualisation de plusieurs machines à la fois.

Donnez libre cours à votre imagination et à vos talents artistiques. Évitez les interfaces style McDo de Windows.

La représentation du graphe d'une machine

Le problème posé pour la représentation du graphe est de définir une bonne représentation. Il n'y a pas de réponse absolue à ce problème. Il y a de nombreux algorithmes pour créer des représentations esthétiques. Un certain nombre de critères objectifs et subjectifs doivent être respectés :

– le graphe doit occuper une surface minimale ;
– le nombre d'arêtes qui se coupent doit être le plus petit possible et dans ce cas la lecture doit rester facile ;
– les étiquettes des arêtes et des sommets doivent être écrites horizontalement ;
– les étiquettes ne doivent pas se chevaucher ;
– le placement des sommets et des arêtes doit coller le plus possible à la logique du graphe ;
– les espacements doivent être les plus réguliers possibles.

On peut ajouter de nombreux autres critères : par exemple, minimiser l'emploi des courbes de Bézier ou des splines voire les interdire etc.

Il n'existe pas d'algorithme parfait pour satisfaire toutes ces contraintes. Vous devez chercher dans la littérature sur ce sujet les algorithmes qui existent, faire le choix de vos critères et construire un algorithme qui génère un placement du graphe le plus proche possible des critères que vous avez retenus.

Reste alors la visualisation du graphe à l'écran. Un graphe comportant plusieurs milliers d'états n'est pas facile à voir sur un écran, il faut alors des techniques de zoom et de déplacement pour pouvoir tirer profit de la visualisation.

Pour vous aider dans cette partie, vous êtes autorisés à utiliser les outils d'Emmanuel Pietriga `http://claribole.net` en apportant les modifications nécessaires pour visualiser vos graphes. Dans ce cas, bien entendu, c'est à vous de générer le codage svg de votre graphe sans passer par Graphviz (mais il n'est pas interdit d'étudier ce programme `http://www.research.att.com/sw/tools/graphviz`.)

Si vous préférez, vous pouvez interfacer votre programme à d'autres visualisateurs tant que c'est votre logiciel qui calcule le placement.

Bon courage !

Références

[1] G. Cornell C. H. Horstmann. *Au cœur de Java 2*, volume I et II. Campus Press, 2003.

[2] D. Wagner M. Kaufmann, editor. *Drawing Graphs*, volume LNCS 2025. Springer, 2001.

[3] J. Shirazi. *Java Performance Tuning*. O'Reilly, 2003.

[4] K. Topley. *Core SWING advanced programming*. Prentice Hall PTR, 2000.

[5] K. Topley. *Core JFC, Java Foundation Classes*. Prentice Hall PTR, 2002.