

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 Nouredine Kerzazi

Hiver 2020

4.1. Construire le diagramme d'états de l'interpréteur « 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.

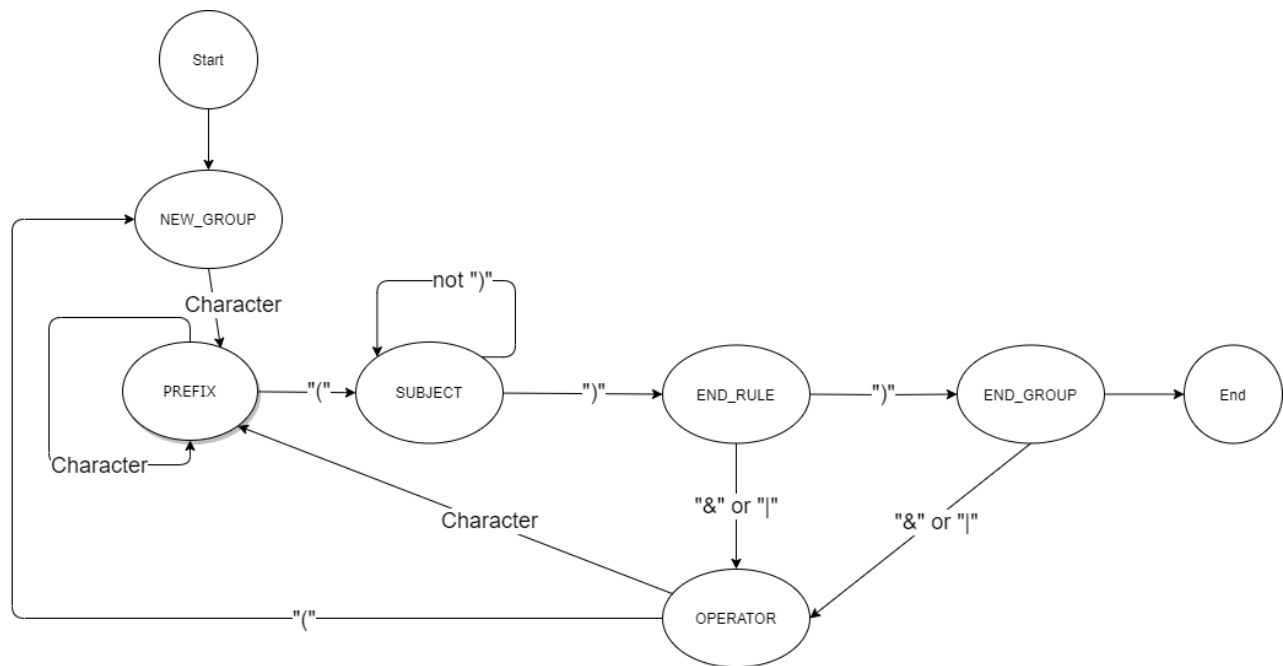


Figure 1: Diagramme d'états de "Paser_FSM.py"

4.2. Construire l'arbre des transitions de l'interpréteur « 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.

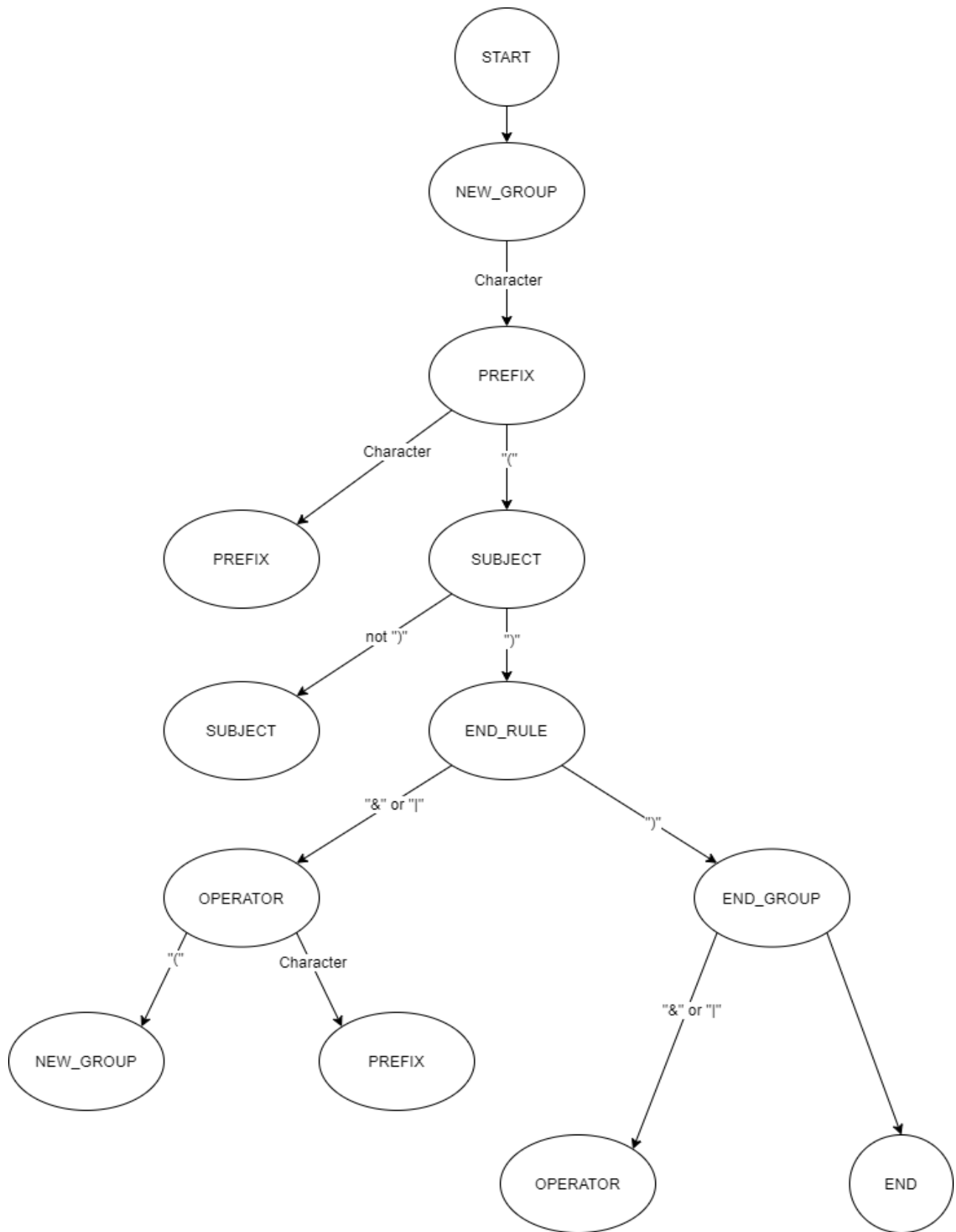


Figure 2: Arbre de transitions de Parser_FSM.py

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]
```

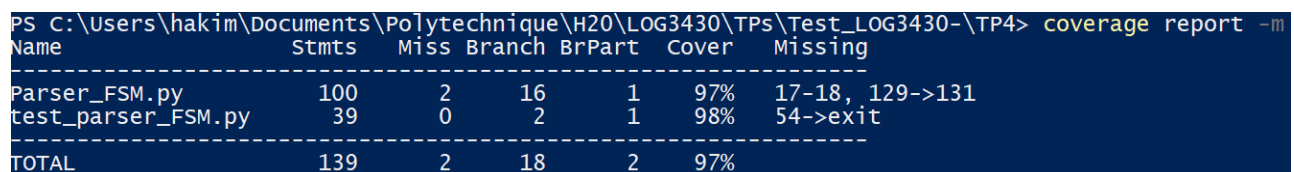
Ce test produit ces transitions: START -> NEW_GROUP -> PREFIX -> SUBJECT -> END_RULE
-> END_GROUP -> END

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'interpréteur « 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 :



Name	Stmts	Miss	Branch	BrPart	Cover	Missing
Parser_FSM.py	100	2	16	1	97%	17-18, 129->131
test_parser_FSM.py	39	0	2	1	98%	54->exit
TOTAL	139	2	18	2	97%	

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).

```
127     def __repr__(self):
128         op = self.op
129         if not op:
130             op = ''
131         return "<Rule: {} {}({})>".format(op, self.prefix, self.subject)
```

Figure 4: Branche non couverte aux lignes 129-131