CIS 623 Case Study 3

In this case study, we have defined a Kripke model state machine and applied some properties on it in order to verify the satisfiability for generation a concrete model. The complete source code for the alloy for this case study is given below:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Part I.a: In this we create the signature for the statemachine, states and properties

and define basic facts that are applicable on it.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

module KripkeModel

// Represents a property (atom) contained within each state

sig Prop { }

// Represents a state with mapping to its properties

sig State {

prop: set Prop

}

// The model itself

sig StateMachine {

// set of all states used in the model

states: set State,

// set of initial states

init: set states,

// state transitions

next: states -> states,

// set of final states

final: set states

} {

// basic facts about the model

// We have non-empty set of initial states (as mentioned in text)

some init

// We have non-empty set of distinct final states (as given in case study)

some final

// Initial and final states are distinct

no init & final

// For all final states, if there'a next state, then its that state itself

all f: final | f.next in { f }

// Some non-final state need to transition to final state

some s: (states - final) | some s.next and s.next in final

// Initial state transitions need to be a non-initial state

no (init.next & init)

}

pred show { }

run show for 3 State, 3 Prop, 1 StateMachine

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Part I.b: In this we define the 'Reaches' predicate that can test the reachability of

a set of states from the initial states of a given model

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Predicate to check whether all states are reachable from initial state

pred Reaches [m: StateMachine, s: set State] { s in (m.init).\*(m.next) }

// Run Reaches for 3 states, 3 properties and 1 state machine

run Reaches for exactly 5 State, exactly 3 Prop, exactly 1 StateMachine

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Part I.c: Implement DeadlockFree, Deterministic, Reachability and Liveness predicates

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// i. Predicate to ensure that only final states can deadlock

pred DeadlockFree [m: StateMachine] { all s: m.states | (s.(m.next) = s) => (s in m.final) }

run DeadlockFree for exactly 5 State, exactly 3 Prop, exactly 1 StateMachine

// ii. Predicate to check that for every state reachable from init, it can have either

// zero or one next states

pred Deterministic [m: StateMachine] { all s: (m.init).\*(m.next) | lone s.(m.next) }

run Deterministic for exactly 5 State, exactly 3 Prop, exactly 1 StateMachine

// iii. Predicate to check that we can reach a state that has the given property set to true

pred Reachability [m: StateMachine, p: Prop] { some s: (m.init).\*(m.next) | p in s.prop }

run Reachability for exactly 5 State, exactly 3 Prop, exactly 1 StateMachine

// iv. Predicate to check that starting from any reachable state, we can reach another state

// that has given property

pred Liveness [m: StateMachine, p: Prop] { all r: (m.init).\*(m.next) | some s: r.\*(m.next) | p in s.prop }

run Liveness for exactly 5 State, exactly 3 Prop, exactly 1 StateMachine

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Part I.d: Implement Implies assertion

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

assert Implies { all m: StateMachine, p: Prop | Liveness[m, p] => Reachability[m, p] }

check Implies for exactly 15 State, exactly 5 Prop, exactly 1 StateMachine

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Part I.e: Implement Converse assertion which is the inverse of Part I.d

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

assert Converse { all m: StateMachine, p: Prop | Reachability[m, p] => Liveness[m, p] }

// As expected the Converse is not true because Reachability requires just a single path

// from init to reach to the state that contains p. But that does not mean for every other

// state in the model, there exists a path from that state to the state having property p.

check Converse for exactly 5 State, exactly 3 Prop, exactly 1 StateMachine

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Part I.d: Predicate to check model for Figure 2.15 in the text book for the following

properties:

a. no non-final state deadlocks; which means that all non-final state

should have a next state

b. two states with same properties are identical

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

pred Figure215 [m: StateMachine] {

all s: m.states - m.final | some s.(m.next)

all x,y: m.states | (x.prop = y.prop) => (x = y)

}

run Figure215 for exactly 2 Prop, exactly 3 State, exactly 1 StateMachine

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Part II: Implement the following

a. NoCycle predicate that says a state s that has property p can not be

within a cycle. A cycle can be detected if that state is reachable from

itself through 'next'.

b. EventuallyFails predicate that says the model can start from an initial

state where it can eventually reach a state where p fails and remain

so from there onwards.

c. Check if NoCycle => EventuallyFails and vice versa

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// For all states that are reachable from init, if the state has p property, then that state

// can not exists in any cycle.

pred NoCycle [m: StateMachine, p: Prop] {

// since the requirement did not say the states need to reachable from initial state,

// we consider all the states

all s: m.states |

(p in s.prop) => { all t: (m.init).\*(m.next) | (t in t.\*(m.next)) => (s not in t.\*(m.next)) }

}

run NoCycle for exactly 5 State, exactly 3 Prop, exactly 1 StateMachine

// For all states reachable from init, if that state does not have p property, then

// for all states reachable from that state, p is unsatisfied.

pred EventuallyFails [m: StateMachine, p: Prop] {

all s: (m.init).\*(m.next) |

(p not in s.prop) => { all t: s.\*(m.next) | p not in t.prop }

}

run EventuallyFails for exactly 5 State, exactly 3 Prop, exactly 1 StateMachine

assert Implies2 { all m: StateMachine, p: Prop | NoCycle[m,p] => EventuallyFails[m,p] }

check Implies2 for exactly 6 State, exactly 3 Prop, exactly 1 StateMachine

assert Converse2 { all m: StateMachine, p: Prop | EventuallyFails[m,p] => NoCycle[m,p] }

check Converse2 for exactly 6 State, exactly 3 Prop, exactly 1 StateMachine

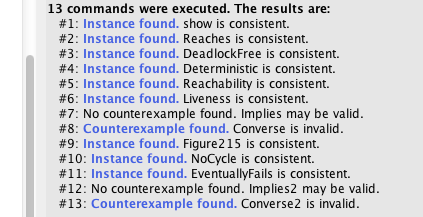
The above alloy source defines the following signatures:

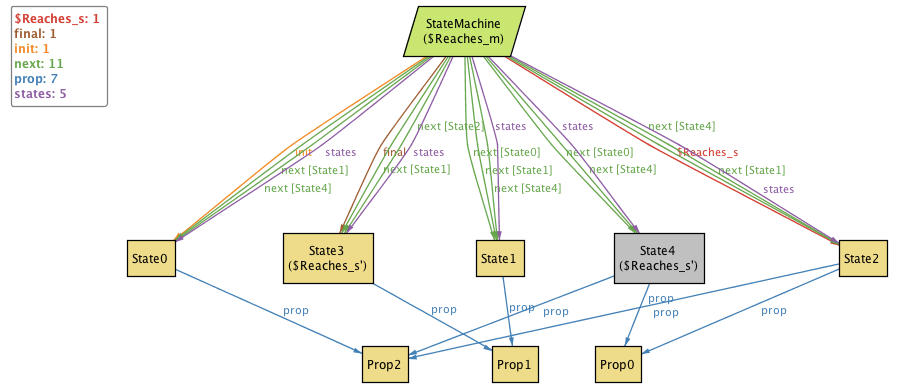
* 1. Prop - defines each atomic property that a state can contain
  2. State - defines a state in the state machine which can contain some properties in it
  3. StateMachine - defines the signature for the Kripke model which will be used to validate some predicates and assertions using these predicates.

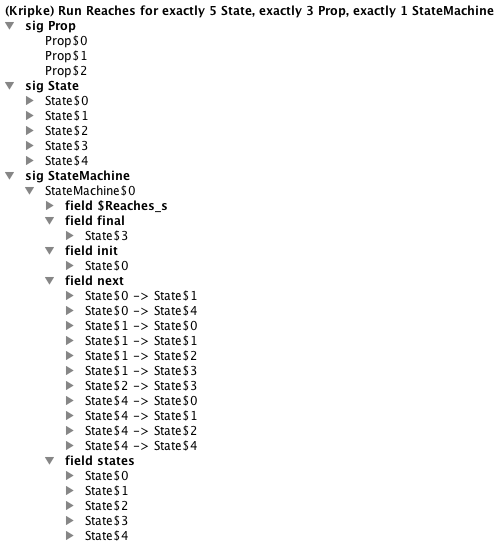
Some of the basic facts that we have defined for the model are:

* + There has to be at least one init state (as per text)
  + There has to be at least one final state (as per case study)
  + A state can be be both init and final states which we can presume in any generic state machine
  + We also know that once the machine reaches the final state, it should not switch to any other state. The only state that it could switch to, if it wanted, was itself - which is basically a deadlock in the final state which was also permitted.
  + We have also added a constraint that there has to be at least one transition from a non-final state to a final state
  + Although not mandatory, we have added a constraint that an initial state should always transition out to a non-initial state.

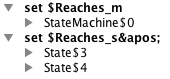
The execution result of the above program is shown in the below screenshot with further explanation about models generated by predicates and :



The **Reaches** predicate attempts to check if in a given model m, a set of states can be reached from m’s initial states. Alloy was able to success generate a model with 5 states and 3 properties that satisfies the above predicate. The visualization of the same if given below:

The complete state machine details are shown in the screenshot below as well in a tree format:

The parameters passed to **Reaches** predicate are:

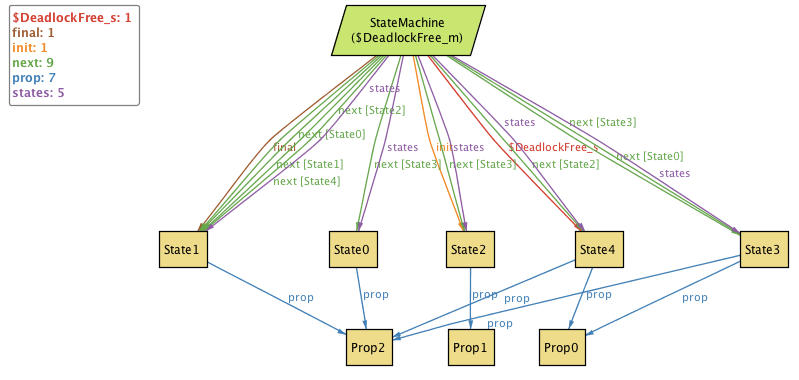


In the above model, Alloy had verified if we could reach states 3 and 4 from the initial state 0. By visual analysis, we can clearly see the following paths in the state machine:

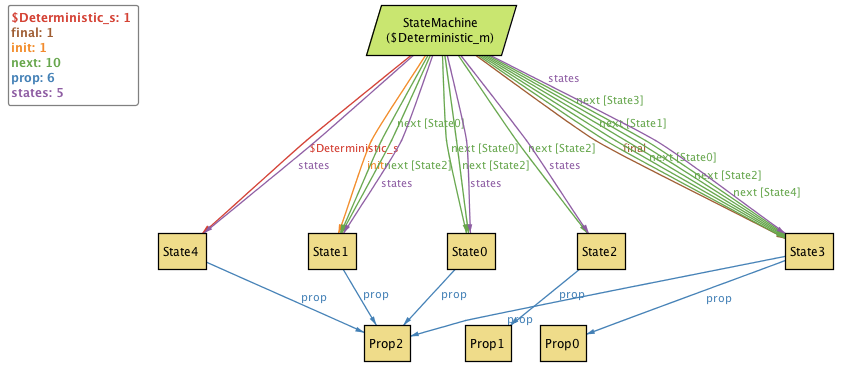
* state 0 -> state 4
* state 0 -> state 1 -> state 3

Thus we are also able to manually verify the predicate on the generated model.

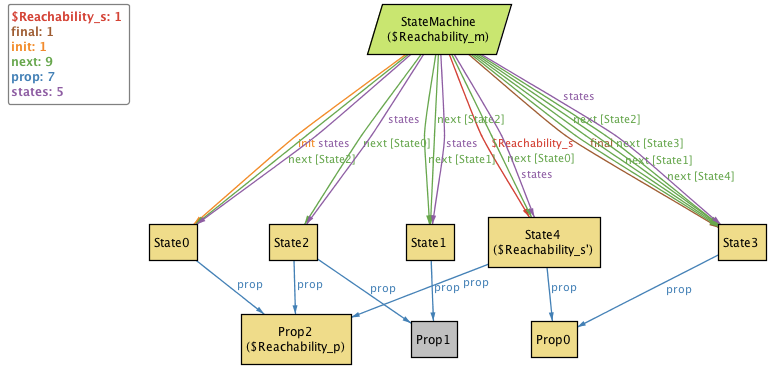
The following models were verified for DeadlockFree, Deterministic, Reachability and Liveness predicates respectively:



**Part I.c(i): Model that satisfied DeadlockFree feature**

****

**Part I.c(ii): Model that satisfied Deterministic feature**

****

**Part I.c(iii): Model that satisfied Reachability feature**

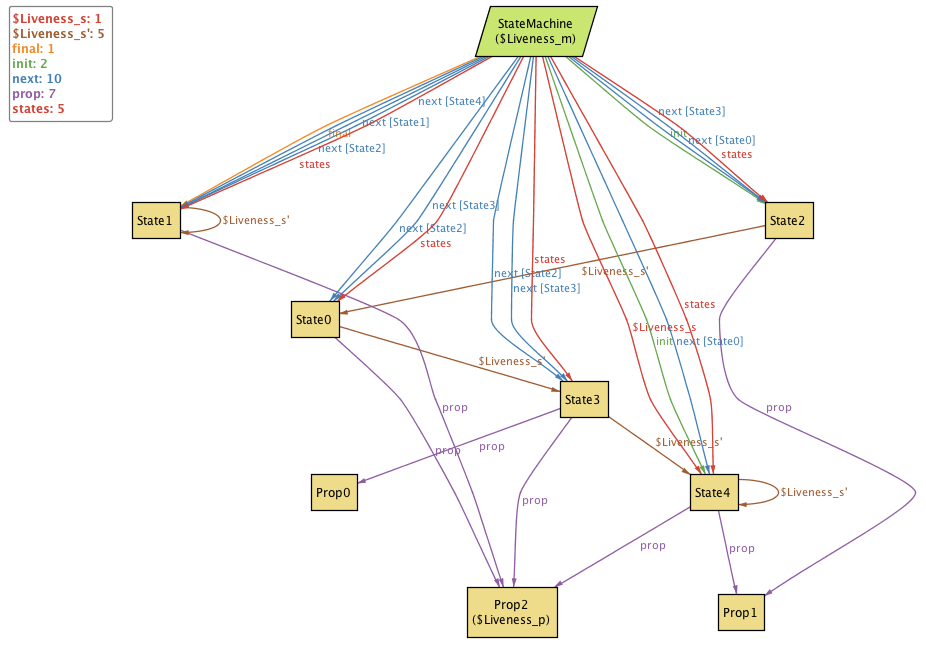
**Part I.c(iv): Model that satisfied Liveness feature**

With the above predicates in place and models that could be generated to satisfy them, we checked the following two assertions:

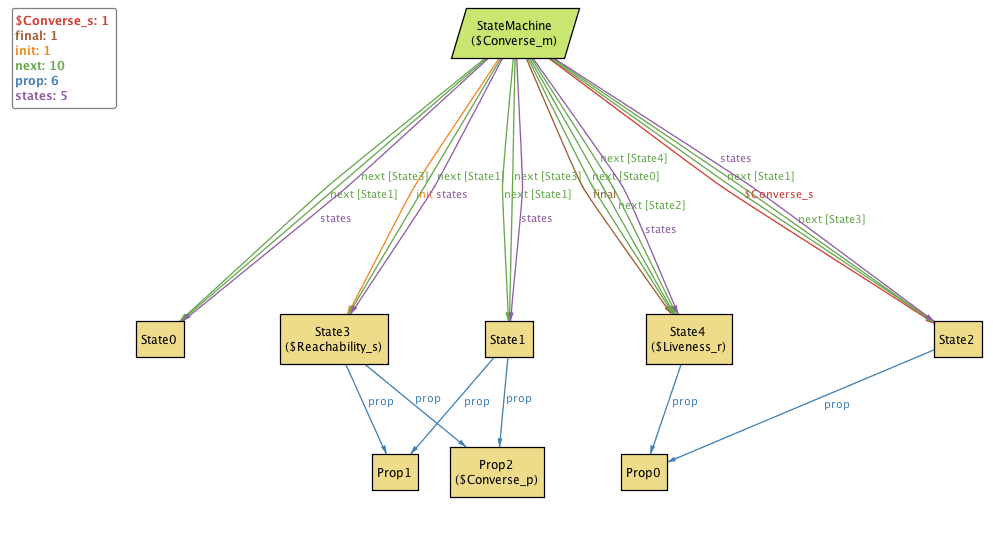
1. Liveness implies Reachability
2. Reachability implies Liveness

From the definition of Liveness, it becomes clear that if a model has Liveness feature, no matter which state it has reached from initial state, we could always find a path to another state that has the property p set to true. This also means that if a model satisfies Liveness, it is guaranteed to find some path from initial states to a state that has property p. This is what Reachability feature require. We could also verify from the Alloy runtime output that Liveness -> Reachability is true.

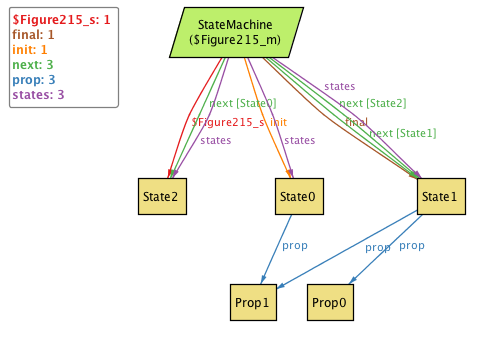
On the other hand, just because we are able to reach a state from initial state that has property p does not mean that we can reach from any state to that state. This means that Reachability does not guarantee Liveness or in other words Reachability does not imply Liveness. This was also shown by the alloy test results as it was able to generate a counter example to prove it.

****

**Part I.e: Model generated by Alloy as a counter example to prove that Reachability does not implies Liveness**

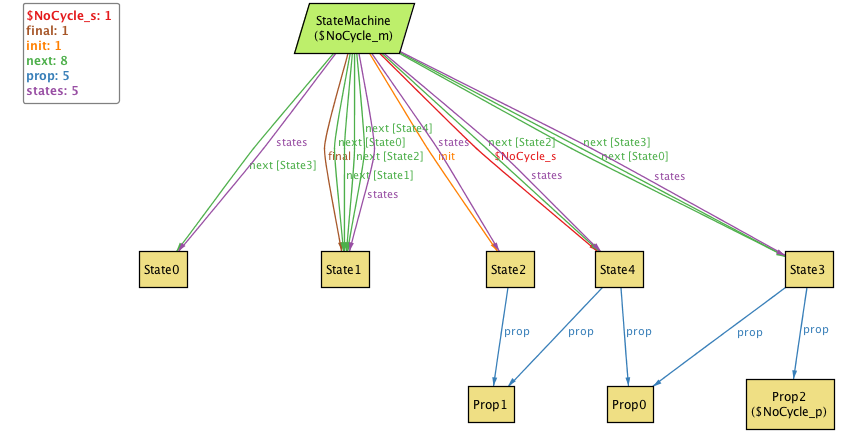
We can also manually verify the same by see that although Prop0, Prop1 and Prop2 are reachable from initial states via states 4, 2, 3 and 1, we can see that Liveness of Prop1 or Prop2 are not there since once we reach state 4 (final state), we can not go to any other states. Since Prop1 and Prop2 are properties of states 1 and 3, we can not satisfy Liveness feature.

Next we have written the Figure215 predicate to generate a model to satisfy the statement of caption given for Figure 2.15. The model generated by Alloy for the same is given below:

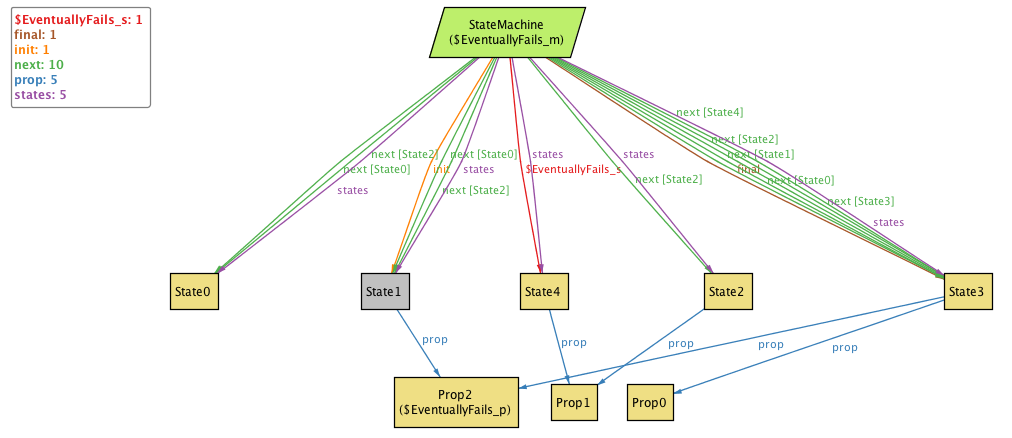


As can be verified in the above move, the only deadlock we have is in the final state ‘State 1’. We can also see that all states have different sets of properties.

Next we define the predicates NoCycle and EventuallyFails features and were successfully verified for their satisfiability using models displayed below:



**Part II.(i) Model that satisfies NoCycle feature**

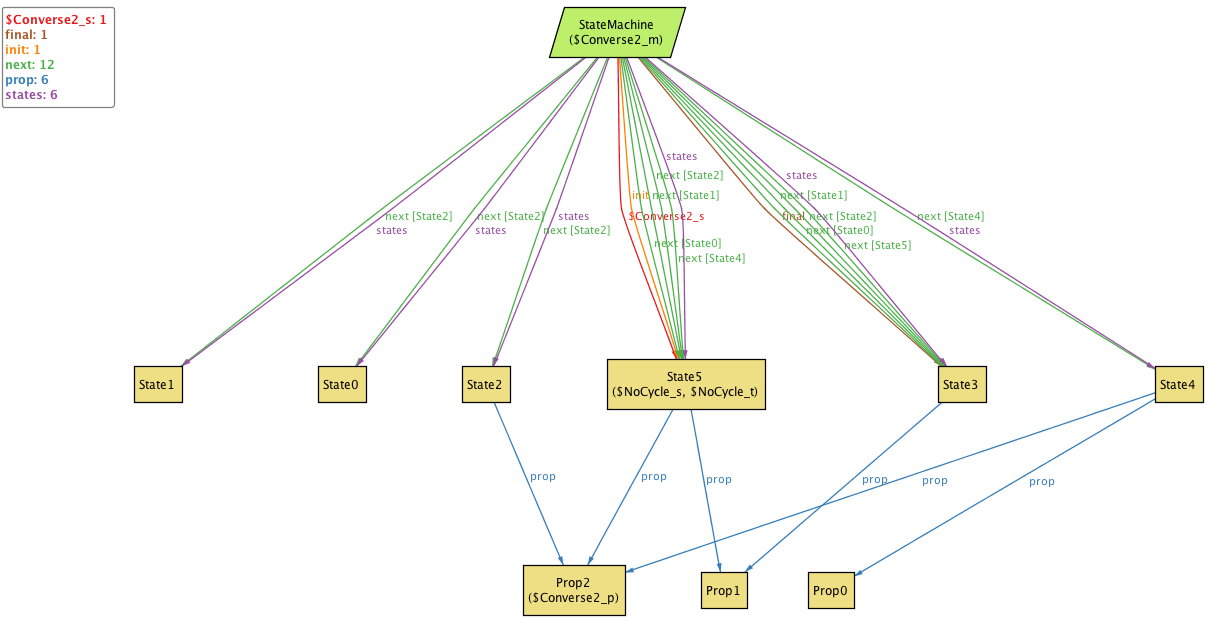


**Part II.(ii) Model that satisfies EventuallyFails feature**

Next we have checked the following assertions for their validity:

* 1. Implies2: NoCycle implies EventuallyFails
  2. Converse2: EventuallyFails implies NoCycle

Results of the evaluation of the above assertions prove that NoCycle -> EventuallyFails but its converse is not true. The counter example provided by Alloy for the second assertion is given below:



**Part II. Model that proves that EventuallyFails feature does not imply NoCycle feature**

From the definition of EventuallyFails can only guarantee that there are no cycles within the path under consideration. But, for NoCycle feature to be present, the EventuallyFails feature need to be satisfied along all paths where p is present.

If we look at the above model for example, we can see that in the case of Prop2, we get the path State 5 -> State 3 where Prop2 eventually becomes false and then remains so forever since State 3 is a final state. But, although such is the case, we can see that we can see that there exists a cycle in the state machine State 4 -> State 4 -> State 4 …. that has Prop2 set to true which dissatisfies the NoCycle feature.