Arhan Unay
Ege Turkmen

12.03.2023
INM02

COMPTE RENDU DU MINI PROJET INF304

pour le TP7 :

On a tout d'abord écrit les tests fonctionnels en fin de tester que la programme est conforme a sa spécification.Ca veut dire il affiche 'OK' ou 'No' selon le test formée correctement en utilisant les méthodes tester les bornes,partitionner l'ensemble d'entrée en fonction du résultat attendu et de sa structure et à la fin test aléatoire. (en donnant les informations voulu dans l'ordre correspondant de l'exercice.Et après on fait des tests robustesse pour permet de tester que le programme résiste à un environnement d'exécution dégrade (par exemple nom de fichier mangue, valeur de position x manque …).Et a la fin on affiche les résultats obtenu en comparant les résultats attendu selon les tests qu'on a fait dans des terrains et programme différents pour remarquer les erreurs dans l'interprète plus facilement.

Une bref explication des test :

test 1,2,3,4,5 : ATTENTION on a fait ces tests suivant leur spécification correcte(Par exemple x est type in qui est positif, orientation est type char compris entre N,S,O;E).la On a exécuté la curiosity-test avec un test formé correctement. Pour la première on essaye d'obtenir Oui avec les paramètres correct. Pour la deuxième on essaye d'obtenir No en écrivant une exemple d'événement attendu (e) différent que l'obtenu . Pour le troisième test on a entree different valeur de x par rapport a x qu'on va obtenir.Pour la quatrième test on a entree different valeur de y par rapport a y qu'on va obtenir.Dans cinquième test on a fait la même chose pour l'orientation cette fois ci.

Maintenant on fait les tests dans une environnement dégradé(robustesse).Par exemple en donnant le nom fichier qui n'existe pas ou vide , nom du programme manqué…

Test 6:Nom fichier terrain manque(affiche Erreur).
Test 7:Nom fichier terrain incorrect ou échec d'ouverture(affiche Erreur).
Test 8:Nom fichier programme manque(affiche Erreur).
Test 9:Nom fichier terrain incorrect ou échec d'ouverture(affiche Erreur).
Test 10:Nombre max incorrect(négatif ou dépasse les limites)(affiche Erreur).
Test 11: Nombre max manque(affiche Erreur).
Test 12:'e' est une caractère inconnu (hors du N,F,S,O,P)(affiche Erreur).
Test 13:Quand e =N ou F ,x ou y manque.(affiche Erreur).
Test 14:Valeur de x ou y incorrect(négatif ou dépasse les limite)(affiche Erreur).
Test 15:Orientation manque(affiche Erreur).
Test 16 :Orientation inconnu (affiche Erreur).
On ne corrige pas ces erreurs et on passe les tests de l'interprète.

Maintenant pour tester intérieur du fonction on fait des test unitaires et robustesse pour trouver les erreurs(par exemple { bloc de fermeture manque) comme s' il lit correctement le programme et le terrain. puis si il crée l'environnement correctement et si il change l'état de robot correctement :

Test 17 :on test les commande(A , D ,G) du langage dans un terrain correct avec un programme correct(test déroule correctement(affiche OK))
Test 18:on teste les commandes(M , !,{ , } , ? , C) du langage dans un terrain correct avec un programme correct(test déroule correctement(affiche OK))
Test 19 :on teste les commandes(+, - , /) du langage dans un terrain correct avec un programme correct(test déroule correctement(affiche OK))
Test 20 :on teste les situations quand on ouvre le bloc ' {' mais on ne le ferme pas(affiche Erreur).
Test 21 :on teste les situation quand on ferme le bloc ' }' mais on ne l' ouvre pas(affiche Erreur).
Test 22 :on test les situation quand on essaye avec le caractère inconnu (Hors du langage) (affiche Erreur).
Test 23:on essaye maintenant la commande ? avec une pile vide (sans avoir une commande devant lui)
normalement comme la commande ? dépile deux commandes et dans la situation qui y a pas commande dans la pile il faut afficher une erreur mais le programme n'affiche pas .
Test 24:Même chose que test précédent mais avec les commandes !, C ,I
normalement comme commande ? dépile une seule commande et dans la situation qui y a pas commande dans la pile il faut afficher une erreur mais le programme n'affiche pas .

-> maintenant on test les etats N,F,S,O,P

Test 25,26,27,28,29: On teste l'état N,F,S,O,P avec un programme simple.(affiche OK)
Test 30: On test avec des programme plus compliqué(affiche OK)

Il ne reste que des tests de terrain mais comme on a déjà testé dans le tp précédent:On n'écrit pas ici.
Vous pouvez les trouve dans la répertoire terrain_correct

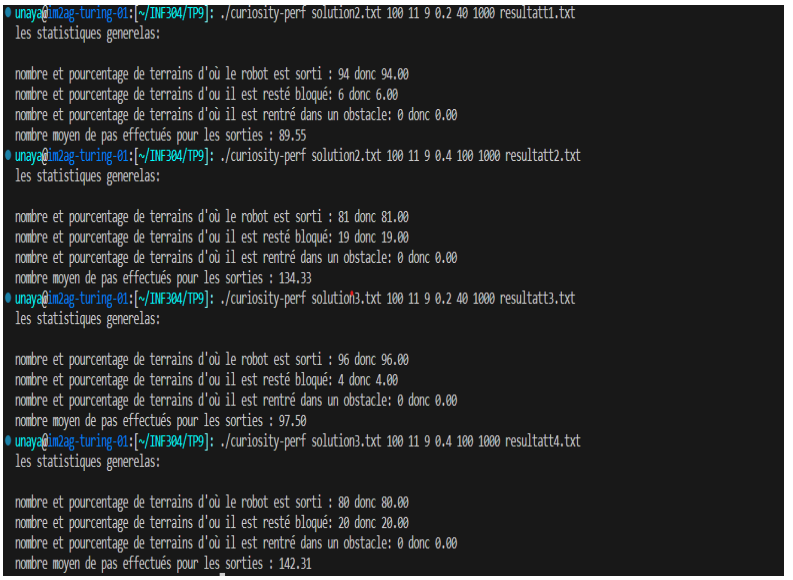
Conclusion:

En effectuant toutes ces actions, notre objectif est de prendre en compte divers types de tests tels que les tests fonctionnels et les tests de robustesse. Ainsi, nous cherchons à disposer d'un ensemble complet de tests afin de garantir que notre programme, dans la mesure du possible, est capable de gérer toutes les situations susceptibles de perturber le bon fonctionnement du programme en question. De plus, la rédaction de tests intentionnellement incorrects nous permet de vérifier si les retours de notre programme sont suffisamment informatifs et s'ils aident le programmeur à repérer l'emplacement du problème. En conclusion, pour cette partie, nous pouvons affirmer que notre programme gère efficacement la plupart des erreurs courantes dans la mesure du possible.Seulement si on entre plus de ligne qu'on a déclaré au dessus de labyrinth on n'obtient pas une erreur ce qui est nécessaire normalement.De plus comme on a indiqué au dessus les commandes C,!,?,I ne dépend pas l'état du pile(vide ou plein).

pour le TP8 :

En partant du principe des exemples que vous avez nous donne, on a pu trouvé 3 different solution qui a un meilleur performant par rapport à avancer indéfini.En essayant trouver ces solutions d'abord on avait essayé avec des labyrinths qui ont gande nombre de dimension et il avait une grande différence de pourcentage de solvant les labyrinth et on a constaté que lorsqu'on diminue leurs dimensions , les taux de probabilité de résoudre une labyrinth(pourcentage de

solution) étaient plus proche.Et dans le cas de densité obstacle on a constaté que plus qu'on augmente les densite d'obstacle plus les taux de probabilité de résoudre une labyrinth diminue.Or plus qu'on diminue les densite d'obstacle plus les taux de probabilité de résoudre une labyrinth augmente donc il y a proportion inverse.
Voici 3 solutions La probabilité de résolution est dans l'ordre du plus petit au plus grand :
les pourcentages ont été déterminés selon cet exemple <<./curiosity-perf solution?.txt 100 25 25 0.2 17 10000 denemeeee.txt>> au lieu de ? on a essayé avec 1,2,3
1: «{ 1 M { D } { A } ? C ! } C !» 62% ->pourcentage de résoudre En partant du principe de l'exemple on fait mesure la case devant lui et avance si la case est libre, tourne à droite s'il y a un obstacle et grâce à ce commande C ! on fait execute 1 M { D } { A } ? indéfiniment et comme ça jusqu'à il sort il fait cette boucle.
2:«{1M {3M {G} {D} ?} {A} ? C ! } C !» 77% ->pourcentage de résoudre Dans cette solution on fait mesure la case devant lui pour controler si il est LIBRE puis et on fait avancer dans le cas libre si non on contrôle son droite si il est LIBRE si oui on fait tourner droite si non on fait tourner à gauche et on met tout ca dans boucle avec la commande C ! .Il va répéter ces étapes jusqu'à ce que le robot sortira.
3:«{1M {7M {D} {G} ?}{A}?C!}C!» %81 ->pourcentage de résoudre Dans cette solution on fait mesure la case devant lui pour controler si il est LIBRE puis et on fait avancer dans le cas libre si non on contrôle son gauche si il est LIBRE si oui on fait tourner gauche si non on fait tourner à droite et on met tous ça dans boucle avec la commande C !.Il va répéter ces étapes jusqu'a le robot sortira.
On a essayé de trouver les meilleur solution en contrôlant difference direction(nombre qui se trouve devant M) dans chaque essai et les 2 plus efficace solutions sont 2eme et 3eme.
Pour les le résultat statistique de l'évaluation de ces programmes-robots sur un ensemble de terrains générés de manière aléatoire on peut regarder les fichiers resultatt1.txt ,resultatt2.txt.,resultatt3.txt,resultatt4.txt. (avec des densités et graine différents .Voici les résultats obtenu dans l'image ci dessous.

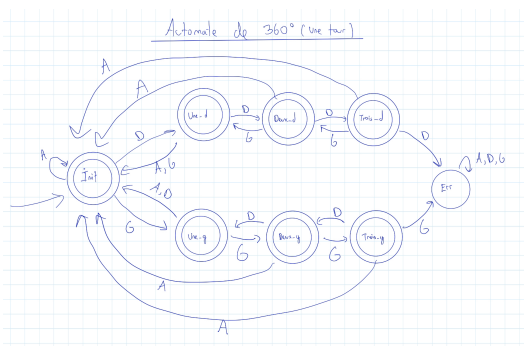


● n=20, L = H = 25, occupation totale de 40 %
Soution1
Min
1-./curiosity-perf solution2.txt 20 25 25 0.4 9931 10000 reponse2_min.txt
->donc %10 pour minimum
2-./curiosity-perf solution2.txt 20 25 25 0.4 99 10000 reponse2_max.txt
->donc %30 pour maximum
Soution2
./curiosity-perf solution3.txt 20 25 25 0.4 9931 10000 reponse3_min.txt
->%5 pour minimum
./curiosity-perf solution3.txt 20 25 25 0.4 19 10000 reponse3_max.txt
->%40 pour maximum
○ n=20, L = H = 9, occupation totale de 70 %
Soution1
./curiosity-perf solution3.txt 20 9 9 0.7 31 10000 reponse3b_max.txt
->%90 pour maximum
./curiosity-perf solution3.txt 20 9 9 0.7 72 210000 reponse3b_min.txt
->%65 pour minimum
Soution2
./curiosity-perf solution2.txt 20 9 9 0.7 232 10000 reponse2b_max.txt
->%80 pour maximum
./curiosity-perf solution2.txt 20 9 9 0.7 253213 10000 reponse2b_min.txt
->%65 pour minimum

Conclusion:

En résumé, la sortie est étroitement liée à la graine (elle a un effet aléatoire sur le résultat), à la dimension des terrains concernés et à la densité des obstacles. Conformément à ce qui a été observé, une fois que l'on dépasse un seuil spécifique pour la dimension et la densité, il n'y a plus de sortie.

pour le TP9 :



NOTE:Pour l'exo 5 du TP9 on a pensé une nouvel propriété dont la description est la suivante:Voici en gauche notre automate a état fini qu'on a créée pour la propriété en question

Si notre robot effectue une rotation complète de 360 degrés autour de lui-même dans une même case,
c'est-à-dire s'il tourne quatre fois au total vers la gauche ou vers la droite dans la même case, nous voulons qu'il signale une erreur. Pour ce faire, nous avons ajouté de nouveaux états à notre fonction de transition et faire des nouvels transitions, ces états sont les suivants(on a une automate avec 8 etats (7 etats accepteur et une état non accepteur)) :

Arhan Unay
Ege Turkmen

12.03.2023
INM02

Init :(0 degree) Lorsque le robot avance dans cet état, il reste dans cet état. Cependant, s'il tourne à droite, il passe à l'état une_d après avoir effectué une rotation totale de droite une fois. De même, s'il tourne à gauche, il passe à l'état une_g après avoir effectué une rotation totale de gauche une fois.

Une_d :-Lorsque le robot avance dans cet état, il revient à l'état init_etat car il se déplace vers un nouveau bloc donc on reste son rotation totale à 0 degré. -S'il tourne à droite, il passe à l'état deux_d car il a effectué une rotation totale de droite deux fois. -S'il tourne à gauche, il revient à l'état init_etat car on a 90 degree à droite + 90 degré gauche donc ça fait 0
degree.

Une_g : -Lorsque le robot avance dans cet état, il revient à l'état init_etat car il se déplace vers un nouveau bloc donc on reset son rotation totale à 0 degree.-S'il tourne à gauche, il passe à l'état deux_g car il a effectué une rotation totale de gauche deux fois. -S'il tourne à droite, il revient à l'état init_etat car on a 90 degree à droite + 90 degré gauche donc ça fait 0 degree.

Deux_d : -Lorsque le robot avance dans cet état, il revient à l'état init_etat car il se déplace vers un nouveau bloc donc on reset son rotation totale à 0 degree.-S'il tourne à droite, il passe à l'état trois_d car il a effectué une rotation totale de droite trois fois.-S'il tourne à gauche, il revient à l'état une_d car il a effectué une rotation totale de gauche une fois
(car on a 180 degré à droite + 90 degré gauche donc ca fait 90 degree droite).

Deux_g : -Lorsque le robot avance dans cet état, il revient à l'état init_etat car il se déplace vers un nouveau bloc donc on reset son rotation totale à 0 degree.-S'il tourne à gauche, il passe à l'état trois_g car il a effectué une rotation totale de gauche trois fois. -S'il tourne à droite, il revient à l'état une_g car il a effectué une rotation totale de droite une fois.(car on a 180 degrés à gauche + 90 degree droite donc ca fait 90 degré gauche).

Trois_d : -Lorsque le robot avance dans cet état, il revient à l'état init_etat car il se déplace vers un nouveau bloc donc on reset son rotation totale à 0 degré. -S'il tourne à droite, il passe à l'état erreur car il a effectué une rotation totale de droite quatre fois (soit 360 degrés). -S'il tourne à gauche, il revient à l'état deux_d car il a effectué une rotation totale de gauche deux fois.(car on a 270 degree a gauche + 90 degree droite donc ça fait 180 degré droite).

Trois_g : Lorsque le robot avance dans cet état, il revient à l'état init_etat car il se déplace vers un nouveau bloc donc on reste son rotation totale à 0 degré. -S'il tourne à gauche, il passe à l'état erreur car il a effectué une rotation totale de gauche quatre fois (soit 360 degrés). -S'il tourne à droite, il revient à l'état deux_g car il a effectué une rotation totale de droite deux fois.(car on a 270 degree a gauche + 90 degree droite donc ça fait 180 degré gauche).
Err : Cet état est atteint lorsqu'il a déjà effectué une rotation complète de 360 degrés. En raison d'une erreur déjà signalée, le robot reste dans cet état pour toutes les actions ultérieures.
Note:Pour toutes les autres commandes (M , ? , ! , la reste) il reste dans la même état.Donc on ne montre pas dans le schéma.
Conclusion:En conclusion, notre observateur fonctionne sans problème et détecte tous les programmes faisant 360 degrés de tour dans une même états.