FSML Python Implementation

Nicolas Beck

GitHub url : https://github.com/nico1510/sle

1) Parsing

- Is done by antlr generated Parser + Lexer (python Code)
- Grammar is identical to specification grammar

```
fsm : state* EOF ;
state : initial? 'state' id {self.addState($initial.text,$id.text)} '{' transition* '}' ;
initial : 'initial' ;
transition : input_ ('/' action )? ( '->' id )? ';'
                                {self.addTransition($input_.text, $action.text, $id.text)} ;
id : NAME ;
input_ : NAME ;
action : NAME ;
NAME : ('a'..'z'|'A'..'Z')+ ;
WS : ( '\t' | ' ' | '\r' | '\n'| '\u000C' )+ { self.skip() } ;
```

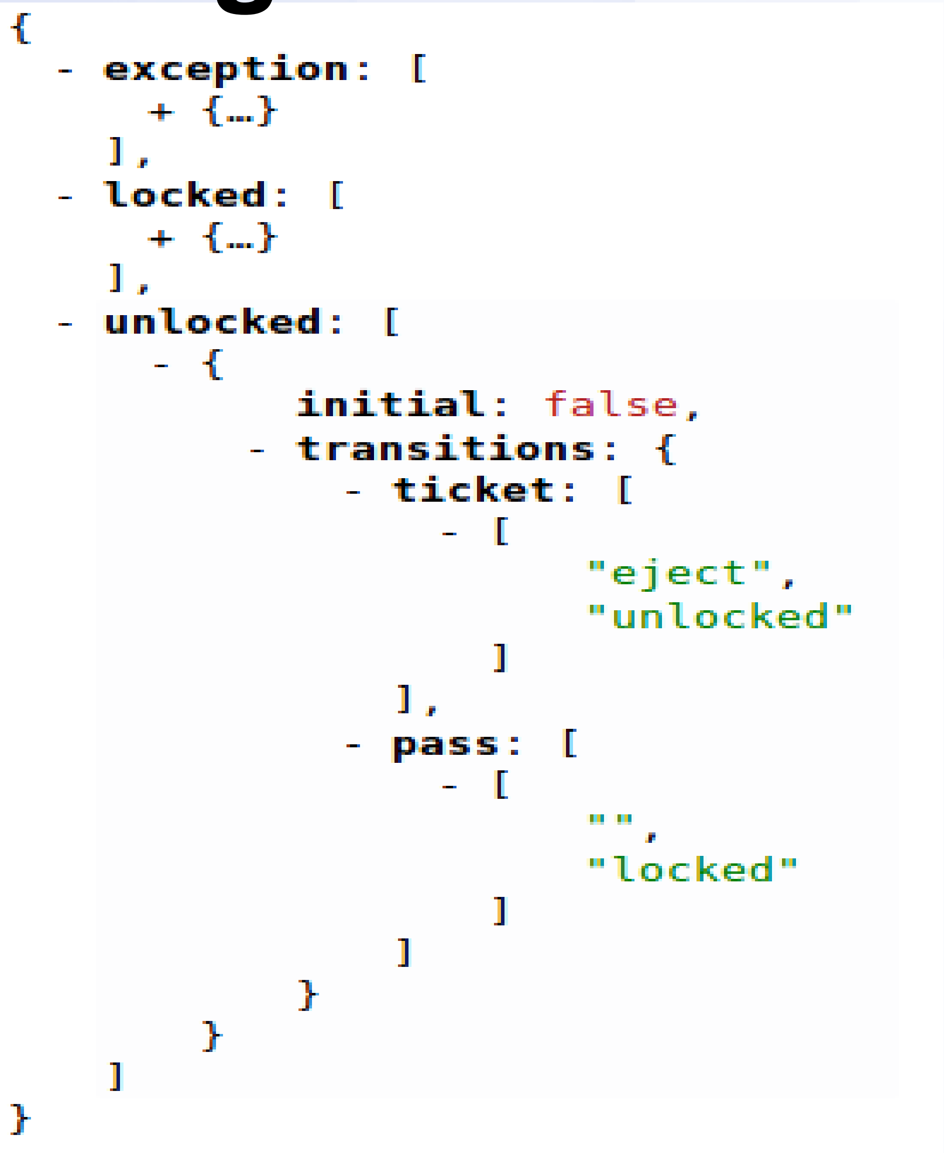
1) Parsing

- Is done by antlr generated Parser + Lexer (python Code)
- Grammar is identical to specification grammar
- semantic actions are included directly in the grammar

```
fsm : state* EOF ;
state : initial? 'state' id {self.addState($initial.text,$id.text)} '{' transition* '}' ;
initial : 'initial' ;
transition : input_ ('/' action )? ( '->' id )? ';'
                                        {self.addTransition($input_.text, $action.text, $id.text)} ;
id : NAME ;
input_ : NAME ;
action : NAME ;
NAME : ('a'..'z'|'A'..'Z')+ ;
WS : ( '\t' | ' ' | '\r' | '\n'| '\u000C' )+ { self.skip() } ;
```

1) Parsing

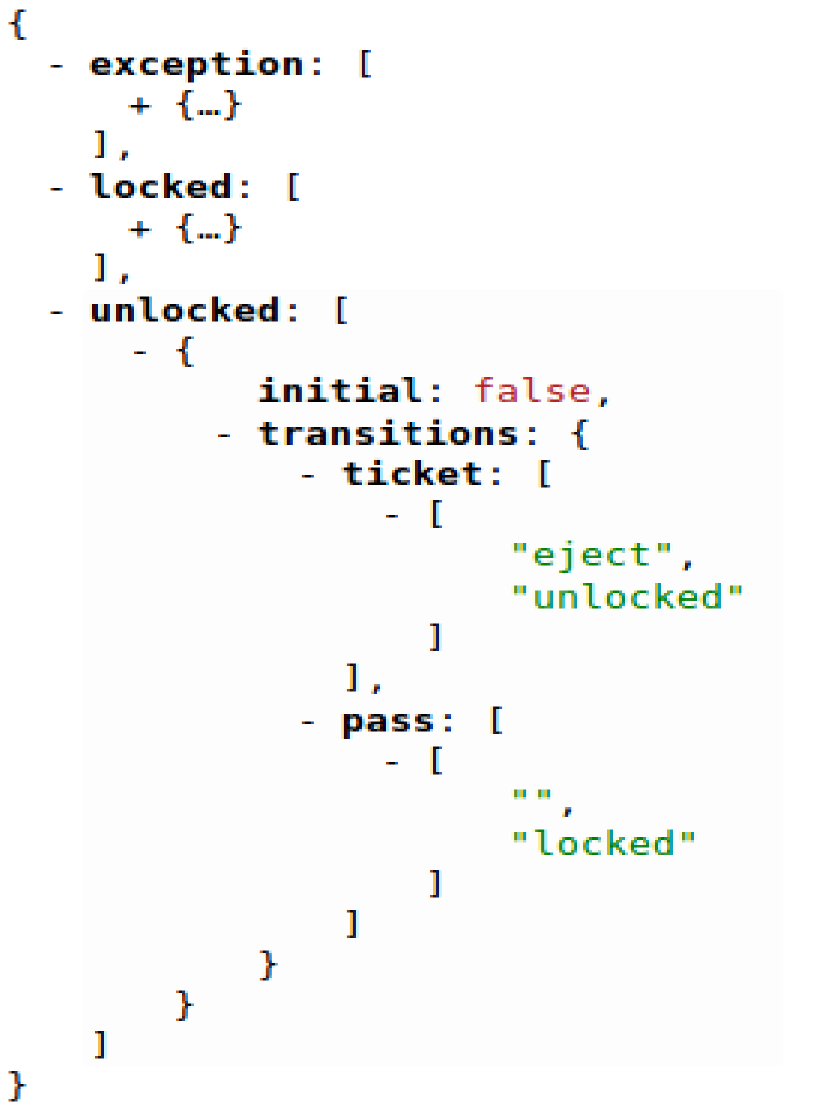
→ Result is a python Dictionary, representing the FSM



```
{
  - exception: [
      + {…}
    ],
  - locked: [
      + {…}
    ],
  - unlocked: [
      - {
            initial: false,
          - transitions: {
              - ticket: [
                  - [
                        "eject",
                        "unlocked"
                    ]
                ],
              - pass: [
                  - [
                        "",
                        "locked"
                    ]
                ]
            }
        }
    ]
}
```

2) Constraints

- Constraints are checked with the help of the dictionary



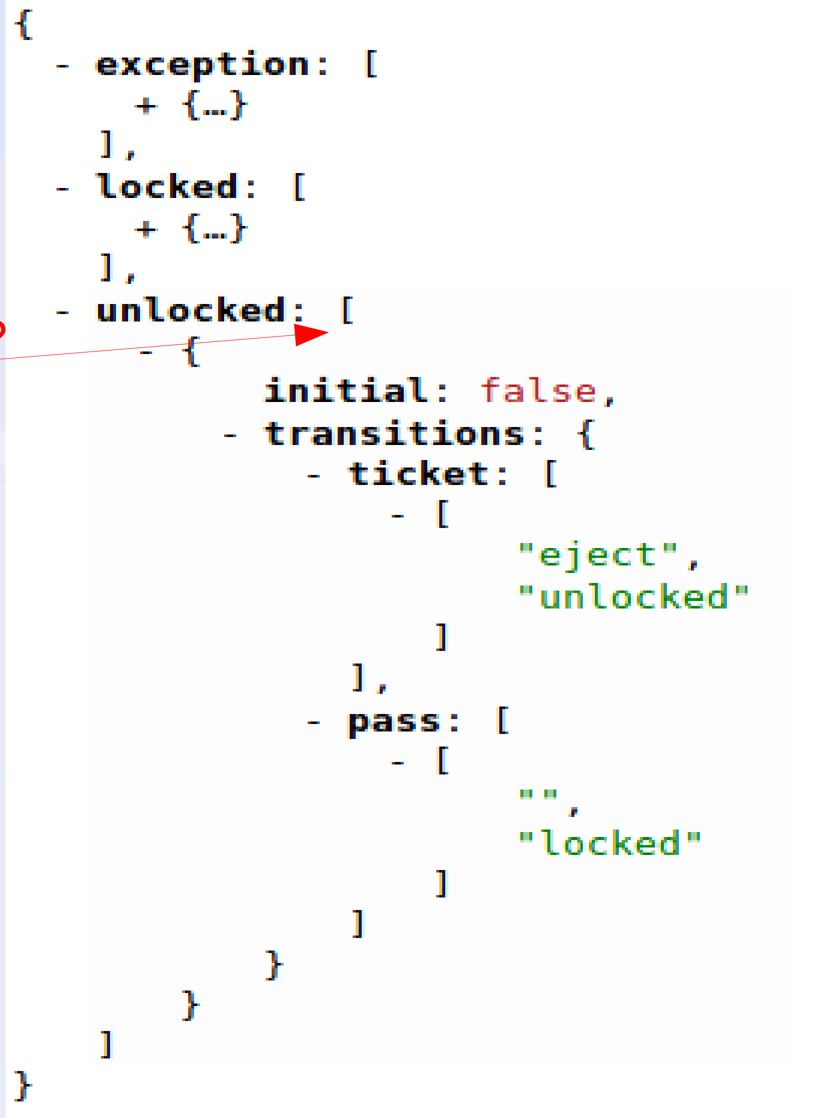
```
{
 - exception: [
    + {…}
   ],
 - locked: [
    + {…}
   ],
 - unlocked: [
    - {
          initial: false,
        - transitions: {
            - ticket: [
                - [
                      "eject",
                      "unlocked"
                  ]
              ],
            - pass: [
                - [
                      "",
                      "locked"
                  ]
              ]
          }
      }
   ]
}
```

2) Constraints

- Constraints are checked with the help of the dictionary

1) distinct Ids

length == 1 ?



```
{
  - exception: [
      + {…}
    ],
  - locked: [
      + {…}
    ],
  - unlocked: [
      - {
            initial: false,
          - transitions: {
              - ticket: [
                  - [
                        "eject",
                        "unlocked"
                    ]
                ],
              - pass: [
                  - [
                        "",
                        "locked"
                    ]
                ]
            }
        }
    ]
}
```

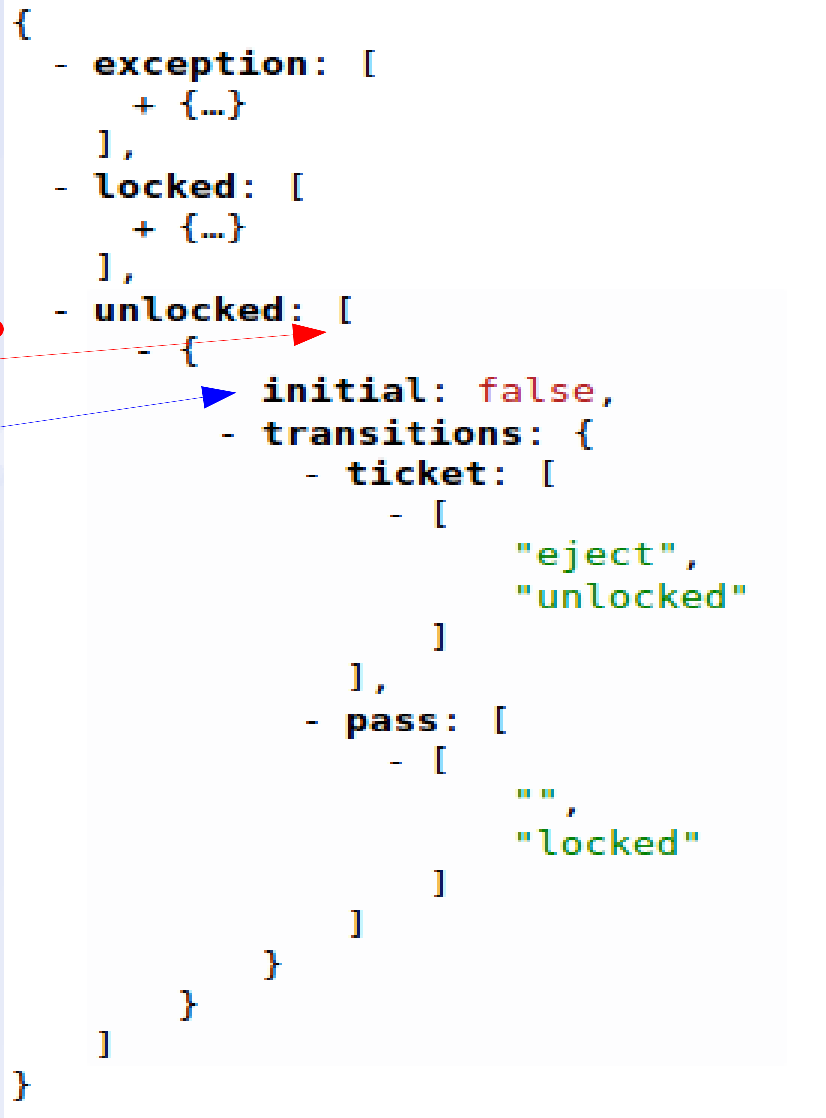
2) Constraints

- Constraints are checked with the help of the dictionary

1) distinct Ids

length == 1 ?

2) single initial



```
{
  - exception: [
      + {…}
    ],
  - locked: [
      + {…}
    ],
  - unlocked: [
      - {
            initial: false,
          - transitions: {
              - ticket: [
                  - [
                        "eject",
                        "unlocked"
                    ]
                ],
              - pass: [
                  - [
                        "",
                        "locked"
                    ]
                ]
            }
        }
    ]
}
```

2) Constraints

- Constraints are checked with the help of the dictionary

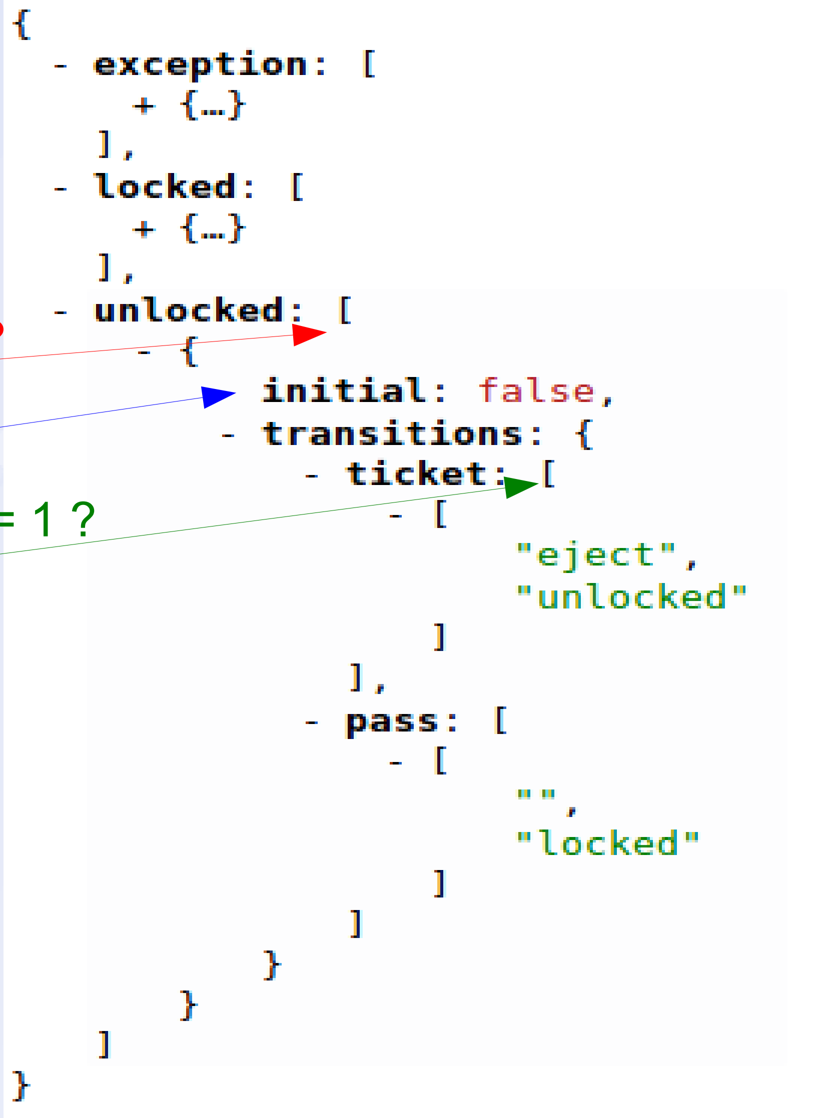
1) distinct Ids

2) single initial

3) deterministic

length == 1 ?

length == 1 ?



```
{
  - exception: [
      + {…}
    ],
  - locked: [
      + {…}
    ],
  - unlocked: [
      - {
            initial: false,
          - transitions: {
              - ticket: [
                  - [
                        "eject",
                        "unlocked"
                    ]
                ],
              - pass: [
                  - [
                        "",
                        "locked"
                    ]
                ]
            }
        }
    ]
}
```

2) Constraints

- Constraints are checked with the help of the dictionary

1) distinct Ids

2) single initial

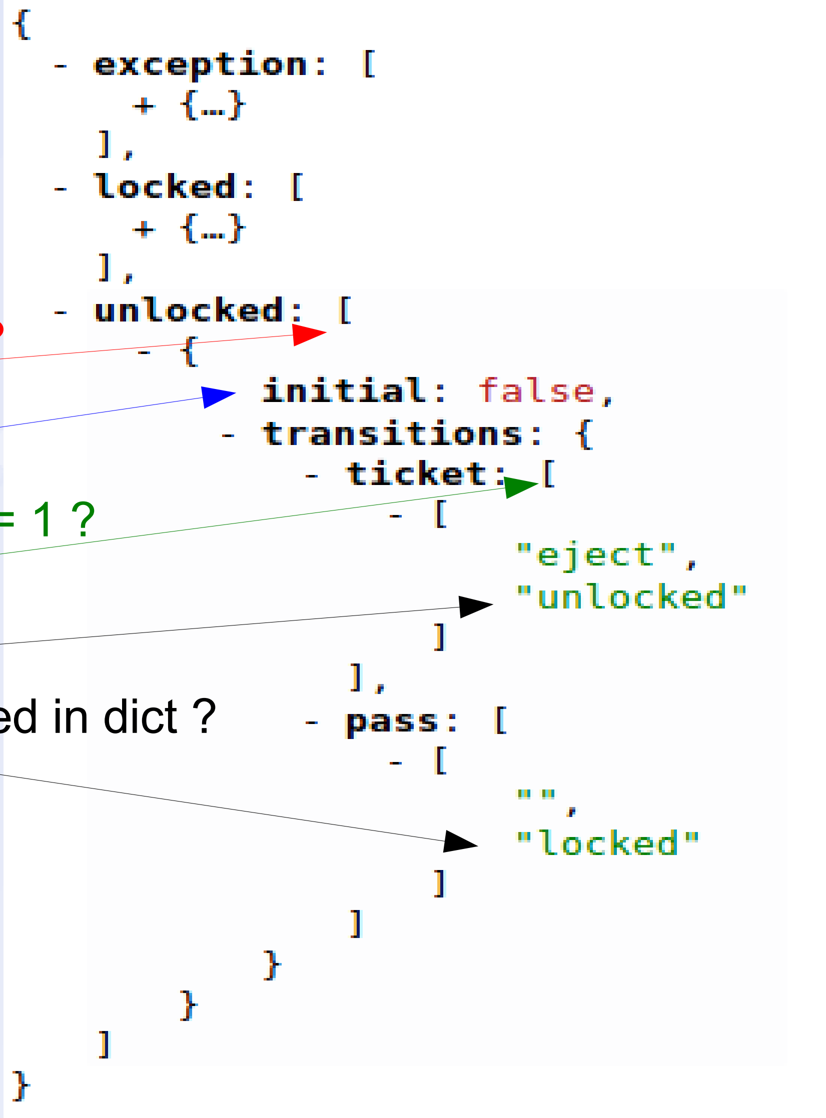
3) deterministic

4) resolvable

length == 1 ?

length == 1 ?

contained in dict ?



```
{
  - exception: [
      + {…}
    ],
  - locked: [
      + {…}
    ],
  - unlocked: [
      - {
          initial: false,
        - transitions: {
            - ticket: [
                - [
                    "eject",
                    "unlocked"
                  ]
              ],
            - pass: [
                - [
                    "",
                    "locked"
                  ]
              ]
          }
        }
    ]
}
```

2) Constraints

- Constraints are checked with the help of the dictionary

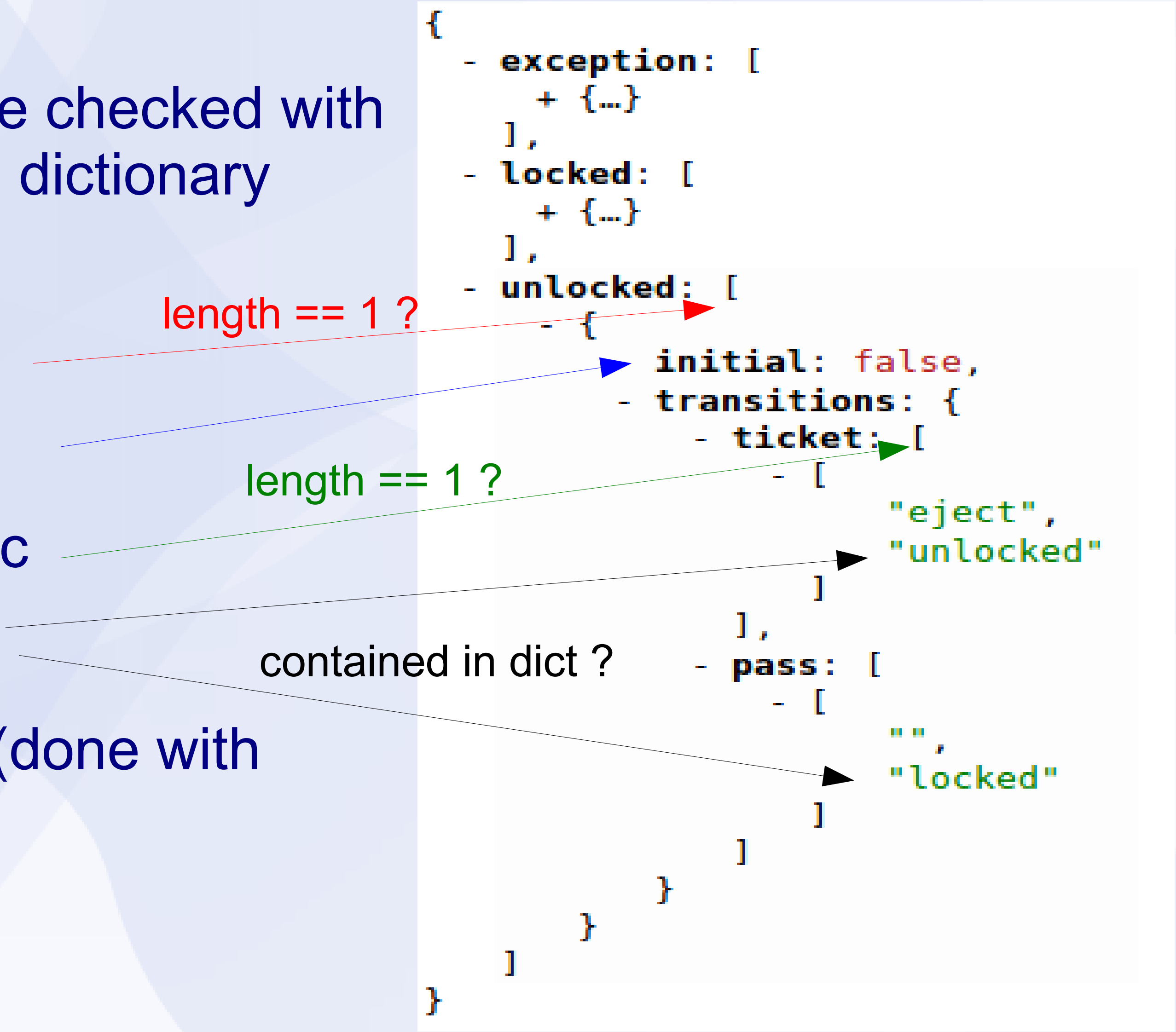
1) distinct Ids

2) single initial

3) deterministic

4) resolvable

5) reachable (done with recursion)



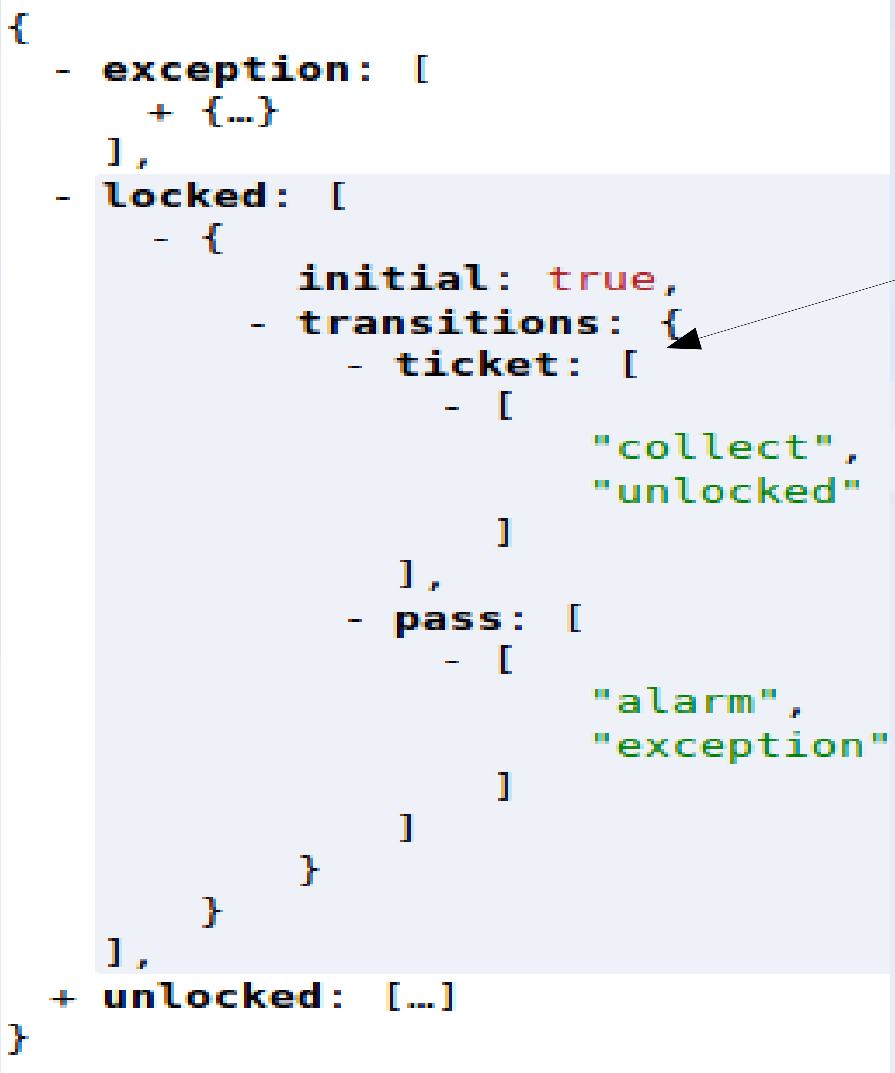
```
{
  - exception: [
      + {…}
    ],
  - locked: [
      + {…}
    ],
  - unlocked: [
      - {
          initial: false,
        - transitions: {
            - ticket: [
                - [
                    "eject",
                    "unlocked"
                  ]
              ],
            - pass: [
                - [
                    "",
                    "locked"
                  ]
              ]
          }
        }
    ]
}
```

length == 1 ?

length == 1 ?

contained in dict ?

3) reference semantics



```
{
  - exception: [
      + {…}
    ],
  - locked: [
      - {
            initial: true,
          - transitions: {
              - ticket: [
                  - [
                        "collect",
                        "unlocked"
                    ]
                ],
              - pass: [
                  - [
                        "alarm",
                        "exception"
                    ]
                ]
            }
        }
    ],
  + unlocked: […]
}
```

Input = [ticket, pass]

Output = [ ]

→ output „collect“

Go to state „unlocked“

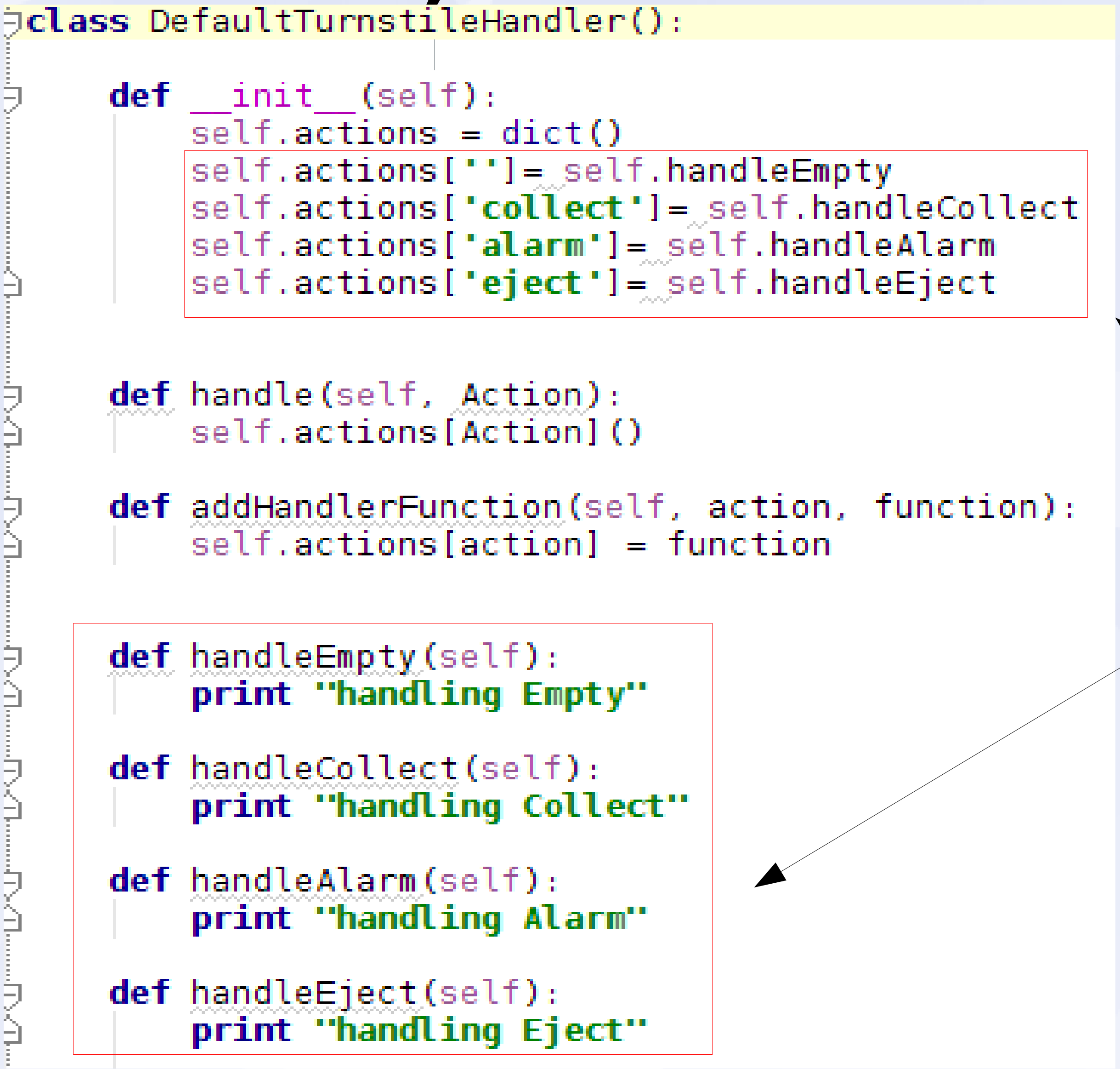
4) Code Generation

- Is done by jinja2 package (template library for python)
- Python Code is generated
- Handler and Stepper Classes
- no Enums like in the Java Version of the Spec

4) Code Generation

```
class DefaultTurnstileHandler():

    def __init__(self):
        self.actions = dict()
        {% for action in actions %}
       self.actions['{{ action }}']= self.handle{{ action|default('Empty', true)|capitalize() }}
        {% endfor %}

    def handle(self, Action):
        self.actions[Action]()

    def addHandlerFunction(self, action, function):
        self.actions[action] = function

    {% for action in actions %}
    def handle{{ action|default('Empty', true)|capitalize() }}(self):
        print "handling {{ action|default('Empty', true)|capitalize() }}"
    {% endfor %}
```

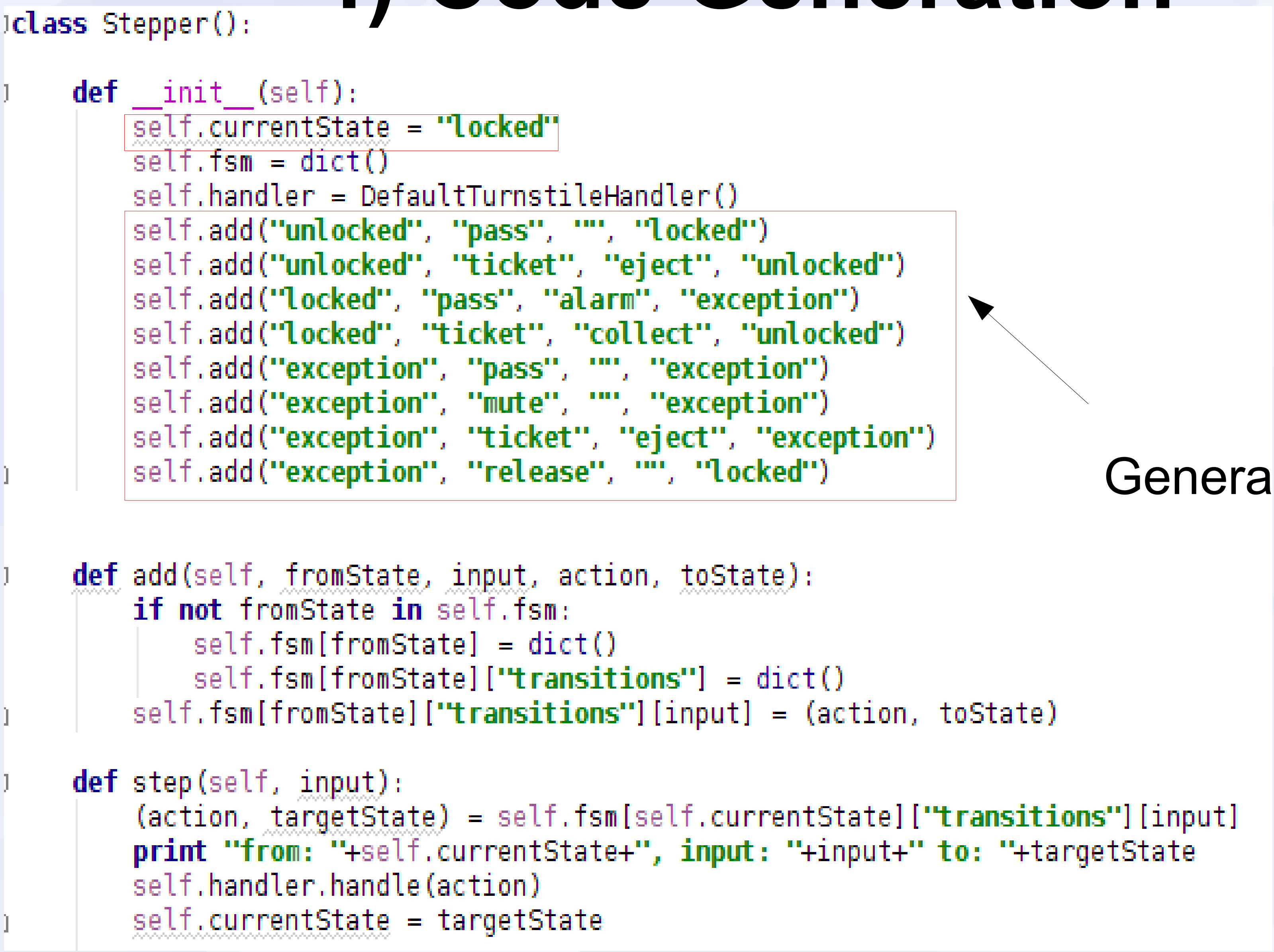
4) Code Generation



```
class DefaultTurnstileHandler():

    def __init__(self):
        self.actions = dict()
        self.actions['']= self.handleEmpty
        self.actions['collect']= self.handleCollect
        self.actions['alarm']= self.handleAlarm
        self.actions['eject']= self.handleEject


    def handle(self, Action):
        self.actions[Action]()

    def addHandlerFunction(self, action, function):
        self.actions[action] = function


    def handleEmpty(self):
        print "handling Empty"

    def handleCollect(self):
        print "handling Collect"

    def handleAlarm(self):
        print "handling Alarm"

    def handleEject(self):
        print "handling Eject"
```

Generated Code

4) Code Generation

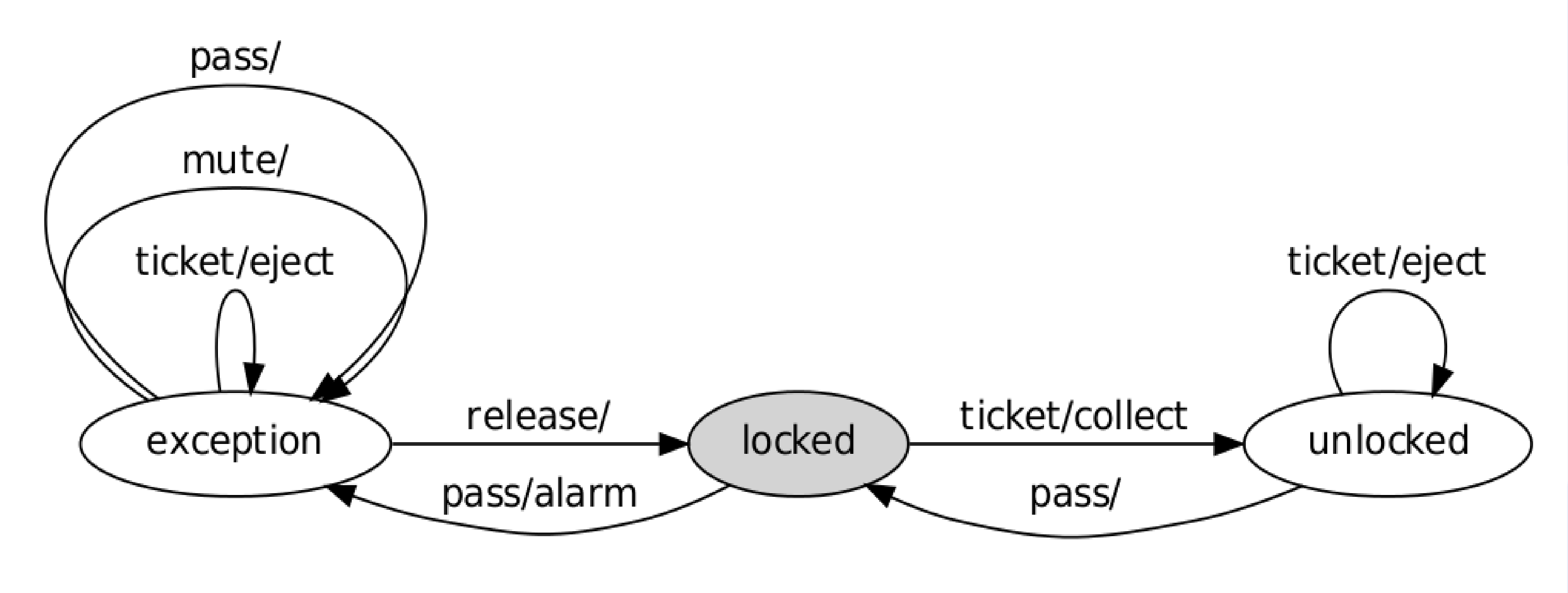


```
class Stepper():

    def __init__(self):
        self.currentState = "locked"
        self.fsm = dict()
        self.handler = DefaultTurnstileHandler()
        self.add("unlocked", "pass", "", "locked")
        self.add("unlocked", "ticket", "eject", "unlocked")
        self.add("locked", "pass", "alarm", "exception")
        self.add("locked", "ticket", "collect", "unlocked")
        self.add("exception", "pass", "", "exception")
        self.add("exception", "mute", "", "exception")
        self.add("exception", "ticket", "eject", "exception")
        self.add("exception", "release", "", "locked")


    def add(self, fromState, input, action, toState):
        if not fromState in self.fsm:
            self.fsm[fromState] = dict()
            self.fsm[fromState]["transitions"] = dict()
        self.fsm[fromState]["transitions"][input] = (action, toState)

    def step(self, input):
        (action, targetState) = self.fsm[self.currentState]["transitions"][input]
        print "from: "+self.currentState+", input: "+input+" to: "+targetState
        self.handler.handle(action)
        self.currentState = targetState
```

Generated Code

5) Visualization

- Is done by pygraphviz package (python API for graphviz)
- Again the fsm dictionary is used :
  - every entry corresponds to a node
  - every transition corresponds to an edge



6) Test Cases

- Testing is done with python's unittest module
- Incorrect *.fsml files are taken from the spec
  - Also test cases from Appendix E (ParserError, infeasible Input, illegal Input) are taken Care of
- Testing Code : https://github.com/nico1510/sle/tree/master/tests