Finite State Machine, state charts

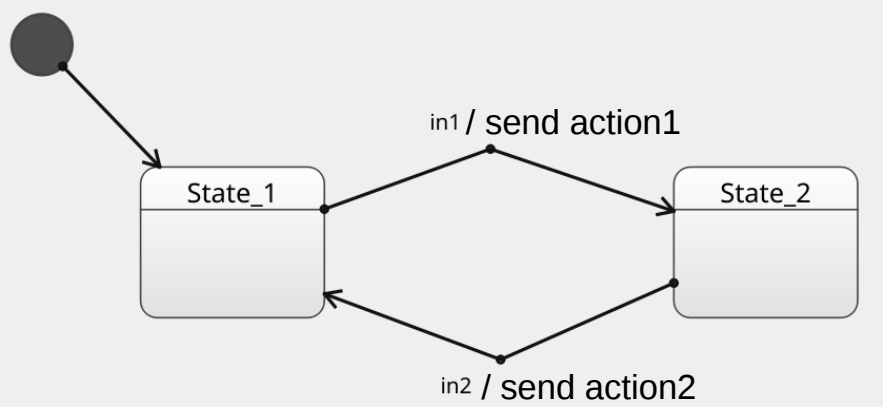
Coding, generating, checking...

First… some examples

- Based on Yakindu by Itemis:
  - https://blogs.itemis.com/en/interactive-3d-visualization-and-simulation-with-state-machines
  - Python code generator:
    https://blogs.itemis.com/en/python-code-generation-with-yakindu-statechart-tools
  - Typescript code generator:
    https://blogs.itemis.com/en/typescript-code-generation-with-yakindu-statechart-tools
  - Programming raspberry pi:
    https://blogs.itemis.com/en/how-to-program-your-raspberry-pi-with-statechart-tools-in-5-minutes
  - Programming arduino:
    https://blogs.itemis.com/en/developing-software-for-arduino-with-yakindu-statechart-tools

Écrire du code implémentant un State chart

Switch case



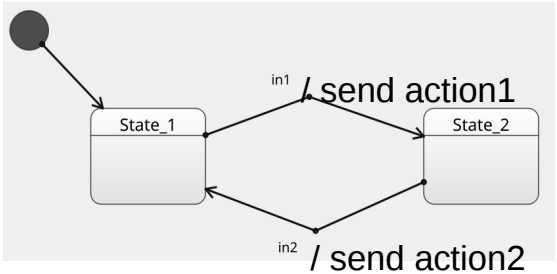

```
int activate(Event newEvent){
   switch(currentState){
      case State_1:
          handleState_1(newEvent);
         break;
      case State_2:
         if (newEvent == in2){
            action2();
            currentState = State_1;
            std::cout << "enter State_1" << std::endl;
         }
         break;
   }
}
```

Peut bien sûr être encapsulé dans une classe !

```
enum State {State_1, State_2};
enum Event {in1, in2};

void action1(){
   std::cout << "action 1 " << std::endl;
}
void action2(){
   std::cout << "action 2 " << std::endl;
}

State currentState = State_1;
```

```
int main(){
  activate(in1);
  activate(in1);
  activate(in2);
}
```

```
void handleState_1(Event newEvent){
   switch(newEvent){
      //transition 1 (t1)
      case in1:
          //exit source of t1 (prefire)
          //actions of t1     (fire)
          //set current state (postfire)
          //enter target of t1
      break;
   }
}
```



State-Event table encoding

	in1	in2
State_1	State_2 (send action1)	X
State_2	X	State_1 (send action2)

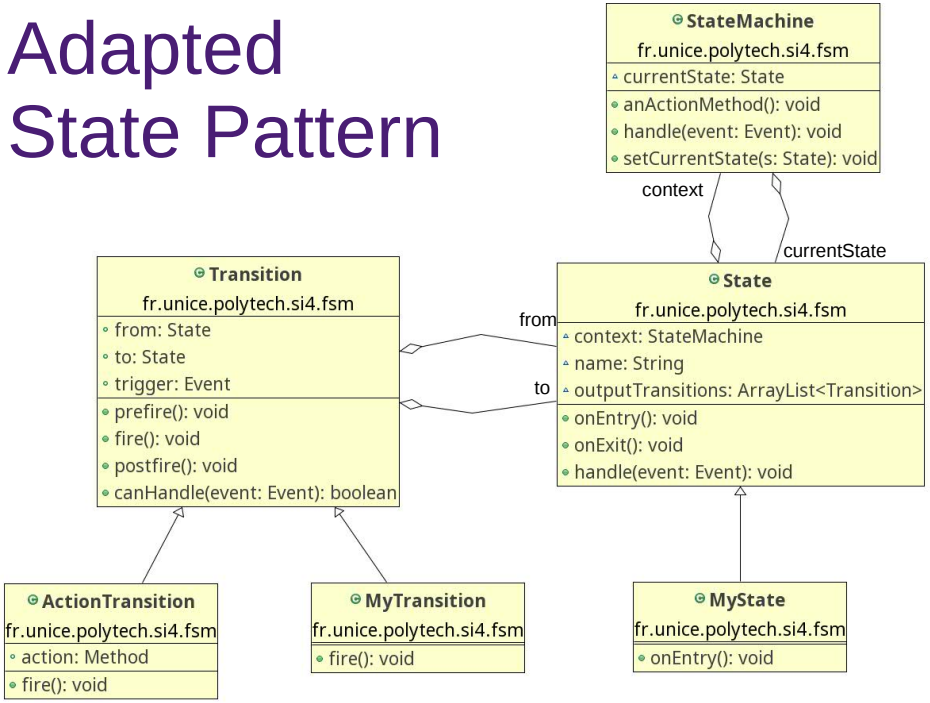
```
void action2(){
    std::cout << "action 2 " << std::endl;
}
void fire2(){
    action2();
    currentState = State_1;
}
void idle(){}
```

```
enum State {State_1, State_2};
enum Event {in1, in2};
const unsigned int nbState = 2;
const unsigned int nbEvent = 2;

State currentState = State_1;

void action1(){
    std::cout << "action 1 " << std::endl;
}
void fire1(){
    action1();
    currentState = State_2;
}
```

Peut bien sûr être encapsulé dans une classe !

```
using FunctionPtr = void (*)();
using FSM = std::array<std::array<FunctionPtr,nbEvent>,nbState>;

FSM fsm =
{{
    {fire1,idle},
    {idle,fire2}
}};

int activate(Event newEvent){
   fsm[currentState][newEvent]();
}
            int main(){
              activate(in1);
              activate(in1);
              activate(in2);
            }
```

Adapted State Pattern



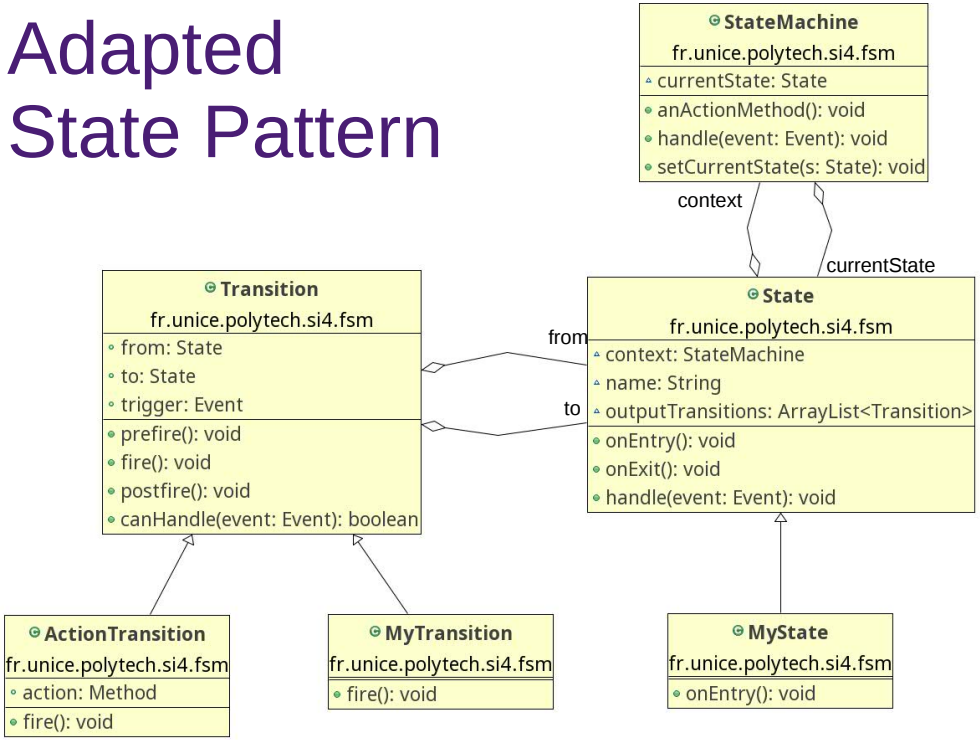

```
public class Transition {
	public State from = null;
	public State to = null;
	public Event trigger;

	public void prefire() {
		this.from.onExit();
	}
	public void fire(){
		System.out.println(
			"  -> fire transition from "+
			from.name +" to "+to.name);
	}
	public void postfire(){
		this.to.onEntry();
	}
	public boolean canHandle(Event event){
		return trigger == event;
	}
}
```

```
public class StateMachine {
	State currentState = null;
	public void handle(Event event){
		currentState.handle(event);
	}
}
```

```
public class State {
	public void handle(Event event){
		System.out.println("receiving "+event);
		for(Transition t : outputTransitions){
			if (t.canHandle(event)){
				t.prefire();
				t.fire();
				t.postfire();
				context.currentState = t.to;
				return;
			}
		}
	}
}
```

```
public class ActionTransition extends Transition{
	public Method action;
	public void fire() {
		super.fire();
		action.invoke(from.context);
	}
}
```

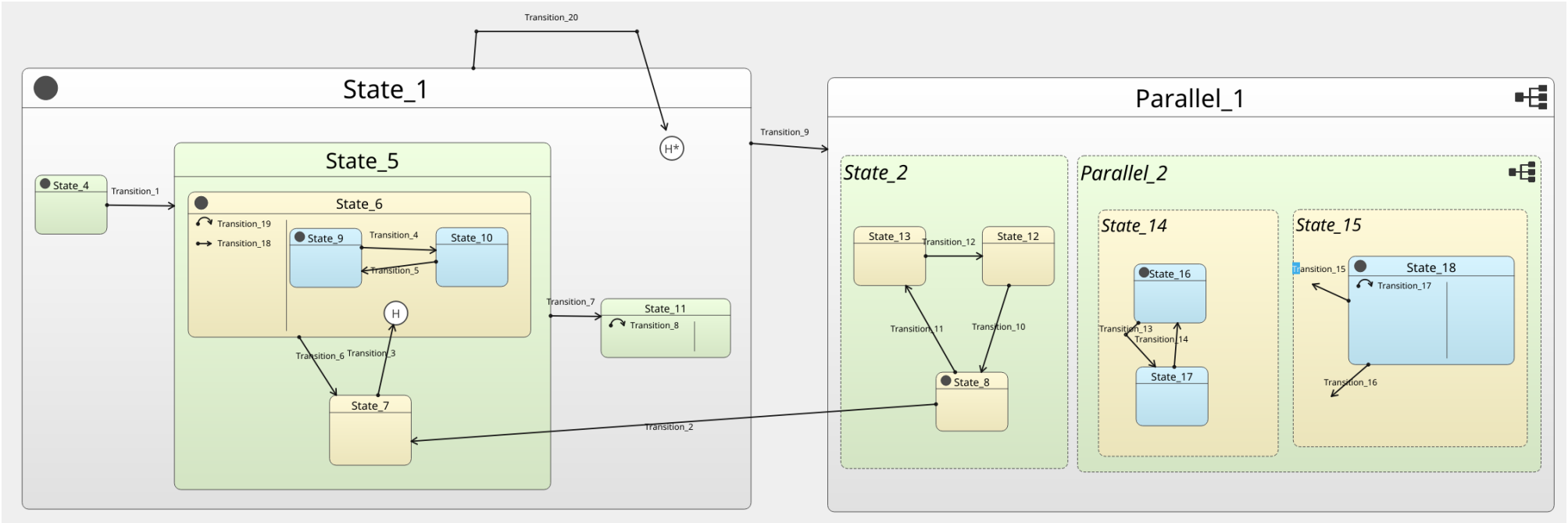
Adapted State Pattern




```
public class Test1 {

public static void main(String[] args) {

  StateMachine fsm = new StateMachine();

  State s1 = new State(fsm,"s1");
  State s2 = new MyState(fsm,"s2");

 Transition t1 = new MyTransition(s1, s2, Event.e1);
 Transition t2 = new Transition(s2, s1, Event.e2);
 ActionTransition t3 = new ActionTransition(s1, s1, Event.e3,
             fsm.getClass().getMethod("anActionMethod"));

 s1.addOutputTransition(t1);
 s2.addOutputTransition(t2);
 s1.addOutputTransition(t3);


 fsm.setCurrentState(s1);
 fsm.handle(Event.e1);
 fsm.handle(Event.e1);
 fsm.handle(Event.e2);
 fsm.handle(Event.e3);
 fsm.handle(Event.e3);
 fsm.handle(Event.e3);
 return ;
 }
}
```

Écrire du code implémentant un State chart

- Supporter toutes les constructions offertes par SCXML peut s'avérer compliqué (vous pouvez aller voir le code générer par Yakindu*compiler* si vous êtes curieux)



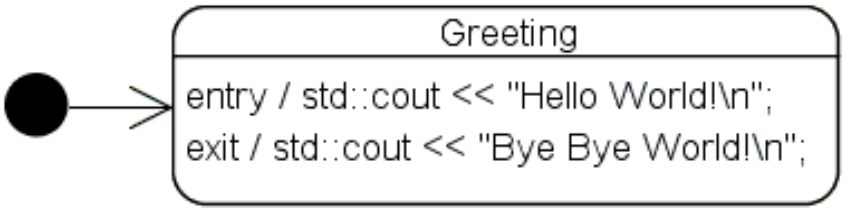
some tools

- Texas instrument FSM generator based on Excel !
  http://www.ti.com/lit/an/slaa402a/slaa402a.pdf
- C++ Boost library: http://www.boost.org/doc/libs/1_65_1/libs/statechart/doc/index.html
- Java library: https://github.com/klangfarbe/UML-Statechart-Framework-for-Java

some examples and tools

- Texas instrument FSM generator based on Excel ! http://www.ti.com/lit/an/slaa402a/slaa402a.pdf
- **C++ Boost library: http://www.boost.org/doc/libs/1_65_1/libs/statechart/doc/index.html**

```
#include <boost/statechart/state_machine.hpp>
#include <boost/statechart/simple_state.hpp>
#include <iostream>
namespace sc = boost::statechart;
class Greeting;
class Machine : sc::state_machine< Machine, Greeting > {};
class Greeting : sc::simple_state< Greeting, Machine >
{public:
 Greeting() { std::cout << "Hello World!\n"; } // entry
 ~Greeting() { std::cout << "Bye Bye World!\n"; } // exit
};
int main()
{
 Machine myMachine;
 myMachine.initiate();
 return 0;
}
```

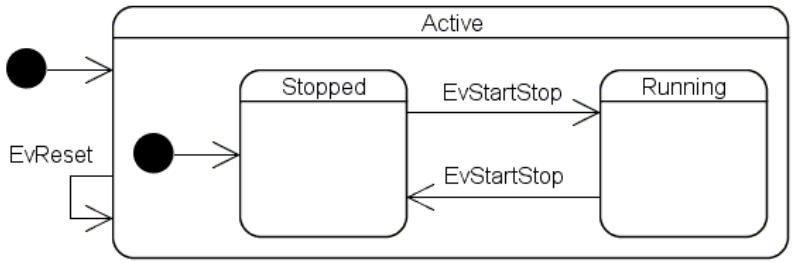


- Java library: https://github.com/klangfarbe/UML-Statechart-Framework-for-Java

some examples and tools

C++ Boost library: http://www.boost.org/doc/libs/1_65_1/libs/statechart/doc/index.html

```
#include <boost/statechart/event.hpp>
#include <boost/statechart/state_machine.hpp>
#include <boost/statechart/simple_state.hpp>
#include <boost/statechart/transition.hpp>
namespace sc = boost::statechart;
class EvStartStop : sc::event< EvStartStop > {};
class EvReset : sc::event< EvReset > {};
class Active;
class StopWatch : sc::state_machine< StopWatch, Active > {};
class Running : sc::simple_state< Running, Active >
{public:
 typedef sc::transition< EvStartStop, Stopped > reactions;
};
class Stopped : sc::simple_state< Stopped, Active >
{public:
 typedef sc::transition< EvStartStop, Running > reactions;
};
class Active : sc::simple_state< Active, StopWatch, Stopped >
{public:
 typedef sc::transition< EvReset, Active > reactions;
};
```




```
int main()
{
 StopWatch myWatch;
 myWatch.initiate();
 myWatch.process_event( EvStartStop() );
 myWatch.process_event( EvStartStop() );
 myWatch.process_event( EvStartStop() );
 myWatch.process_event( EvReset() );
 return 0;
}
```

some examples and tools

- Texas instrument FSM generator based on Excel !
  http://www.ti.com/lit/an/slaa402a/slaa402a.pdf
- C++ Boost library: http://www.boost.org/doc/libs/1_65_1/libs/statechart/doc/index.html
- Java library: https://github.com/klangfarbe/UML-Statechart-Framework-for-Java
  - http://www.java2s.com/Open-Source/Java_Free_Code/Framework_Concurrency/Download_UML_Statechart_Framework_for_Java_Free_Java_Code.htm

```
Statechart chart = new Statechart("t1", 10, false);
State s1 = new State("a", chart, new TestAction("a", "A"), null, new TestAction("a", "D"));
State s2 = new State("b", chart, new TestAction("b", "A"), null, new TestAction("b", "D"));
State p1 = new PseudoState("start", chart, PseudoState.pseudostate_start);
State p2 = new FinalState("end", chart);
p1.setEntryAction(new TestAction("start", "A"));
p1.setExitAction(new TestAction("start", "D"));
p2.setEntryAction(new TestAction("end", "A"));
p2.setExitAction(new TestAction("end", "D"));
new Transition(p1, s1);
new Transition(s1, s2);
new Transition(s2, p2);
```

some examples and tools

- Texas instrument FSM generator based on Excel !
  http://www.ti.com/lit/an/slaa402a/slaa402a.pdf
- C++ Boost library: http://www.boost.org/doc/libs/1_65_1/libs/statechart/doc/index.html
- Java library: https://github.com/klangfarbe/UML-Statechart-Framework-for-Java
  - http://www.java2s.com/Open-Source/Java_Free_Code/Framework_Concurrency/Download_UML_Statechart_Framework_for_Java_Free_Java_Code.htm

Many different libraries on the web but take care their correctness and completeness

Générer du code implémentant un State chart

- En quelques étapes simples:
  1) Récupérer l'AST (Abstract Syntax Tree) correspondant au state chart
  2) Si besoin parcourir l'AST et construire des collections adaptées
  3) parcourir l'AST et *PrettyPrinter* le code attendu

Générer du code implémentant un State chart

- En quelques étapes simples:

  1) Récupérer l'AST (Abstract Syntax Tree) correspondant au state chart

  2) Si besoin parcourir l'AST et construire des collections adaptées

  3) parcourir l'AST et *PrettyPrinter* le code attendu

Si vous avez un fichier en XML il existe tout un tas de parsers génériques !! Il ne faut pas réécrire le votre en lexYacc !!

Générer du code implémentant un State chart

- En quelques étapes simples:

1) Récupérer l'AST (Abstract Syntax Tree) correspondant au state chart
   - Visualiser l'AST: http://countwordsfree.com/xmlviewer or https://codebeautify.org/xmlviewer
   - Requêter l'AST:
     - Xpath: http://xmlgrid.net/xpath.html
     - Python et untangle: http://untangle.readthedocs.io/en/latest/#
     - Java et JDOM: https://fr.wikipedia.org/wiki/JDOM
     - C++ xerces: https://xerces.apache.org/xerces-c/index.html
     - Et plein d'autres…
2) Si besoin parcourir l'AST et construire des collections adaptées
3) parcourir l'AST et PrettyPrinter le code attendu

Générer du code

- Python et untangle:

```
#!/usr/bin/env python
import untangle
ast = untangle.parse(open("watchController.xml"))
```

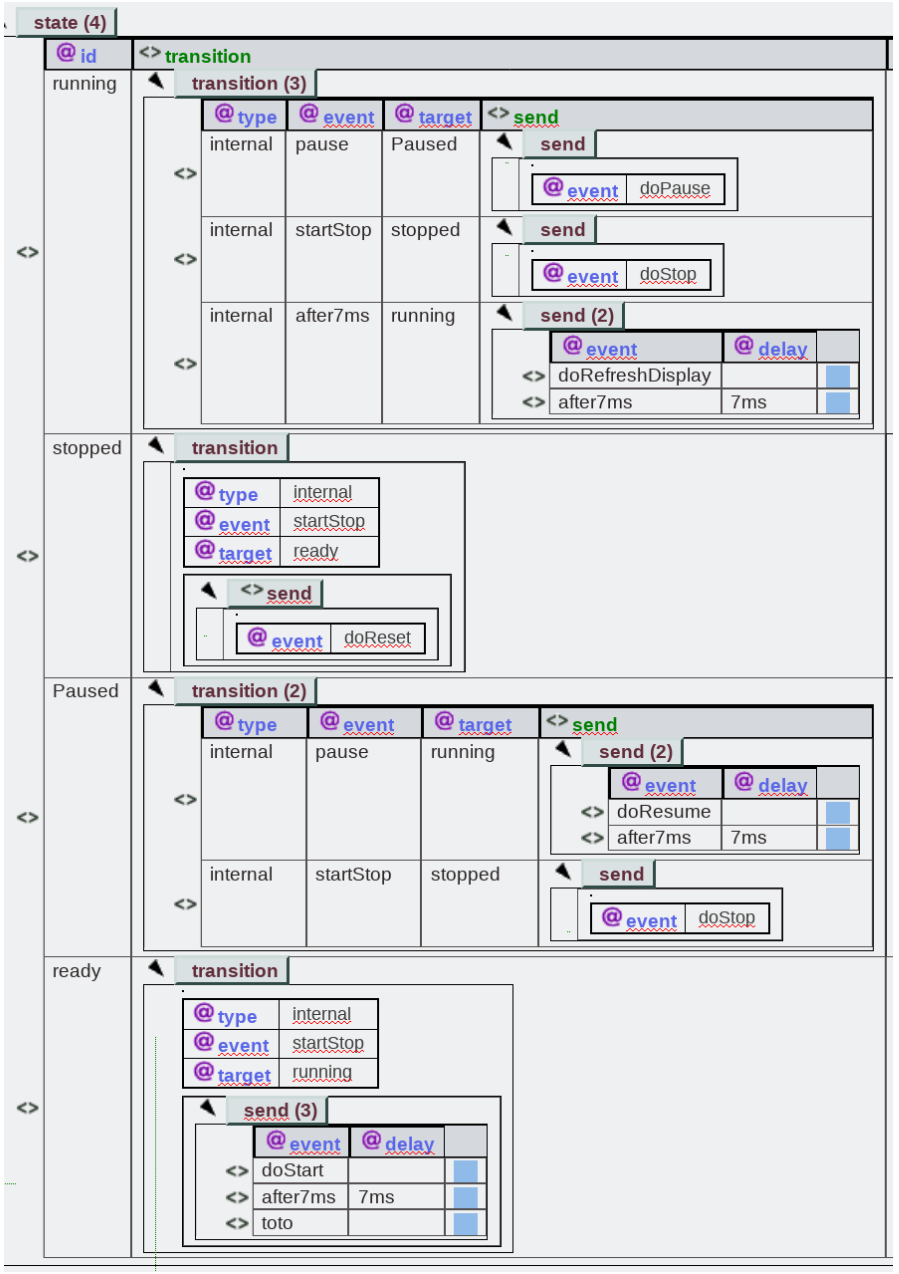
```
<scxml xmlns="http://www.w3.org/2005/07/scxml" version="1.0" binding="early" initial="ready">

    <state id="running">
        <transition type="internal" event="pause" target="Paused">
           <send event="doPause"/>
        </transition>
        <transition type="internal" event="startStop" target="stopped">
            <send event="doStop"/>
        </transition>
        <transition type="internal" event="after7ms" target="running">
            <send event="doRefreshDisplay"/>
            <send event="after7ms" delay="7ms"/>
        </transition>
        <onentry>
            <log expr="controller: is running"/>
        </onentry>
    </state>
    <state id="stopped">
        <transition type="internal" event="startStop" target="ready">
            <send event="doReset"/>
        </transition>
    </state>
    <state id="Paused">
        <transition type="internal" event="pause" target="running">
            <send event="doResume"/>
            <send event="after7ms" delay="7ms"/>
        </transition>
        <transition type="internal" event="startStop" target="stopped">
            <send event="doStop"/>
        </transition>
    </state>
    <state id="ready">
        <transition type="internal" event="startStop" target="running">
            <send event="doStart"/>
            <send event="after7ms" delay="7ms">
            </send>
            <send event="toto"/>
        </transition>
        <onentry>
            <log expr="'I am ready now'" label="ready_onEntry"/>
        </onentry>
    </state>
</scxml>
```

Générer du code

- Python et untangle:

```
#!/usr/bin/env python
import untangle
ast = untangle.parse(open("watchController.xml"))
```

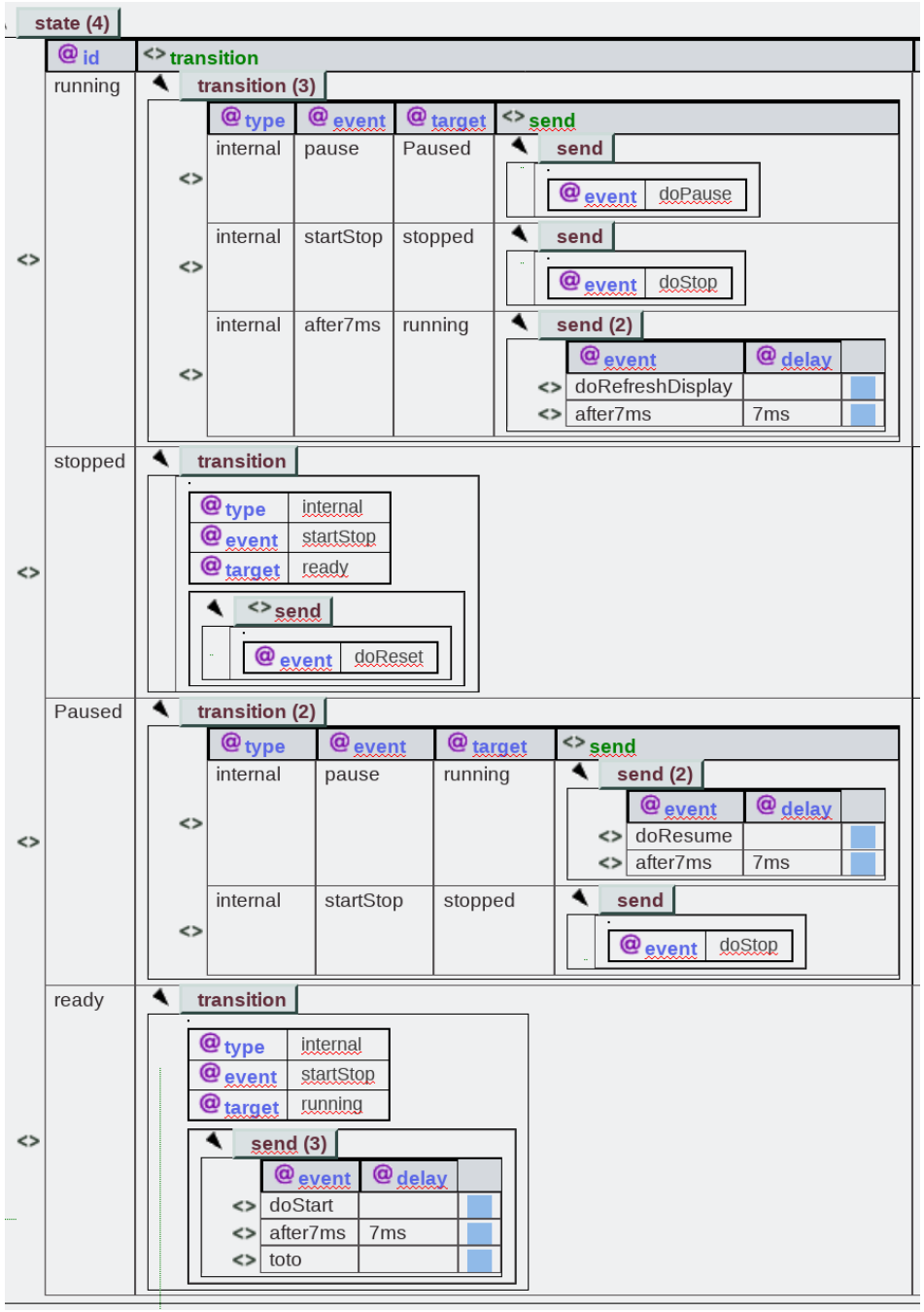
```
<scxml xmlns="http://www.w3.org/2005/07/scxml" version="1.0" binding="early" initial="ready">

    <state id="running">
        <transition type="internal" event="pause" target="Paused">
           <send event="doPause"/>
        </transition>
        <transition type="internal" event="startStop" target="stopped">
            <send event="doStop"/>
        </transition>
        <transition type="internal" event="after7ms" target="running">
            <send event="doRefreshDisplay"/>
            <send event="after7ms" delay="7ms"/>
        </transition>
        <onentry>
            <log expr="controller: is running"/>
        </onentry>
    </state>
    <state id="stopped">
        <transition type="internal" event="startStop" target="ready">
            <send event="doReset"/>
        </transition>
    </state>
    <state id="Paused">
        <transition type="internal" event="pause" target="running">
            <send event="doResume"/>
            <send event="after7ms" delay="7ms"/>
        </transition>
        <transition type="internal" event="startStop" target="stopped">
            <send event="doStop"/>
        </transition>
    </state>
    <state id="ready">
        <transition type="internal" event="startStop" target="running">
            <send event="doStart"/>
            <send event="after7ms" delay="7ms">
            </send>
            <send event="toto"/>
        </transition>
        <onentry>
            <log expr="'I am ready now'" label="ready_onEntry"/>
        </onentry>
    </state>
</scxml>
```



Générer du code

- Python et untangle:

```
#!/usr/bin/env python
import untangle

ast = untangle.parse(open("watchController.xml"))

for s in ast.scxml.state:
    print "State %s" %(s['id'])
    for t in s.transition:
        print '    out transition to %s when %s'
              %(t['target'],t['event'])
        for se in t.send:
            print '        send event %s' %se['event']
```



Générer du code

- Python et pyscxml:
  - Pyscxml (permet d'interpréter le state chart)

```
from scxml.pyscxml import StateMachine

sm = StateMachine("fsm1.scxml")
sm.start_threaded()
sm.send("s2")
sm.send("f")
```

```
theFSM = new StopWatchStatemachine();
TimerService timer = new TimerService();
theFSM.setTimer(timer);
theFSM.init();
theFSM.enter();
theFSM.getSCInterface().raiseLeftButton();
```

Générer du code

- Python et pyscxml:
  - Pyscxml (permet d'interpréter le state chart mais aussi d'accéder à la structure parsée du XML)

```
from scxml.pyscxml import StateMachine
sm = StateMachine("stopwatchController.scxml")
```

```
State currentState = ready;
```

```
print '    State currentState = '+sm.doc.rootState.initial[0]
```

Générer du code

- Python et pyscxml:
  - Pyscxml (permet d'interpréter le state chart mais aussi d'accéder à la structure parsée du XML)

```
from scxml.pyscxml import StateMachine
sm = StateMachine("stopwatchController.scxml")
allStates = sm.doc.rootState.state
lastState = allStates.pop()

print '//State definitions'
print 'enum State={'
for s in allStates:
    print '            '+s.id+','
print '            '+lastState.id
print '            };'
print '    State currentState = '+sm.doc.rootState.initial[0]+';'
```

```
//State definitions
enum State={
            Paused,
            ready,
            running,
            stopped
            };
State currentState = ready;
```

Générer du code

- Python et pyscxml:
  - Pyscxml (permet d'interpréter le state chart mais aussi d'accéder à la structure parsée du XML)

```
from scxml.pyscxml import StateMachine
sm = StateMachine("stopwatchController.scxml")
allStates = Set(getStates(sm.doc.rootState));
lastState = allStates.pop()

print '//State definitions'
print 'enum State={'
for s in allStates:
    print '            '+s.id+','
print '            '+lastState.id
print '            };'
print '    State currentState = '+sm.doc.rootState.initial[0]+';'
```

```
def getStates(s):
    res = Set()
    for subs in s.state:
        res.add(subs)
        res.union(getStates(subs))
    return res
```

Générer du code

- Python et pyscxml:
  - Pyscxml (permet d'interpréter le state chart mais aussi d'accéder à la structure parsée du XML)

```
from scxml.pyscxml import StateMachine
sm = StateMachine("stopwatchController.scxml")
allStates = Set(getStates(sm.doc.rootState));
lastState = allStates.pop()

print '//State definitions'
print 'enum State={'
for s in allStates:
    print '            '+s.id+','
print '            '+lastState.id
print '            };'
print '    State currentState = '+sm.doc.rootState.initial[0]+';'
```

```
def getStates(s):
    res = Set()
    for subs in s.state:
        res.add(subs)
        res.union(getStates(subs))
    return res
```

Il est souvent préférable d'écrire le code attendu une première fois manuellement avant de se lancer dans la génération de code

N'hésitez pas à lever des exceptions si le state chart contient des éléments non supportés par votre générateur

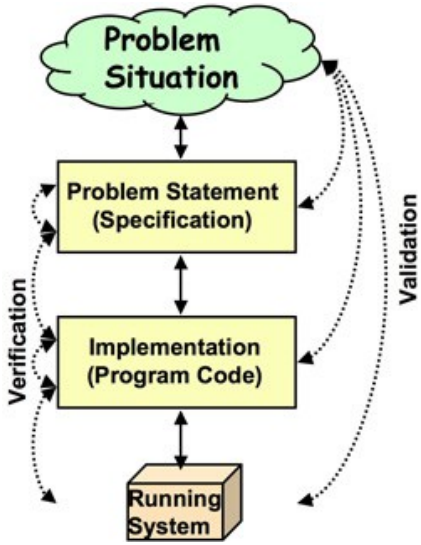
Finite State Machine, state charts and implementations

V&V

V&V ?

- Disclaimer:
  - on ne verra ici qu'une introduction aux notions et problèmes de V&V. Beaucoup de raccourcis sont fait mais cela devrait être suffisant pour vous donner l'intuition derrière ces notions et vous permettre de les approfondir par vous même si besoin.

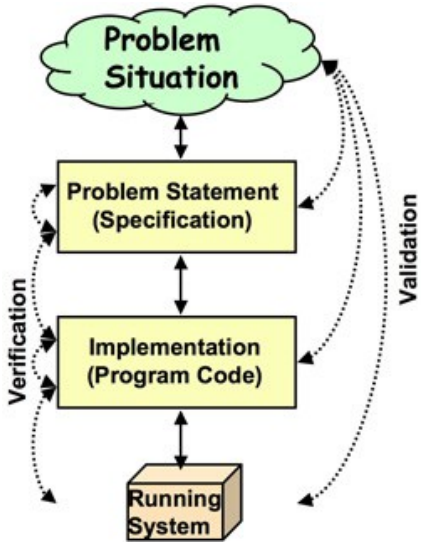
V&V ?



Picture taken from Steve EasterBrook (http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/)

- Verification and Validation
  - Verification: Construisons-nous le système correctement ?
    - Est-ce que le système est implémenté de manière correcte ? (sans erreur, avec les bonnes performances, sans fuites mémoires, rafinement correct, etc)
  - Validation:

V&V ?



Picture taken from Steve EasterBrook (http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/)

- Verification and Validation
  - Verification: Construisons-nous le système correctement ?
    - Est-ce que le système est implémenté de manière correcte ? (sans erreur, avec les bonnes performances, sans fuites mémoires, rafinement correct, etc)
  - Validation: Construisons-nous le bon système ?
    - Est-ce que le système que nous développons est celui que le client voulait / qui répond au problème initial ?

V&V ?

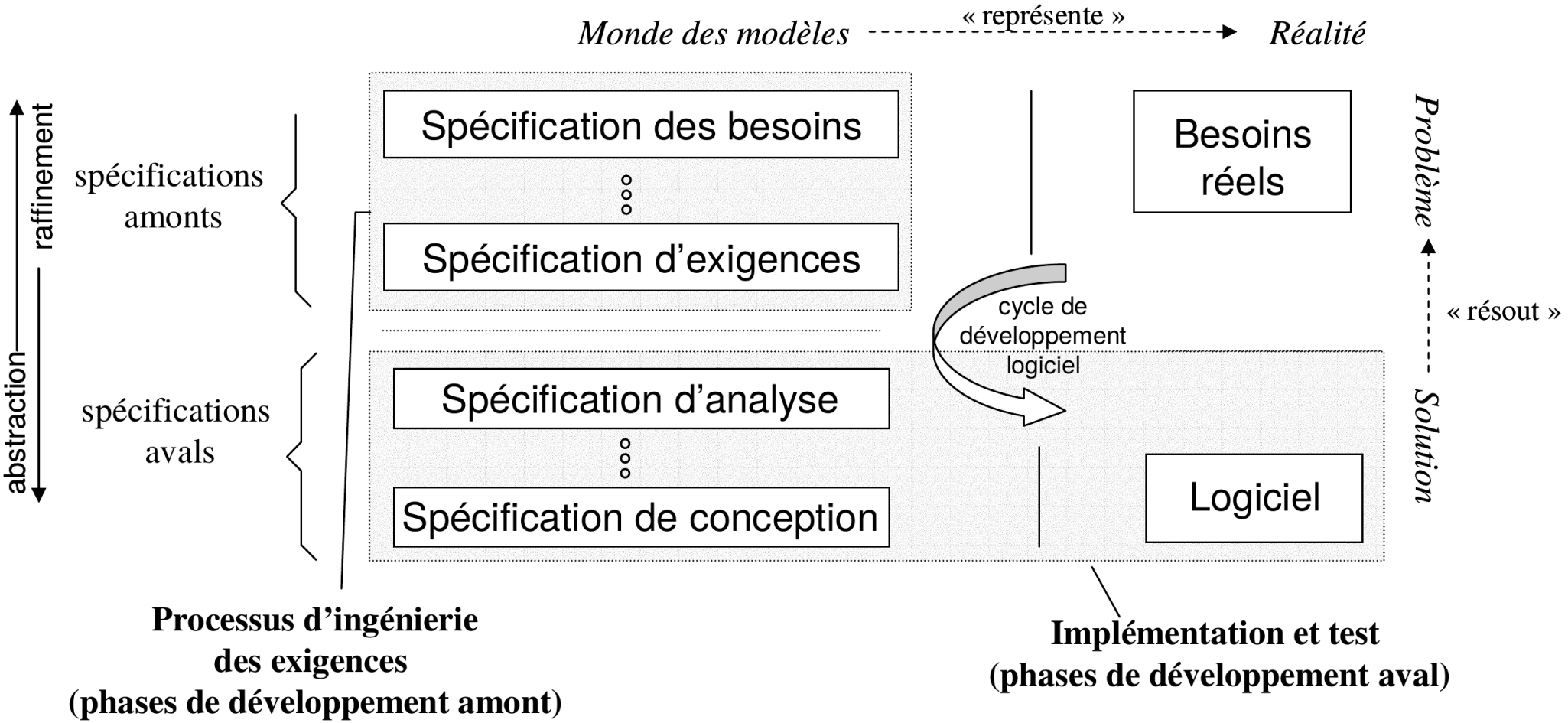
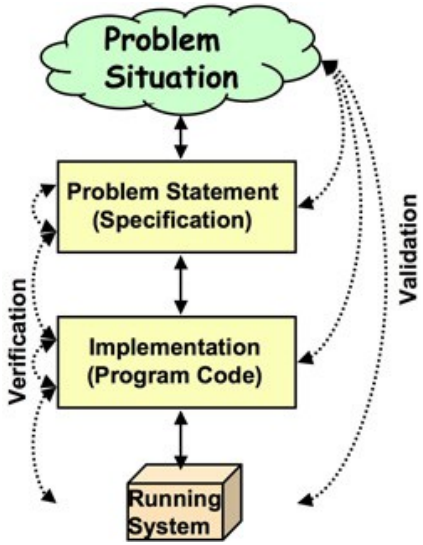


Figure 2 – Le processus d'ingénierie des exigences au sein du processus de développement logiciel.

Figure taken from Erwan Brottier's phd thesis: https://tel.archives-ouvertes.fr/tel-00512174/file/Brottier09.pdf

V&V ?



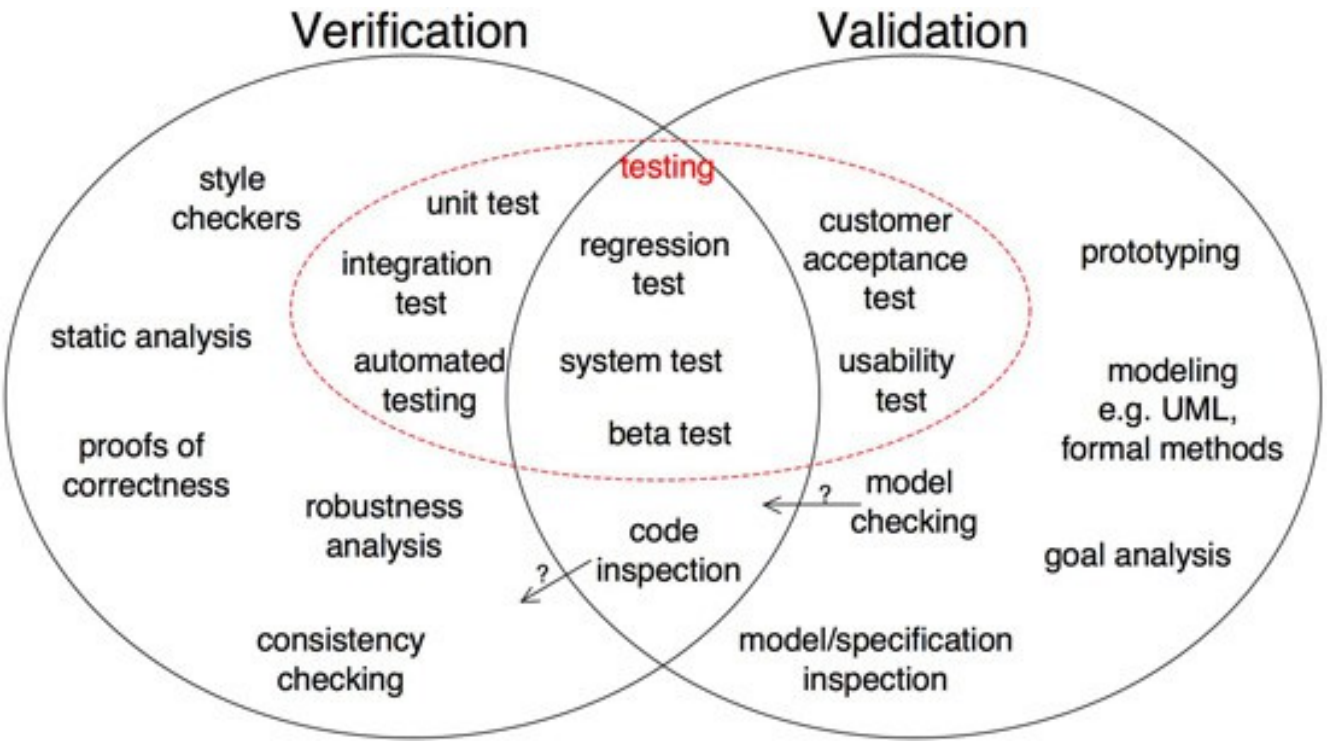
Picture taken from Steve EasterBrook (http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/)

- Verification and Validation
  - Verification: Construisons-nous le système correctement ?
    - Est-ce que le système est implémenté de manière correcte ? (sans erreur, avec les bonnes performances, sans fuites mémoires, rafinement correct, etc)
  - Validation: Construisons-nous le bon système ?
    - Est-ce que le système que nous développons est celui que le client voulait / qui répond au problème initial ?

→ Dans les deux cas on "**pose des questions au système**".

V&V ?

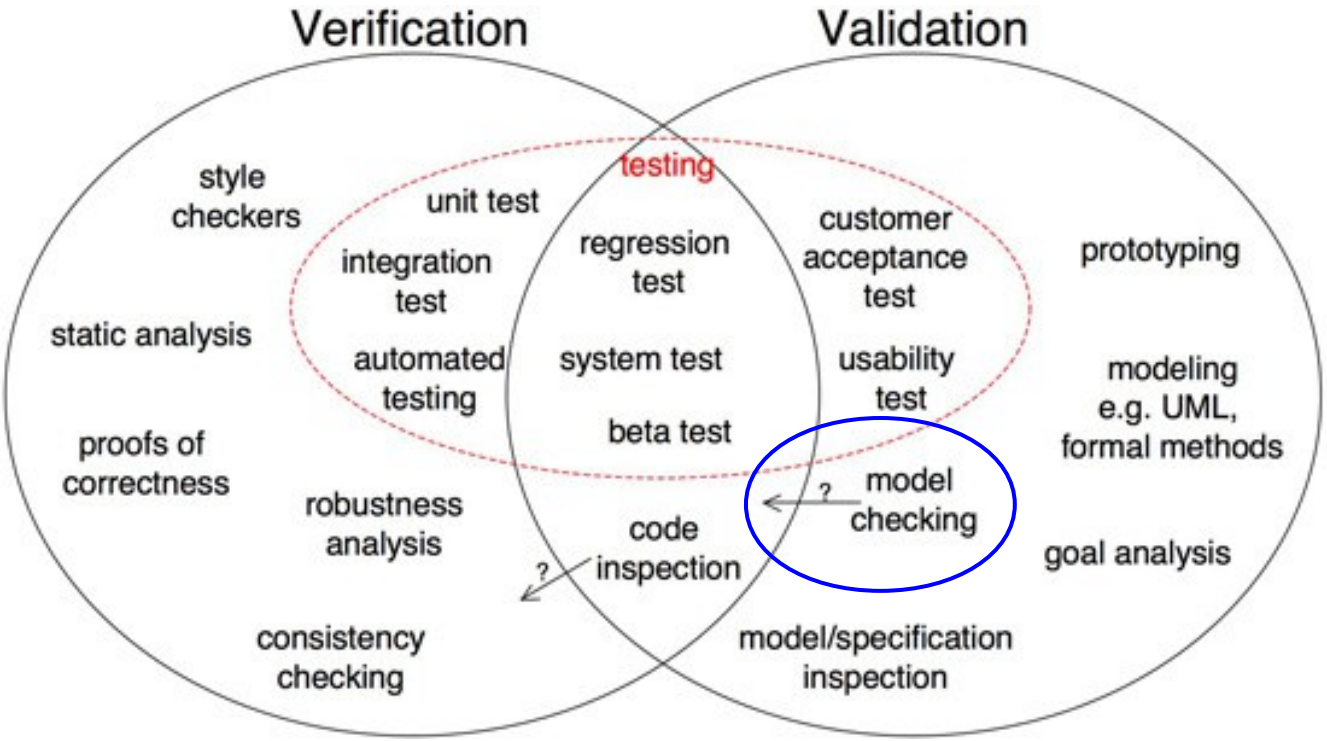
- Verification and Validation



Picture taken from Steve EasterBrook (http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/)

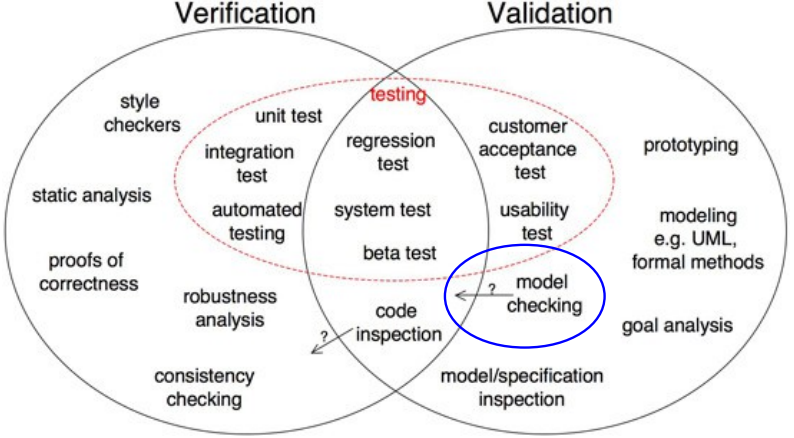
V&V ?

- Verification and Validation



Picture taken from Steve EasterBrook (http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/)

V&V ?



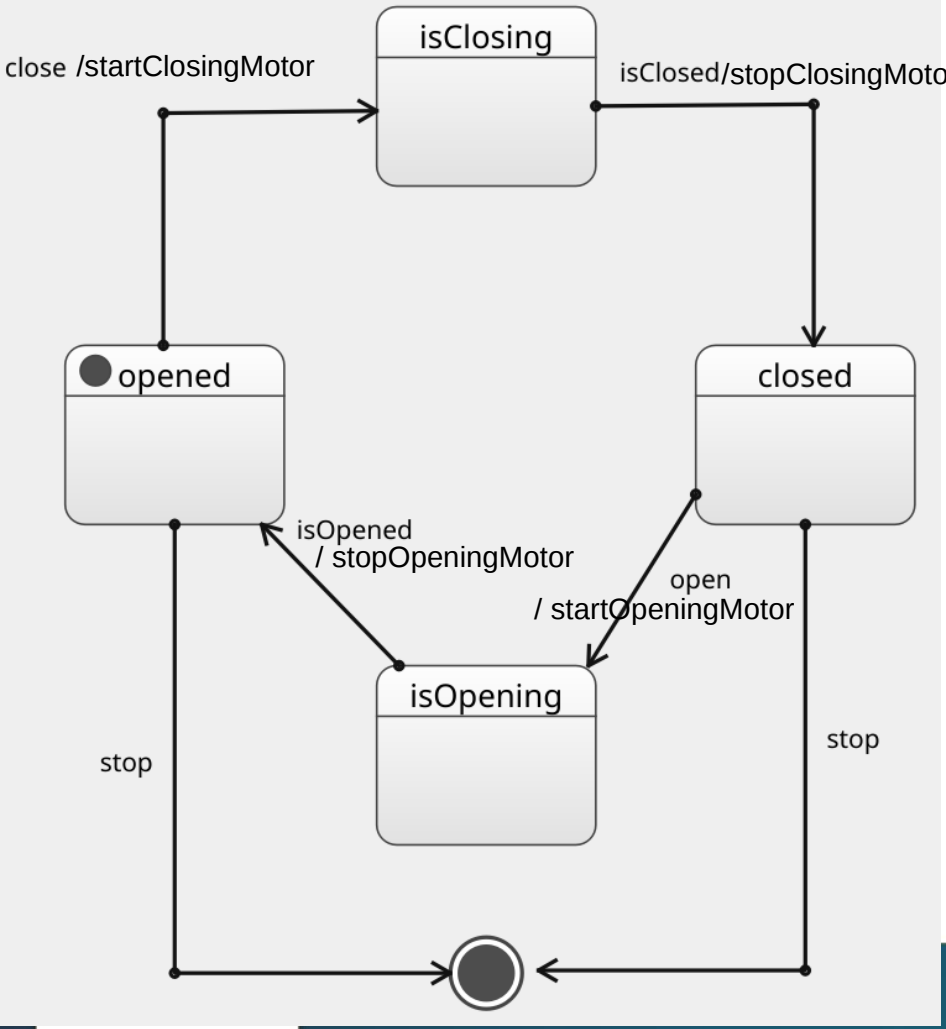
- Testing or model checking ? (intuition)
  - testing: regarder si certains chemins d'exécutions donnent le résultat attendu. On pose autant de questions que nécessaires pour vérifier une propriété du système.

    → taux de couverture ? Nombre de tests ?
  - model checking: regarder si tous les chemins d'exécution donnent le résultat attendu. On exprime une propriété sous la forme d'une expression

    → ensemble de chemins d'exécutions finis ? Quel type de propriétés ?

V&V ?

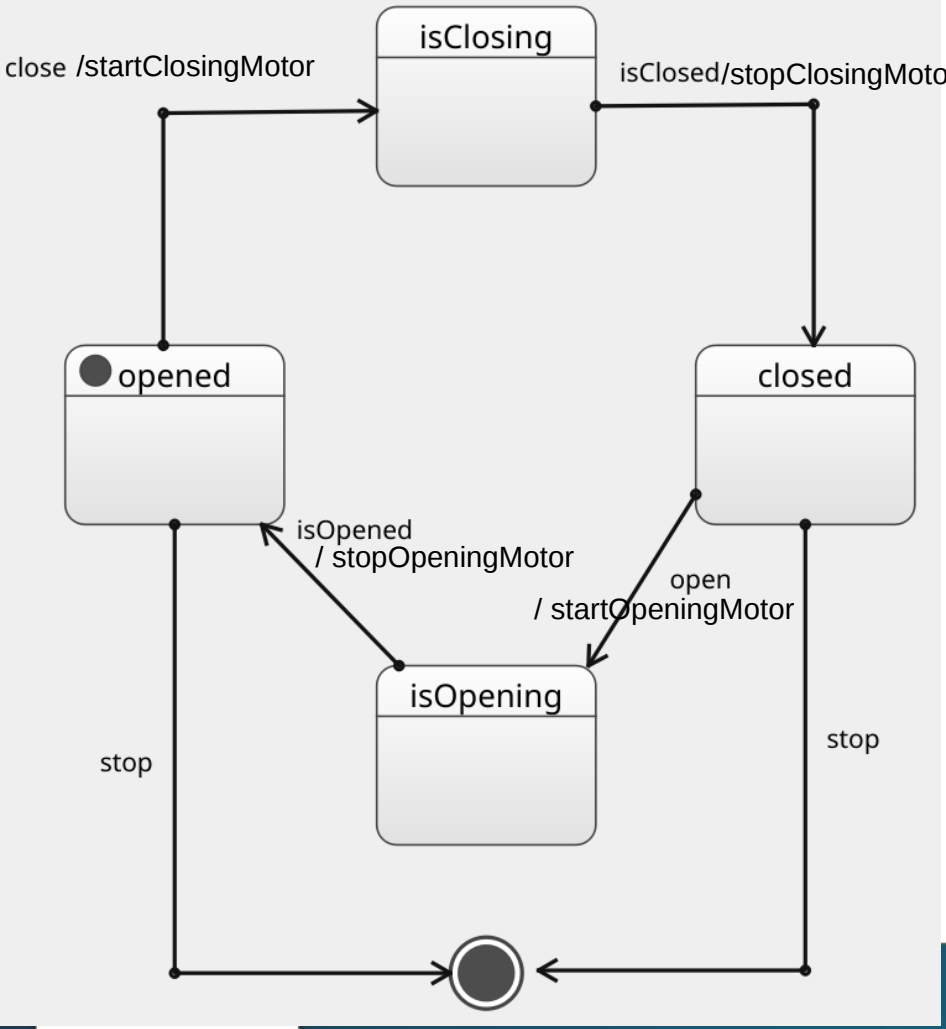
- ensemble de chemins d'exécutions finis ?
  - Énumération de l'espace d'état (habituellement un graphe orienté d'une forme particulière : *Labelled Transition system* or *Kripke structure*)

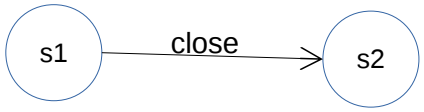


s1

V&V ?

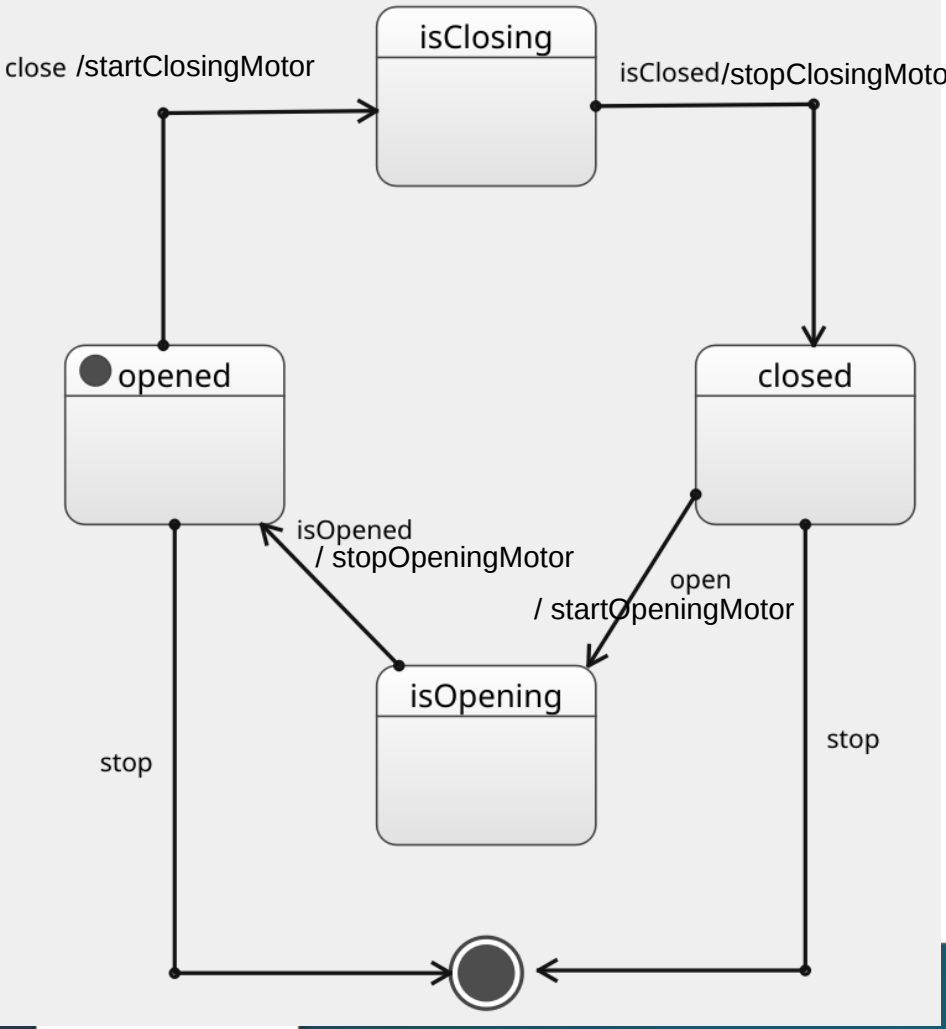
- ensemble de chemins d'exécutions finis ?
  - Énumération de l'espace d'état (habituellement un graphe orienté d'une forme particulière : *Labelled Transition system* or *Kripke structure*)

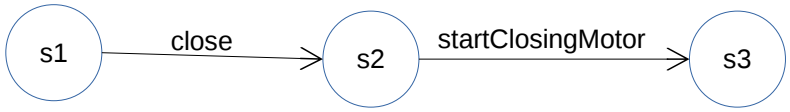




V&V ?

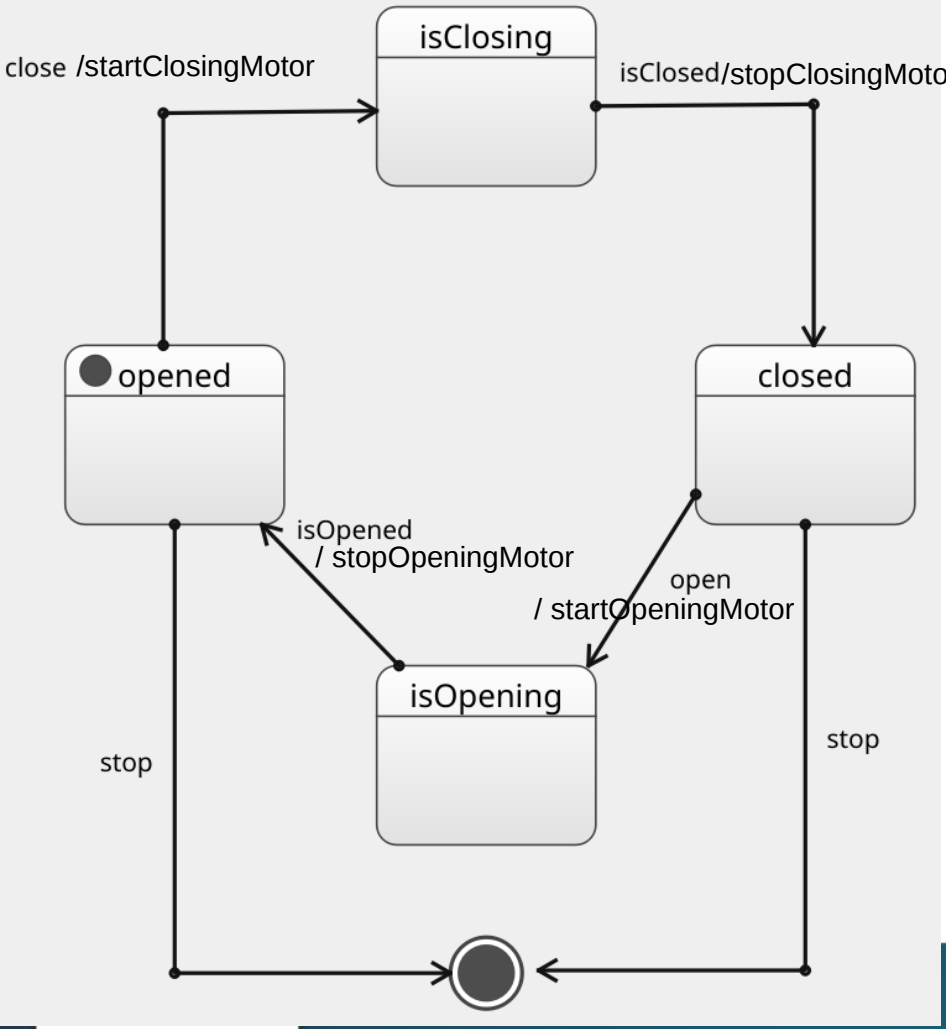
- ensemble de chemins d'exécutions finis ?
  - Énumération de l'espace d'état (habituellement un graphe orienté d'une forme particulière : *Labelled Transition system* or *Kripke structure*)

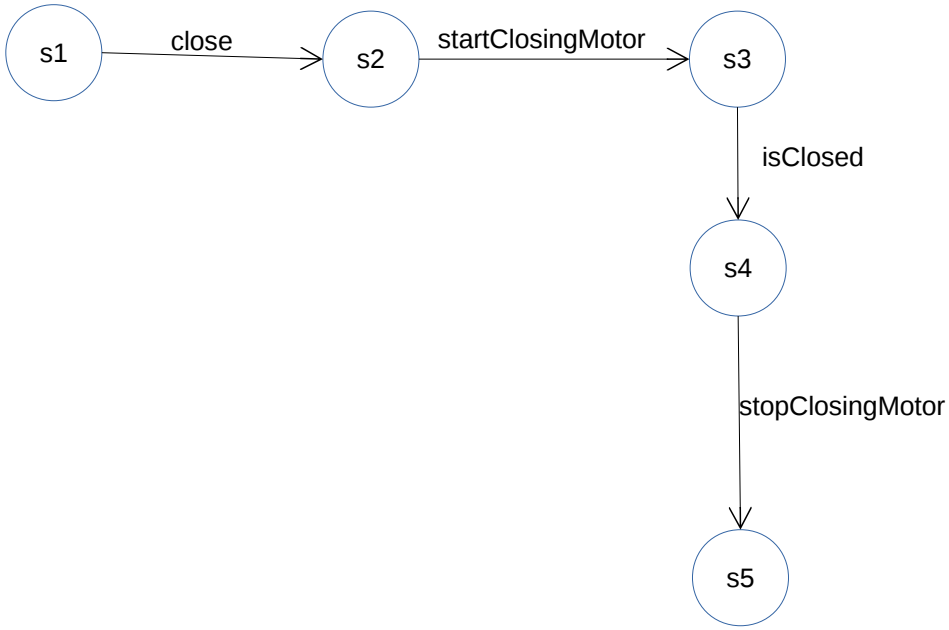




V&V ?

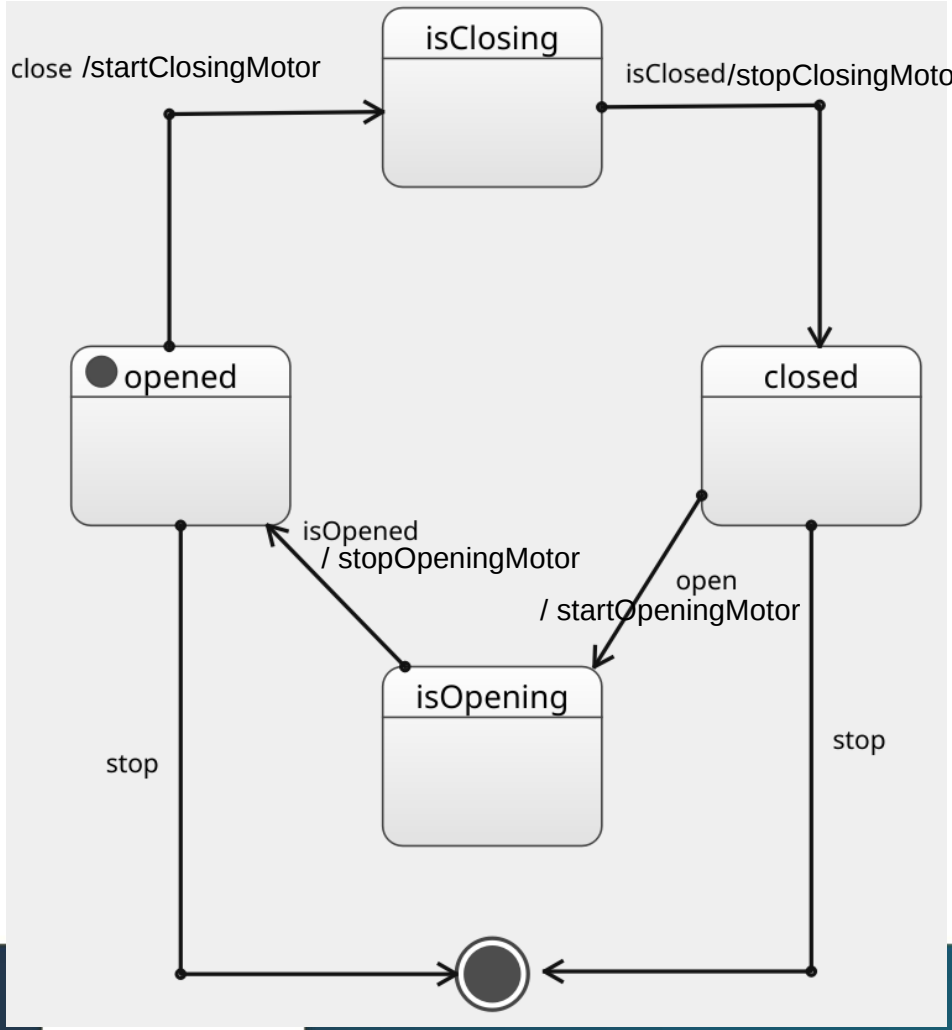
- ensemble de chemins d'exécutions finis ?
  - Énumération de l'espace d'état (habituellement un graphe orienté d'une forme particulière : *Labelled Transition system* or *Kripke structure*)

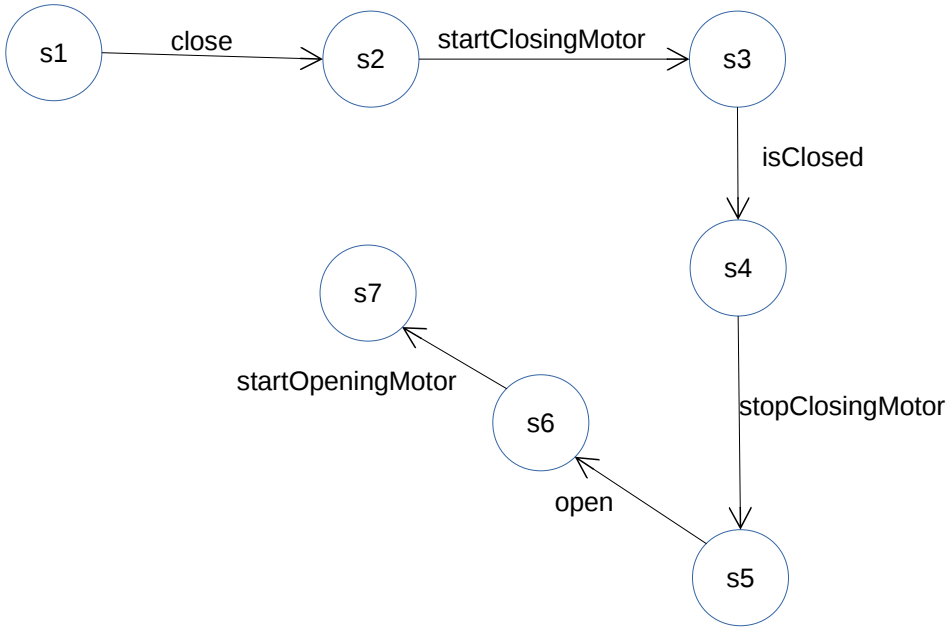




V&V ?

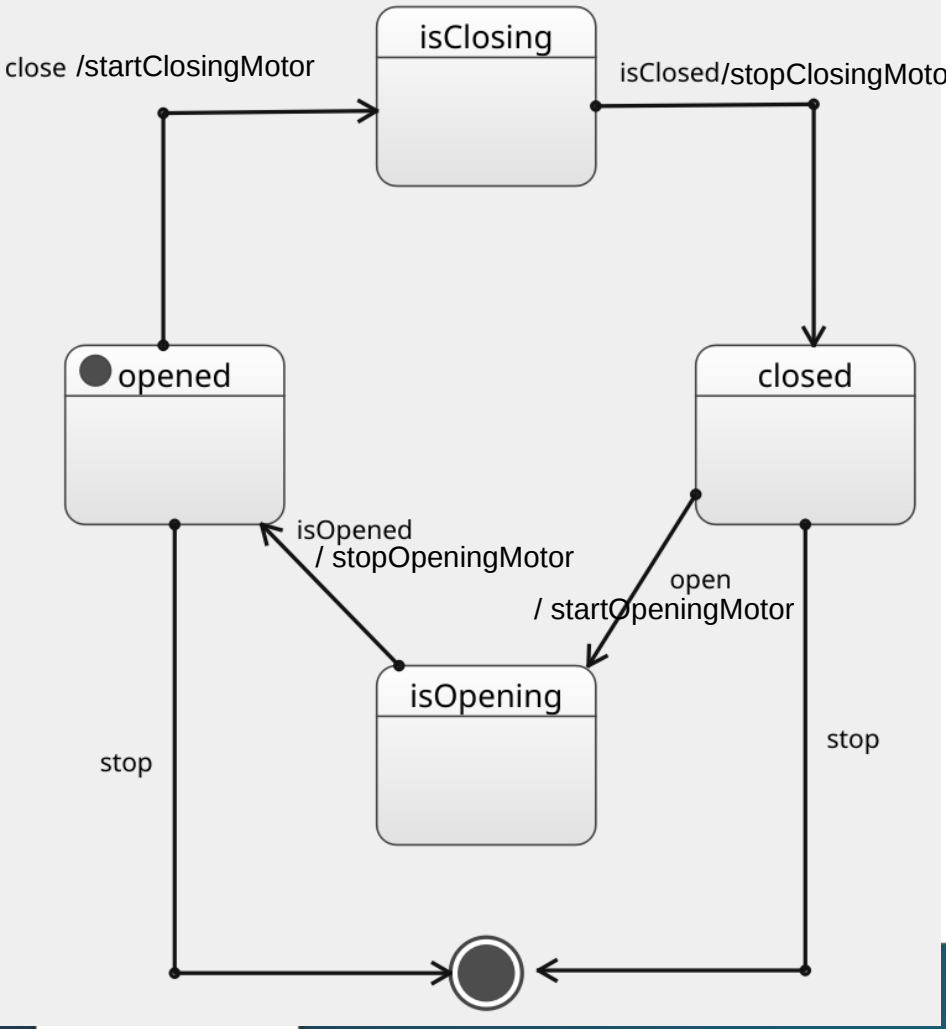
- ensemble de chemins d'exécutions finis ?
  - Énumération de l'espace d'état (habituellement un graphe orienté d'une forme particulière : *Labelled Transition system* or *Kripke structure*)

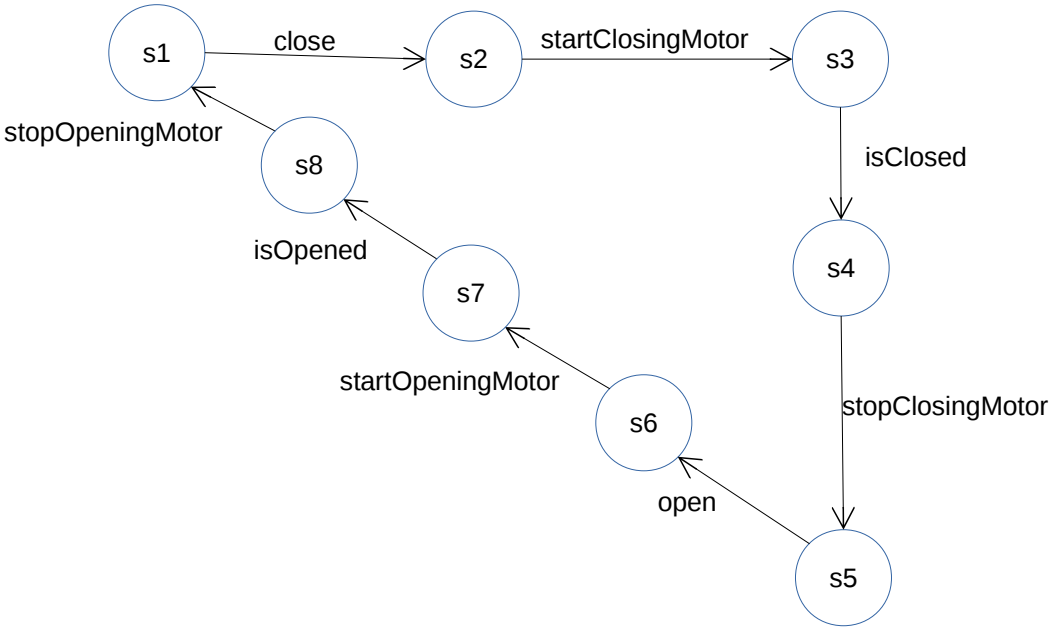




V&V ?

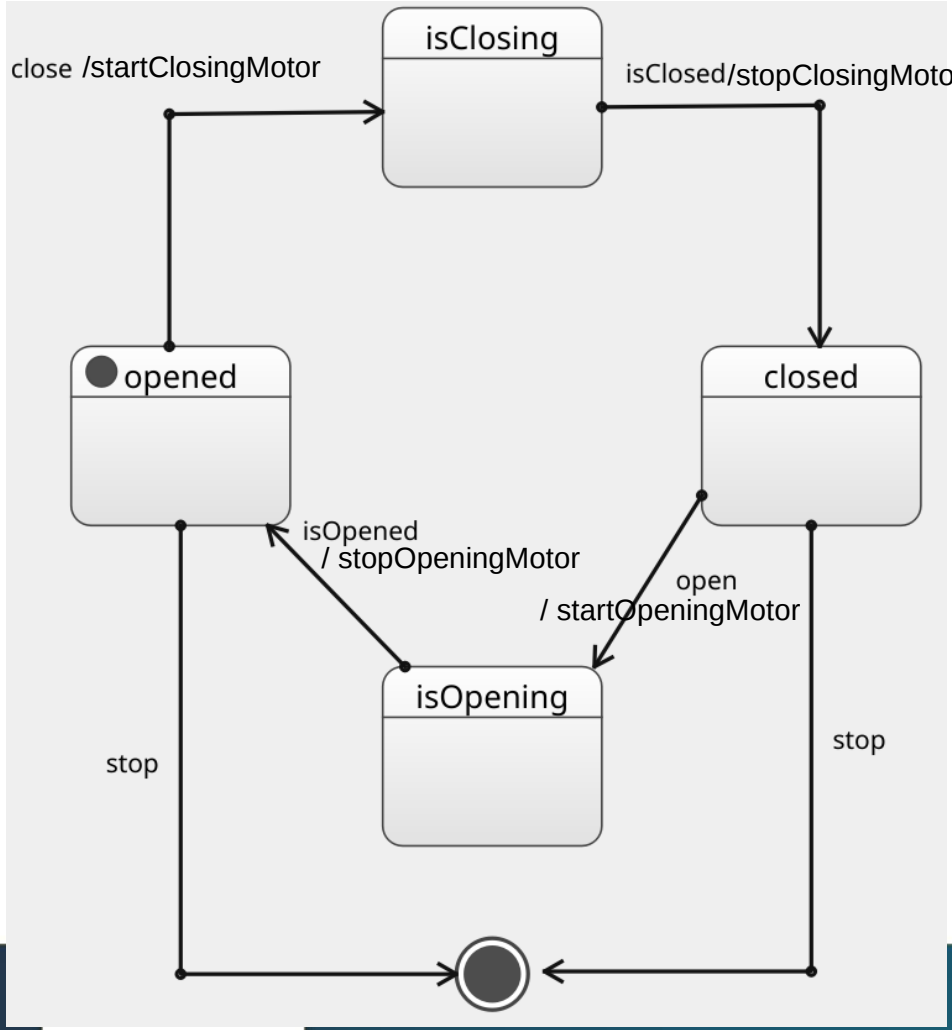
- ensemble de chemins d'exécutions finis ?
  - Énumération de l'espace d'état (habituellement un graphe orienté d'une forme particulière : *Labelled Transition system* or *Kripke structure*)

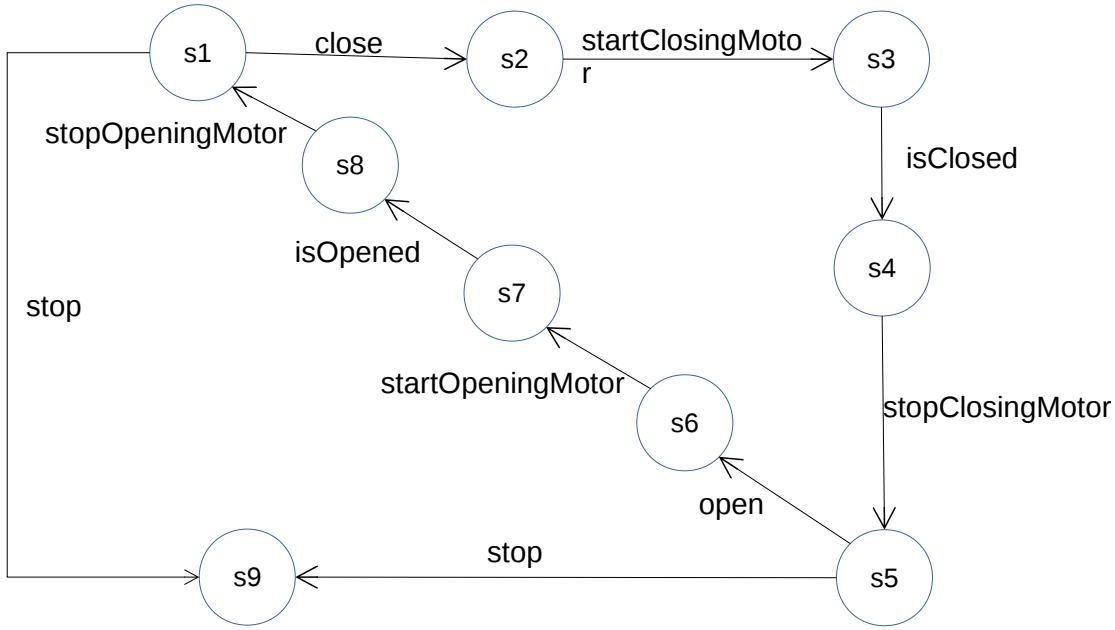




V&V ?

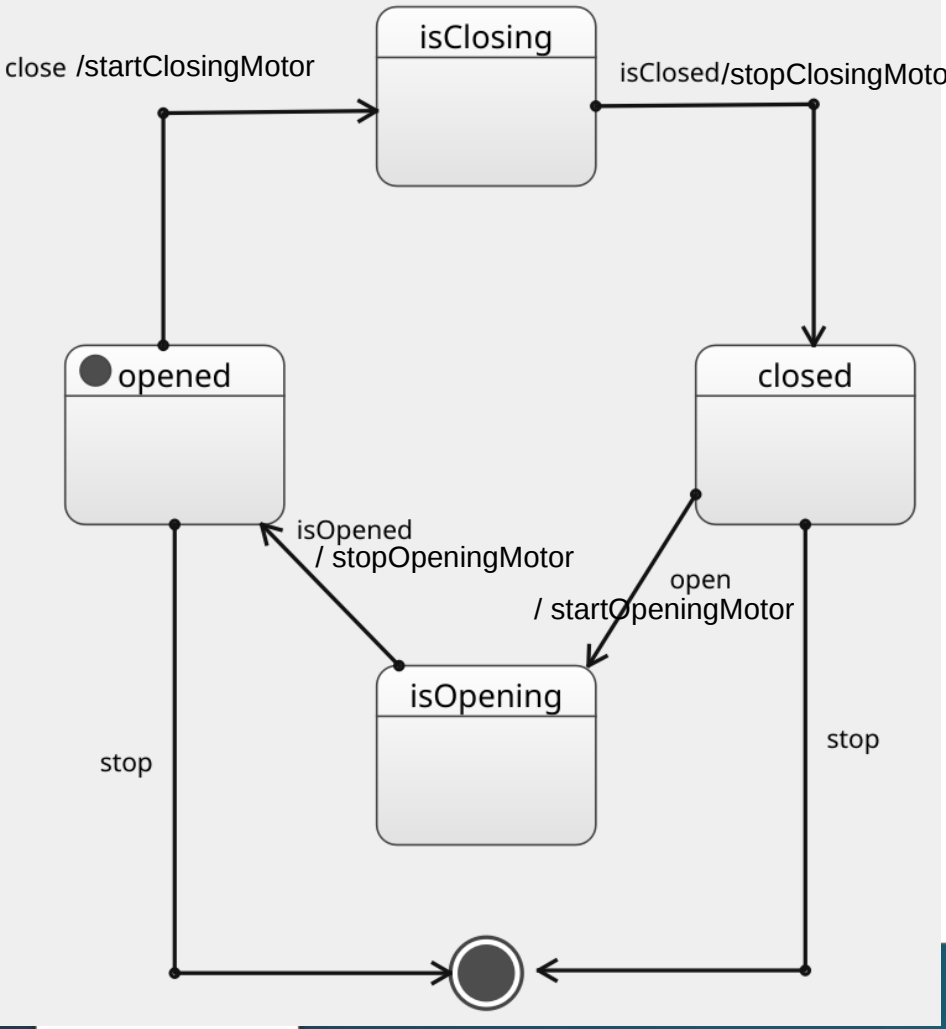
- ensemble de chemins d'exécutions finis ?
  - Énumération de l'espace d'état (habituellement un graphe orienté d'une forme particulière : *Labelled Transition system* or *Kripke structure*)

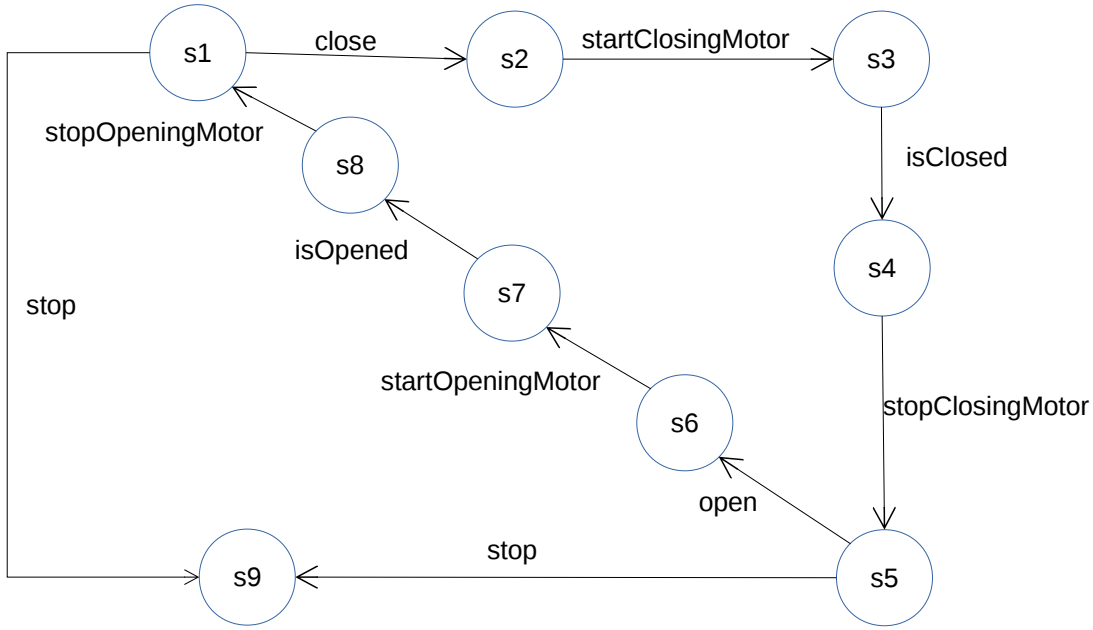




V&V ?

- ensemble de chemins d'exécutions finis ?
  - Énumération de l'espace d'état (habituellement un graphe orienté d'une forme particulière : *Labelled Transition system* or *Kripke structure*)

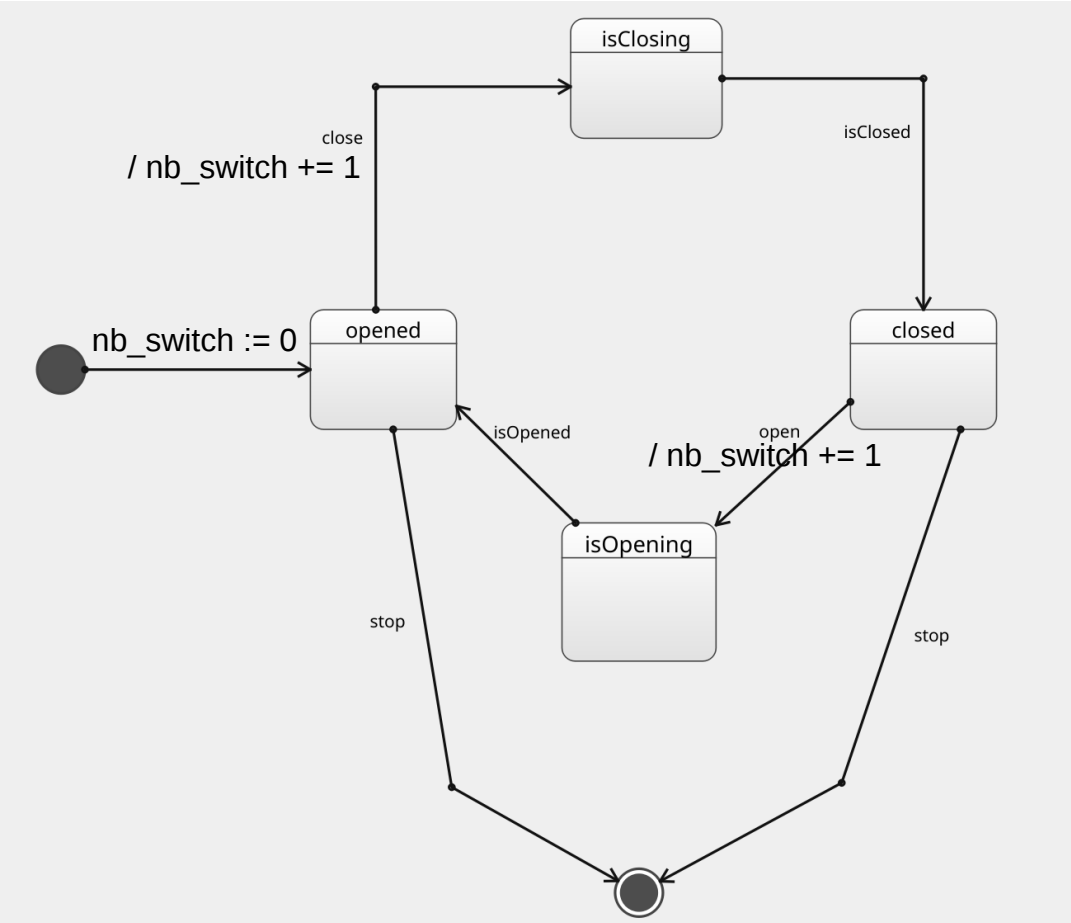




Plus les actions *onEnter* et *onExit !*

V&V ?

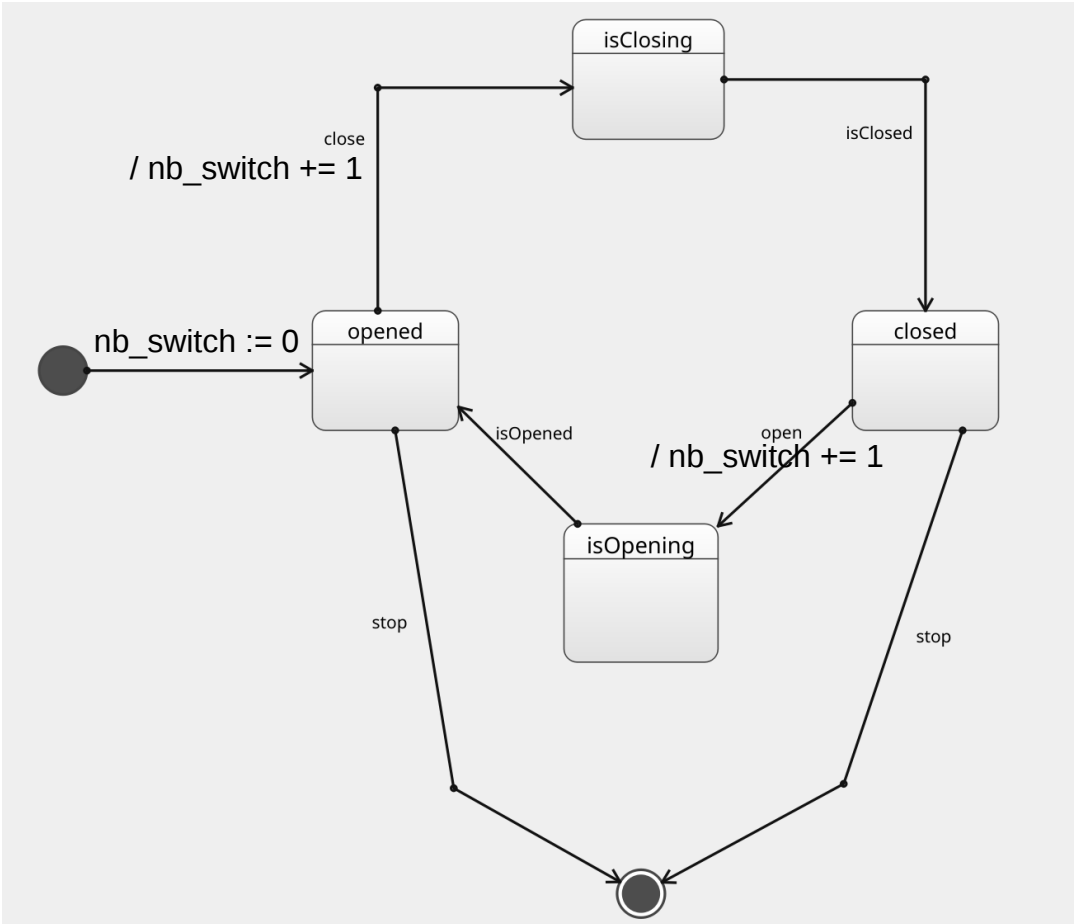
- ensemble de chemins d'exécutions finis ?

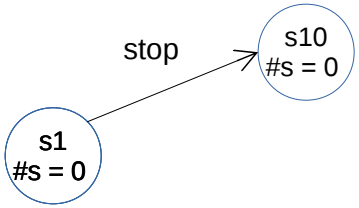


s1
#s = 0

V&V ?

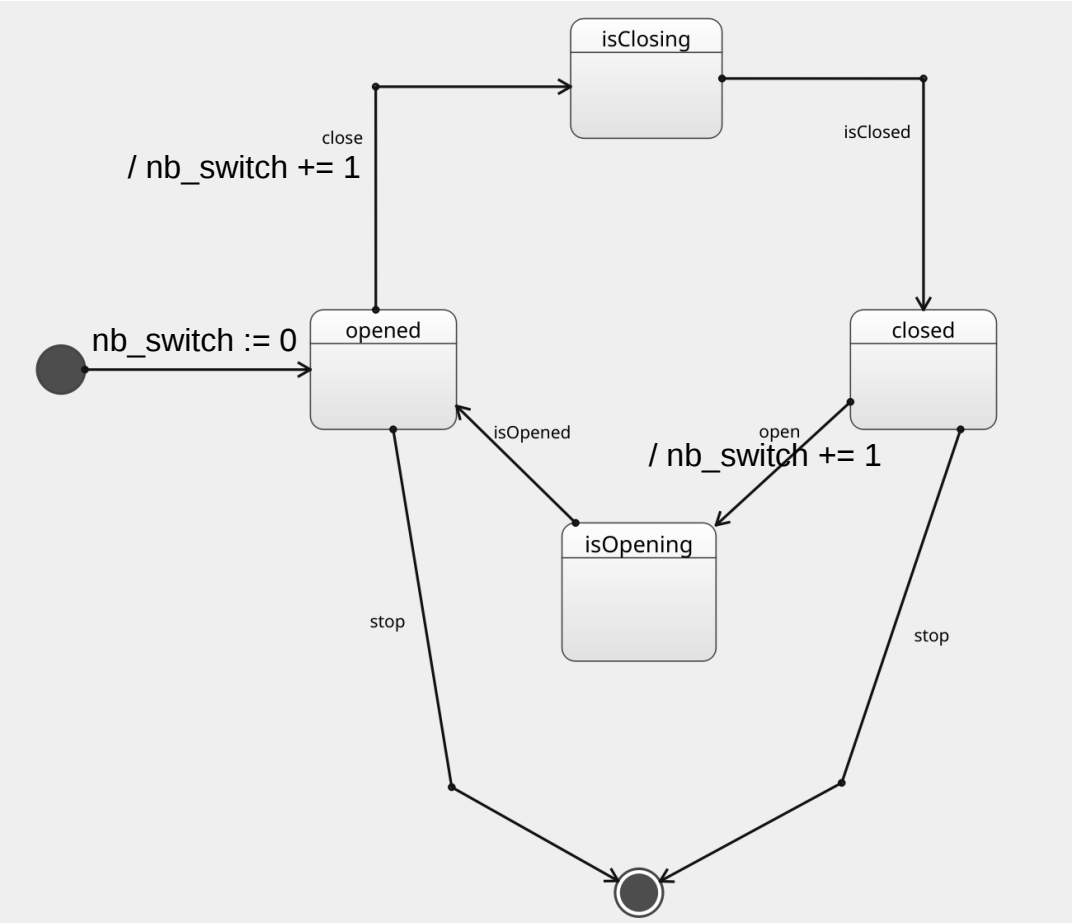
- ensemble de chemins d'exécutions finis ?

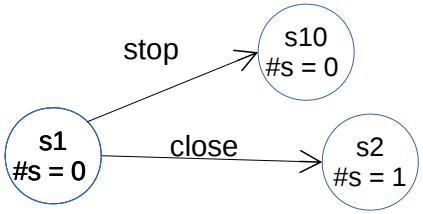




V&V ?

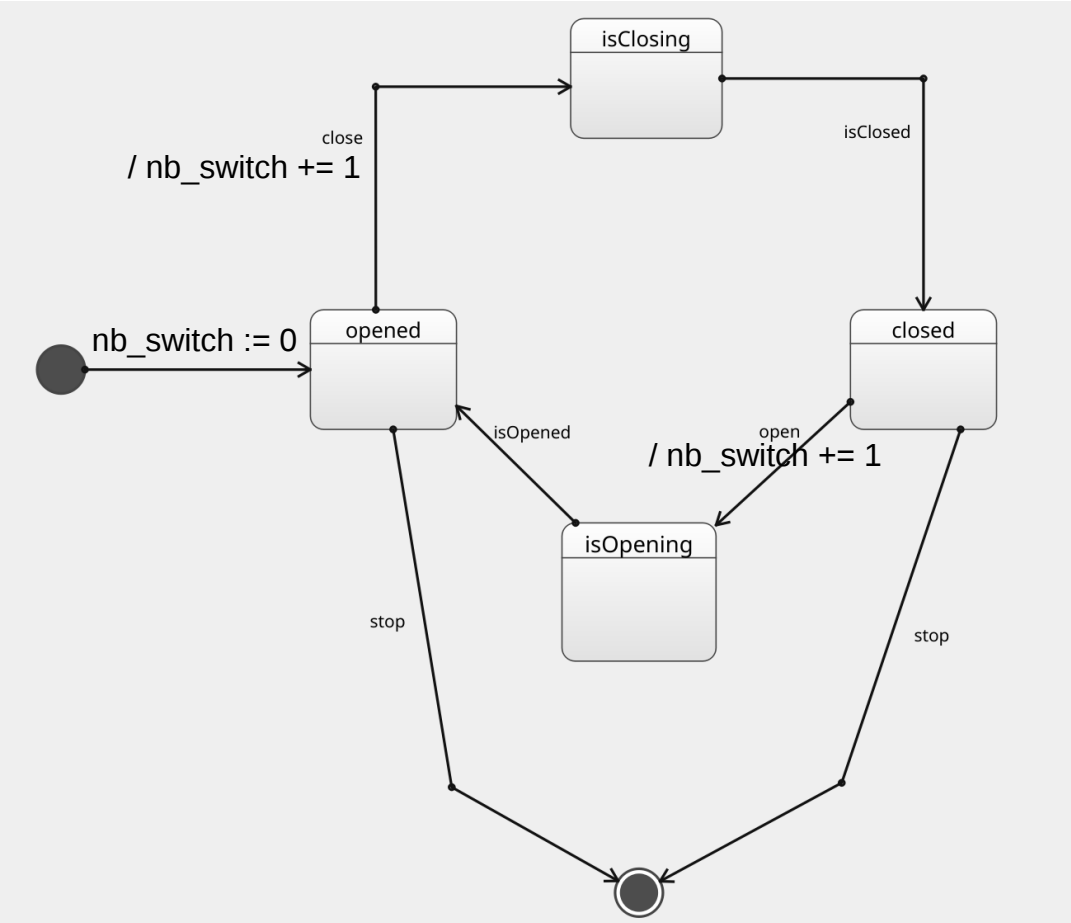
- ensemble de chemins d'exécutions finis ?

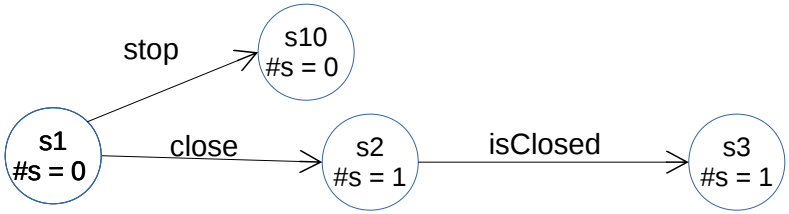




V&V ?

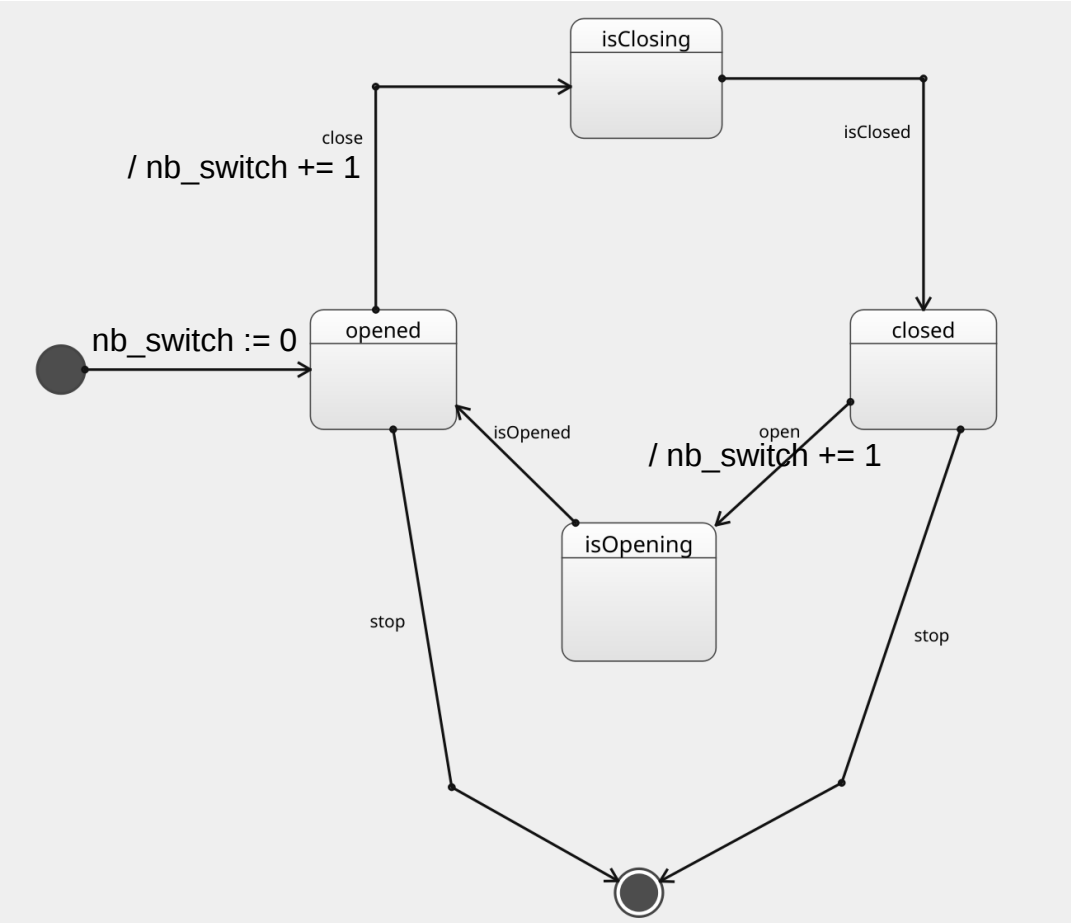
- ensemble de chemins d'exécutions finis ?

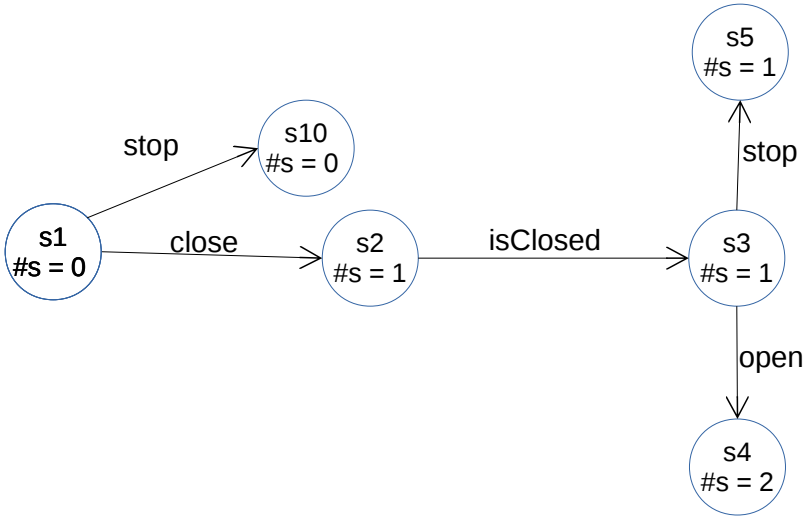




V&V ?

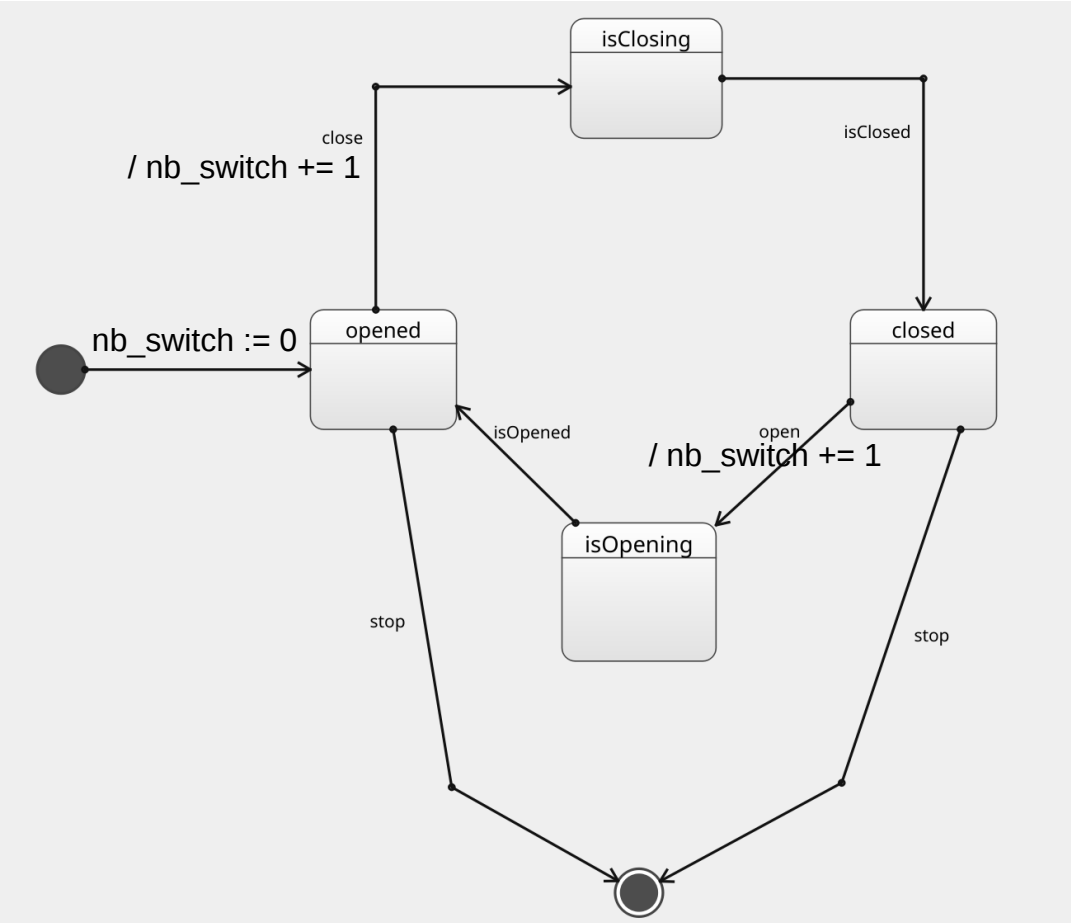
- ensemble de chemins d'exécutions finis ?

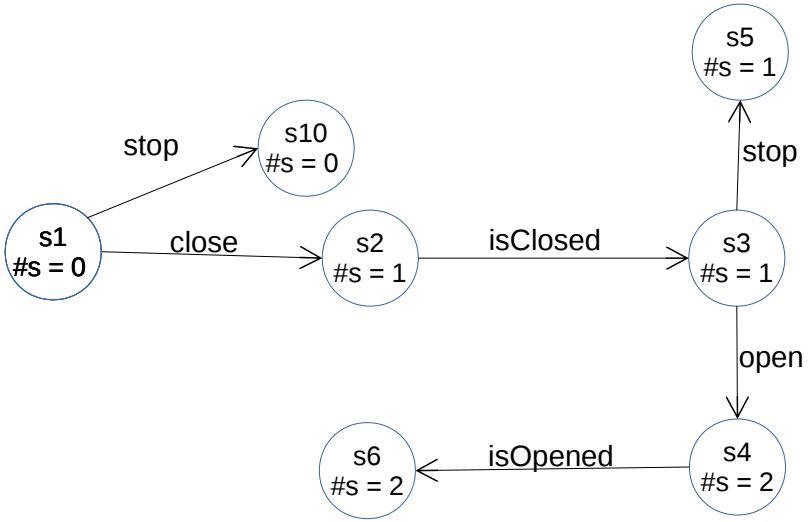




V&V ?

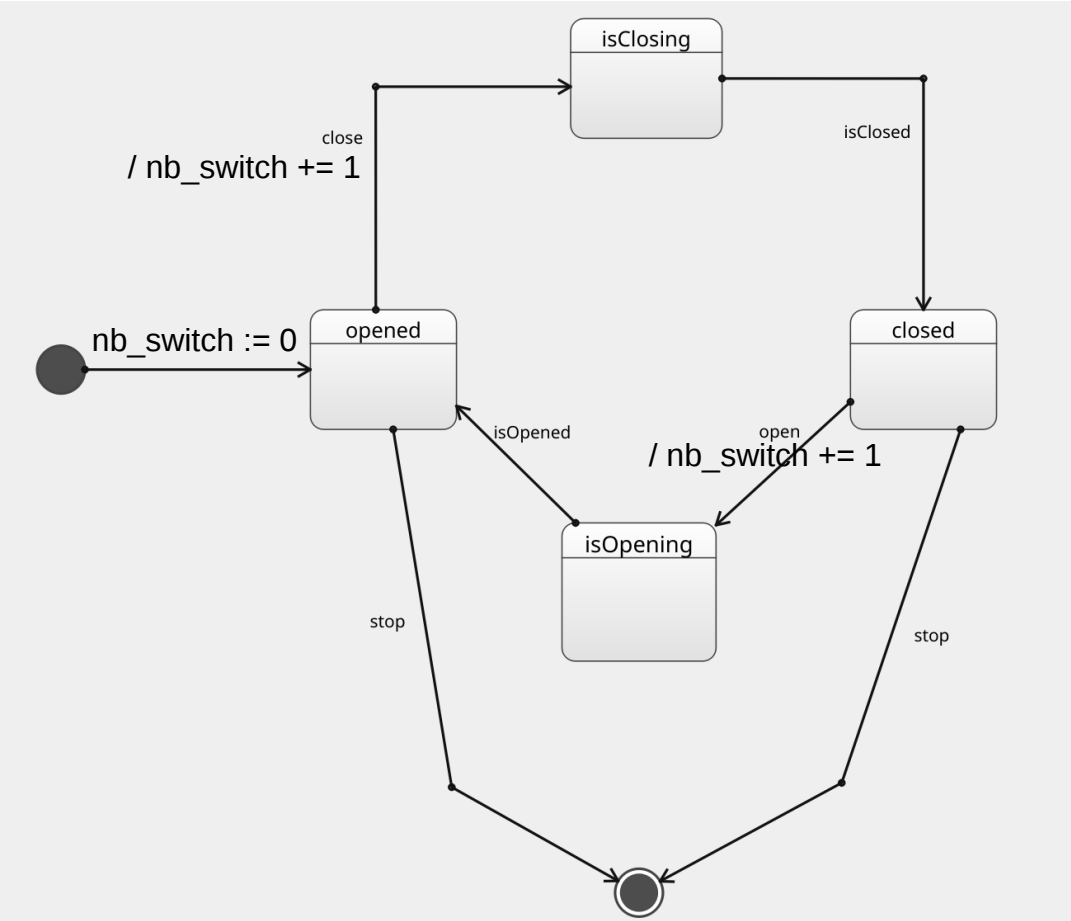
- ensemble de chemins d'exécutions finis ?

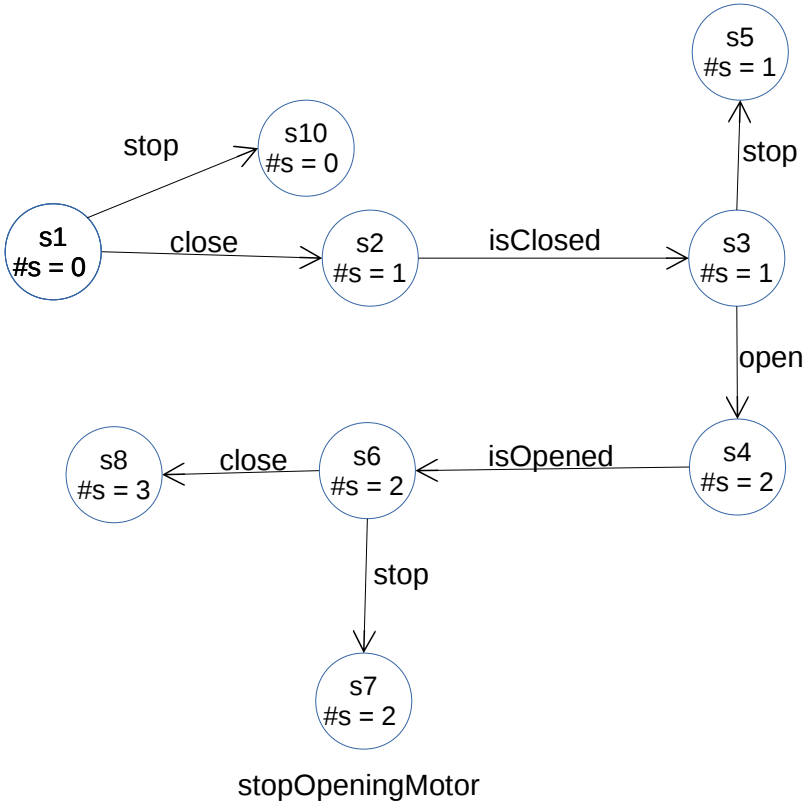




V&V ?

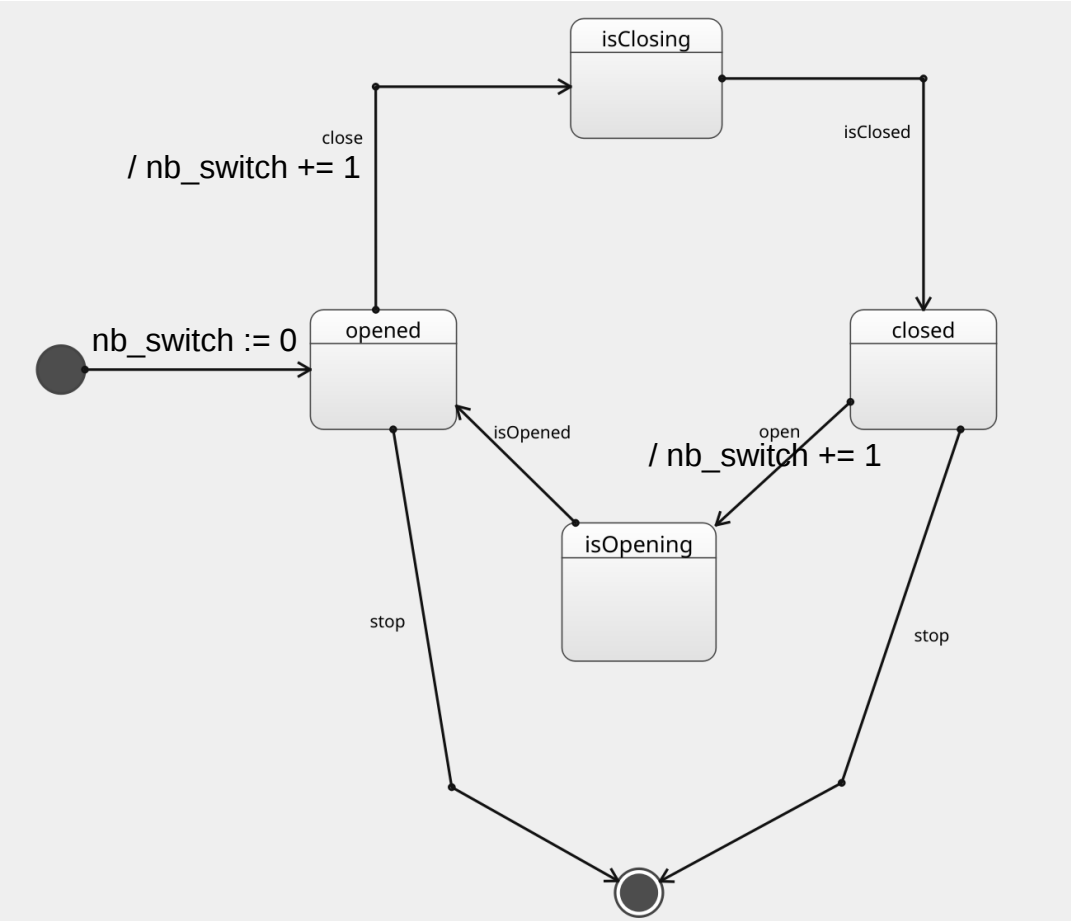
- ensemble de chemins d'exécutions finis ?

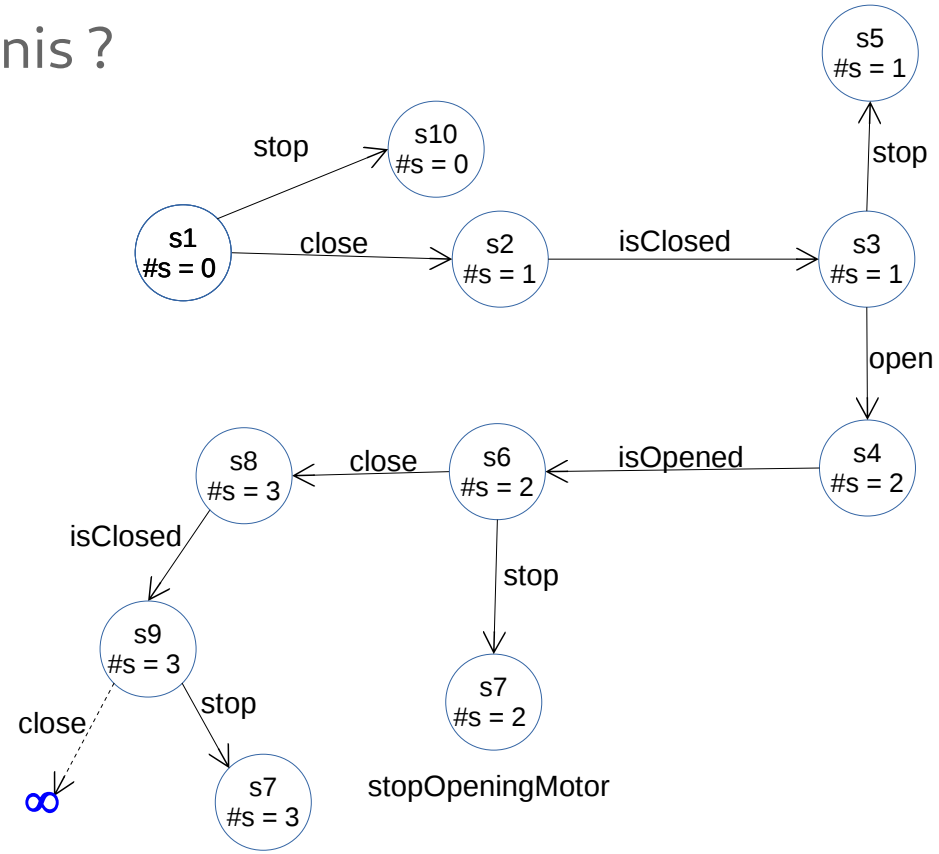




V&V ?

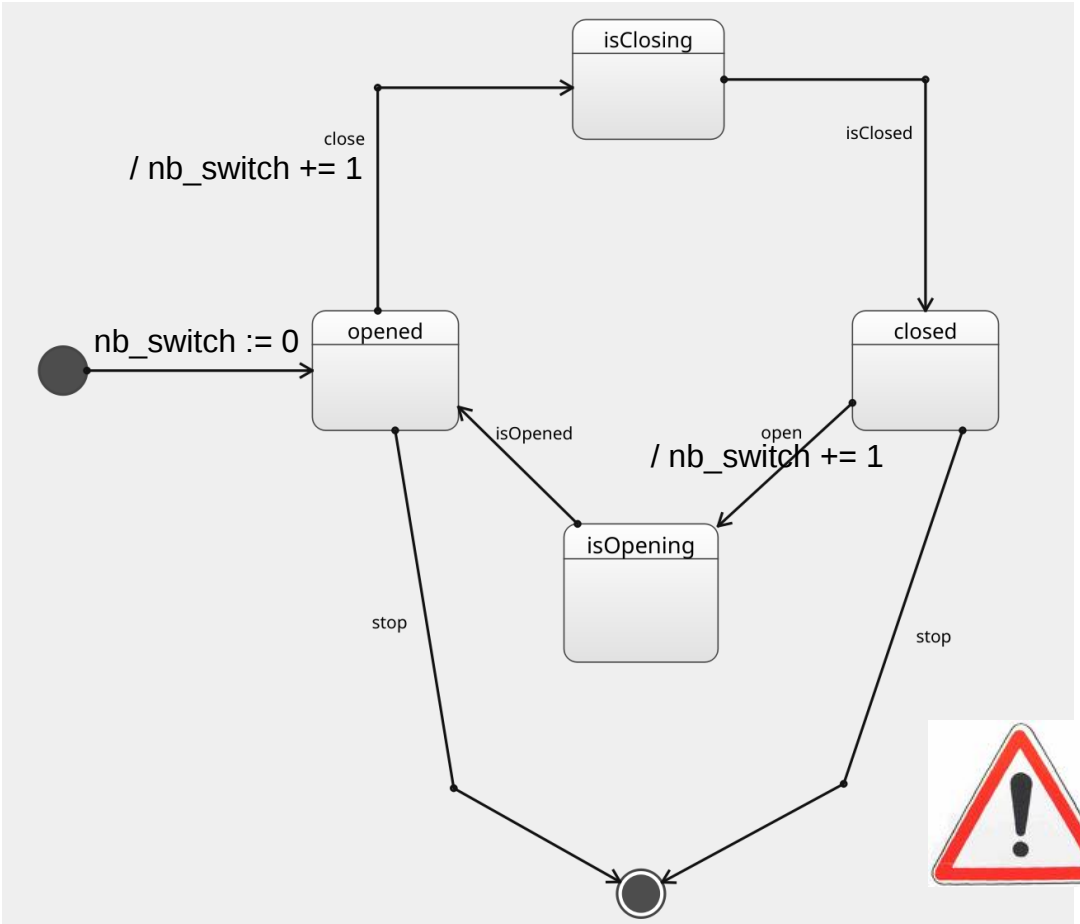
- ensemble de chemins d'exécutions finis ?

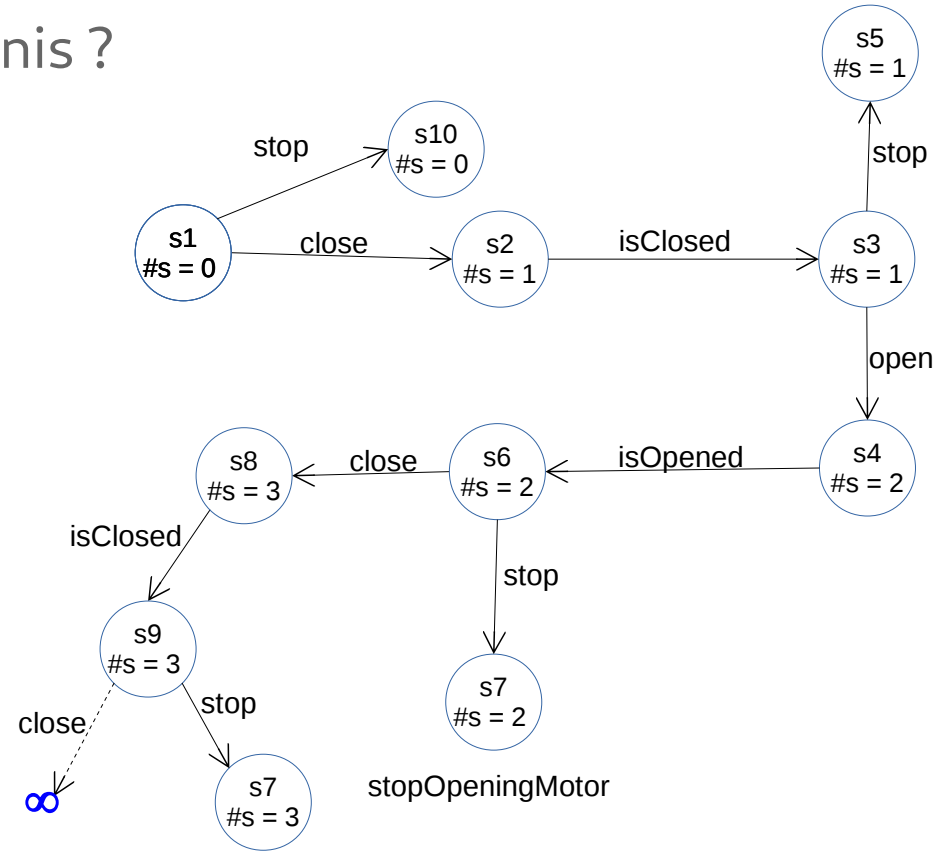




V&V ?

- ensemble de chemins d'exécutions finis ?





Tout ce qui est dans la state machine est considéré dans la construction de l'espace d'état.

Tout ce qui n'est pas dans la state machine ne peut pas être utilisé pour "poser des questions"