LOG3430 – Méthodes de test et de validation du logiciel

Laboratoire 4

Tests basés sur les états

Département de génie informatique et de génie logiciel

École Polytechnique de Montréal



Soumis par

Roman Zhornytskiy (1899786)

Hakim Payman (1938609)

Gabriel Tagliabracci (1935775)

Groupe : 02

Soumis à Noureddine Kerzazi

Hiver 2020

4.1. Construire le diagramme d’états de l’interprétateur « Parcer\_FSM.py » et donner à chaque transition un nom, à chaque état également.

Dans le diagramme suivant, nous considérons que :

« Character » représente l’expression régulière suivante : "[A-Za-z|+|-|\d]"

Les états “START” et “END » servent à indiquer, respectivement, le début et la fin de la traversée de l’arbre.

A screenshot of a cell phone

Description automatically generated

Figure 1: Diagramme d'états de "Paser\_FSM.py"

4.2. Construire l’arbre des transitions de l’interprétateur « Parcer\_FSM.py ». Créer une table de transition des changements d'état en explorant le code et proposez de nouvelles transitions possibles.

Dans l’arbre de transitions et la table de transitions suivants, nous considérons que :

« Character » représente l’expression régulière suivante : "[A-Za-z|+|-|\d]"

Les états “START” et “END » servent à indiquer, respectivement, le début et la fin de la traversée de l’arbre.

A close up of text on a white background

Description automatically generated

Figure 2: Arbre de transitions de Parser\_FSM.py

Tableau 1 : Table des transitions de changements d’états de Parser\_FSM.py

|  |  |  |
| --- | --- | --- |
| Entrée | État courant | État suivant |
|  | START | NEW\_GROUP |
| Character | NEW\_GROUP | PREFIX |
| Character | PREFIX | PREFIX |
| “(” | PREFIX | SUBJECT |
| NOT “)” | SUBJECT | SUBJECT |
| “)” | SUBJECT | END\_RULE |
| “&” | END\_RULE | OPERATOR |
| “|” | END\_RULE | OPERATOR |
| “)” | END\_RULE | END\_GROUPE |
| “(” | OPERATOR | NEW\_GROUPE |
| Character | OPERATOR | PREFIX |
| “&” | END\_GROUPE | OPERATOR |
| “|” | END\_GROUPE | OPERATOR |
|  | END\_GROUPE | END |

Nouvelles transitions :

|  |  |  |
| --- | --- | --- |
| Entrée | État courant | État suivant |
| Character | END\_RULE | END\_RULE |
| “|” | OPERATOR | OPERATOR |
| “&” | OPERATOR | OPERATOR |
| “&” | PREFIX | OPERATOR |
| “|” | PREFIX | OPERATOR |
| “)” | NEW\_GROUP | END\_GROUP |

4.3. Identifiez tous les cas de tests avec les conditions à partir de l’arbre trouvé.

D’après l’arbre trouvé, on peut identifier 6 cas de tests différents.

t1 = < {parse = ApplyRules("a = 1") -> parse.run()}, {parse.current\_state == "STATE : PREFIX"} >, condition = [current\_state = S\_NEW\_GROUP] //Il faut commencer avec l’état « NEW\_GROUP »

Ce test produit ces transitions: START -> NEW\_GROUP -> PREFIX -> PREFIX

t2 = < {parse = ApplyRules("a = (132") -> parse.run()}, {parse.current\_state == "STATE : SUBJECT"} >, condition = [current\_state = S\_NEW\_GROUP]

Ce test produit ces transitions: START -> NEW\_GROUP -> PREFIX -> SUBJECT -> SUBJECT

t3 = < {parse = ApplyRules("a = (132) & (") -> parse.run()}, {parse.current\_state == "STATE : NEW\_GROUP"} >, condition = [current\_state = S\_NEW\_GROUP]

Ce test produit ces transitions: START -> NEW\_GROUP -> PREFIX -> SUBJECT -> END\_RULE -> OPERATOR -> NEW\_GROUP

t4 = < {parse = ApplyRules("a = (132) & z") -> parse.run()}, {parse.current\_state == "STATE : PREFIX"} >, condition = [current\_state = S\_NEW\_GROUP]

Ce test produit ces transitions: START -> NEW\_GROUP -> PREFIX -> SUBJECT -> END\_RULE -> OPERATOR -> PREFIX

t5 = < {parse = ApplyRules("a = (132) &") -> parse.run()}, {parse.current\_state == "STATE : OPERATOR"} >, condition = [current\_state = S\_NEW\_GROUP]

Ce test produit ces transitions: START -> NEW\_GROUP -> PREFIX -> SUBJECT -> END\_RULE -> END\_GROUP -> OPERATOR

t6 = < {parse = ApplyRules("a = ( (132) )") -> parse.run()}, {parse.current\_state == "STATE : END\_GROUP"} >, condition = [current\_state = S\_NEW\_GROUP]

Ce test produit ces transitions: START -> NEW\_GROUP -> PREFIX -> SUBJECT -> END\_RULE -> END\_GROUP -> END

4.4. À l’aide de Unittest, écrire une classe de test unitaire pour tester les cas de test identifiés dans la question précédente.

Voir le fichier test\_parser\_FSM.py.

4.5. À l’aide de l’outil Coverage.py, évaluez la couverture de l’interprétateur « Parcer\_FSM.py » selon la couverture des branches et identifiez les branches non couvertes, s’il y en a.

Voici le la couverture que Coverage.py nous donne :

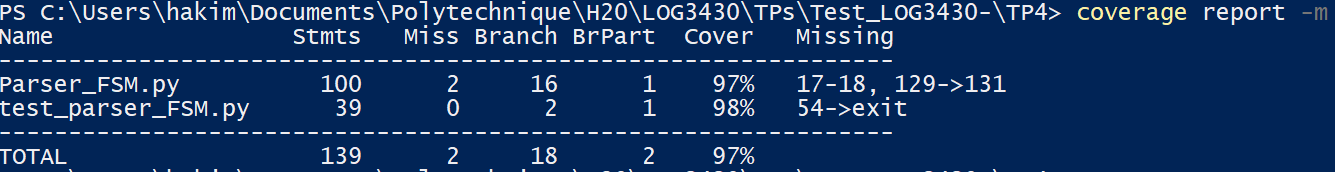


Figure 3: Couverture des branches avec Coverage.py

Nous avons manqué les instructions aux lignes 17-18 de Parser\_FSM.py. Les lignes 17-18 concernent une fonction qui n’est jamais utilisée (*tr\_add\_operator*) dans le programme et qui n’est donc pas couverte par notre jeu de tests.

Ensuite, la seule branche que nous avons manquée dans Parser\_FSM.py est celle couvrant les instructions des lignes 129-131 (voir image ci-dessous).

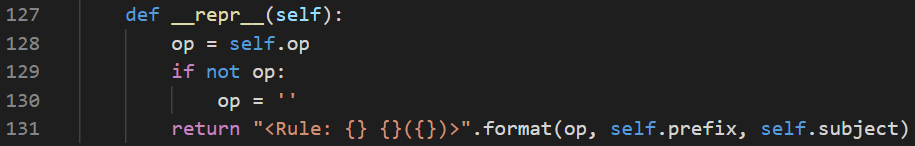


Figure 4: Branche non couverte aux lignes 129-131