Finite State Machine , Case study of Air conditioning system

Technical Report · April 2015

DOI: 10.13140/RG.2.1.3096.7129

1 author:

Nasreen Ahmad
De Montfort University
17 PUBLICATIONS **0** CITATIONS

Formal Methods Engineering
Nasreen Iqbal
De Montfort University

EX: 2.C13. DESCRIBE THE FINITE STATE MACHINE

The finite state machine is an approach to the mathematical model of computation, exploited the design of computer programing and sequential logic circuits.

Definition: A computational model consists of a set of states, start states, alphabet and transaction, that draw input symbols and current status to a next state. The functional transaction links to the start state and traversing to the states with the input string and producing the output. The machine can have only one state at a time which is represented as current state and travels to other states based on trigger event called transactions.

That means each state has it to trigger condition to act. Such as the following below figure the state0 is the start state and q2 is the final state. Then there is transaction link to start.

Fig. 33: sample of Finite state machine having start state as start and q2 has end state.

The FSM model arises naturally from physical settings in which information-denoting signals are processed. Physical reality dictates that such systems are finite.

Only a finite number of operations may be performed in a finite amount of time. Such systems are necessarily discrete. The problems are quite naturally decomposed into sequences of steps – hence our model is sequential. We require that our machine was not subject to uncertainty, hence its behavior is deterministic.

There are two finite state machine models:

Mealy model – in which outputs occur during transitions.

Moore model – outputs are produced upon arrival in a new state.

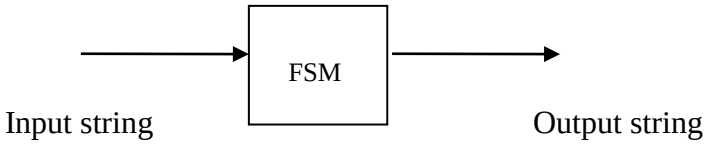


Fig. 34: Finite State statistics

Two types of Finite Automation States

- ***Deterministic Finite State –DFA***

DFA, the set of unique states and single transaction reading to a non empty string that accept language {Λ}. A machine which accepts/rejects input strings and only produce the unique computation.

A deterministic automation state is a quintuple **M**=(***K, ∑, δ, s, F***) where,

K is finite states,

∑ is an alphabet,

s ∈ K is the initial state

F ⊆ K is the set of final states, and

δ, the transaction function, is function from K x ∑ to K.

- ***Nondeterministic Finite State-NFA***

The none deterministic state automation leading to two conditions. The first is, two transactions from the same state that read the same symbol and the second condition is, transaction that read the empty string from the input called nondeterministic states.

The **definition** is:

$M=(K,\sum,\Delta,s,F)$, where

K is a finite set of states

$\sum$ is an alphabet,

$s \in K$ is the initial state,

$F \subseteq K$ is the set of final states, and

Δ, the transaction relation, is a subset of $K \times (\sum \cup \{e\}) \times K$.

- ***Difference between DFA and NFA***

1. NFA does not require to go to unique next state, may not go any state from the current state on reading an input symbol or it may select one of several states. DFA is required a unique state and single state to drive to the next state.
2. The transition function is also called a *next state function*. Contrasting, DFAs an NFA derived into one of the states given by $\delta(s, i)$ if it receives the input symbol in *the states*. Which one of the states in $\delta(s, i)$ to select is controlled nondeterministically.
3. But, any DFA is also a NFA

Please refer the above exercise have nondeterministic examples.

EX: 2.C23. TRANSORMATION OF ONE FORMAL DESCRIPTION TO ANOTHETR ONE

a). Examine the machine and deduct its activity

Solution: the machine accepts the empty word, any word that ends with <R>, any word $a1\ldots..a_k \in \{0,1,2\}^*$ such that

$\sum_{i=1}^{K} \alpha_1 \equiv 0 \pmod 3$, and any concatenation of these words. More specification, if

$L_1 = \{<R>,0,1,2^*) <R>$

In other words, three state machine has *3 R* and four symbol inputs

$\sum = \{R,0,1,2\}$

This machine will count numeric symbols. The three states of the automaton communicated to the three numbers *0,1,2*, the sum could ever be. Every time the automaton receives the <R> symbol, it resets the count to 0 while moving to state q0.

Let w be the string = R012

Then model accept *w* as a sequence of state

Which satisfies which satisfies the three conditions

L(Model)= {w l sum of the symble in w =0, Accept that <R> resets the count to 0}

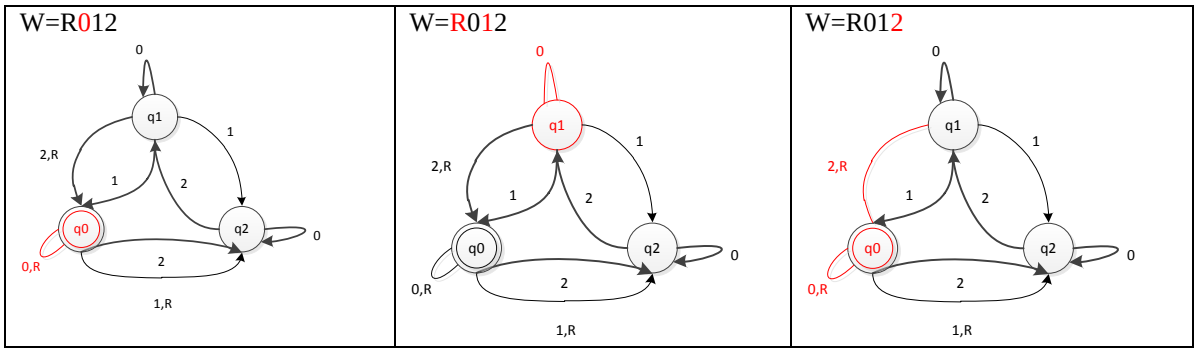


Fig. 35 Formal Description of four symbol input alphabet {R,0,1,2}

- Start qo
- Read 0, follow the transaction from q0 to q0
- Read 1, follow the transaction from q0 to q1
- Read 2, follow the transaction from q1 to q0
- q0=0

i	*wi*	*Current state*	*Next state*
0	*R*	-	*q0*
1	*0*	*q0*	*q0*
2	*1*	*q0*	*q1*
3	*2*	*q0*	*q2*

Definition of finite automaton

5-tuple= $(Q, \Sigma, \delta, q_0, F)$, where

Q= finite set called state
Σ= finite set called the alphabet
δ= Q X Σ-> q, a transition function
q0 E q is the start state
F C q is the accepted set of states.

- For each i ≥ 0 let ai be the language of all strings where the sum of the numbers making up the input is a multiple of i, except that the sum is reset to 0 whenever a symbol <R> appears
- For each ai we construct a finite automaton bi recognizing ai

The machine bi is described formally as follows:

bBi = (qi,_, δi, q0, {q0}) where

- qi = {q0, q1, . . . , qi−1}

- Σ = {0, 1, 2, <R>}

-δi is designed so that for each j, 0 ≤ j ≤ i − 1, if Bi is in the state qj then the running sum is j modulo i. For each qj:

δi(qj , 0) = qj

δi(qj , 1) = qk, k = j + 1 modulo i

δi(qj , 2) = qk, k = j + 2 modulo i

δi(qj , <R>) = q0

b). Describe informally the languages accepted by the FSMs shown below.

Informal theorem = what happened with the BLACk BOX?

b. i). Informally the language accepted by the given FSM's

Must read *a* to reach into the unique final state

Once it reads then it may read *ba =acceptable*

So, the language is ***a(ba)*. Or (ab)*a)***

Informally, this can describe as all strings that begin and end with *a*, where symbol alternate *a*.

b. ii). Informal language

In this machine we have two reachable final state, in this way once the final state arrived then further change would be blocked.

Thus, the informal language is ***aa*b U b=a*b,*** as in simply way.

c). Given the following languages.

c.i). Construct deterministic automata accepting L1

Problem:

Deterministic: for each state q 2 States, Transition maps each a 2 Inputs to exactly one "next" state q0, i.e., Transition is total and deterministic.

L1 = {w ∈ {a, b}* :w has an even number of a's and an odd number of b's}

Solution:

We say: L= { w ∈ {a,b}* : #(a,w) is even, and #(b,w) is odd}.

The function # (σ, x) =number of occurrence of symbol σ in the string x. In this way we need to observe the string, in order to reject and acceptance of occurrence. Therefore,

Observe the equality = (even) of the number of a's

Monitor the equality =(odd) of number of b's

The possible states or properties are 2x2=4:

- (e0/d0): x read, where #(a,x)=even and #(b,x)= even
- (e1/d0): x read, where #(a,x)=even and #(b,x) =odd
- (e1,d1): x read, where #(a,x)=odd and #(b,x)= even
- (e0/d1): x read, where #(a,x)=odd and #(b,x)=odd

So, #(σ,e)=0, and 0 is even, thus:

The *start* state is (e0,d0) and the final state is (e0/d1).

Formally, I designed the above algorithm as, formed using states that monitor whether odd or even numbers of string have been reading of every moving state. The set of automation is:

{add/even, even/odd, odd/odd, even/even}

Where each is read as given the equivalence of the number of a's followed by the equivalence of the number of b's read. The table of values for the transaction function is:

	a	b
e0/d0	e1/d0	e0/d1
e1/d0	e0/d0	e1/d1
e1/d1	e0/d1	e1/d0
e0/d1	e0/d0	e1/d1

For the acceptance states we take {e0/d0 }. Initially no a's and no b's has been read, thus we have all even number and can read even/even as initial state. Thus, a suitable state diagram is:

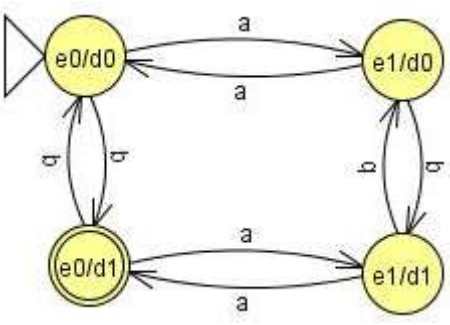


Fig. 36 FA for a's even and b's odd

If we start in the state, even/even and input the string about. Then we pass through the following states:

$$e0,d0 \xrightarrow{a} e1,d0 \xrightarrow{b} e1,d1 \xrightarrow{a} e0,d1$$

However, the effect of starting at *e0/d0* and input *aba* as string that end-up in the state *e0/d1*. So, the sequences of *a's* and *b's* are accepted when the number of *a's* is even and the number of *b's* is odd. However, the language is:

$$\{x \in (a+b)^* : |x|_a \text{ and } |x|_b \text{ are even/odd}\}$$

c.ii). Give the regular expression for the language L2.

Problem:

L2={w ∈ {a, b}* : w has both *aba* and *bab* as substrings}

Definition: *Regular expressions define languages via a* ***semantic interpretation function, we****'ll call L:*

1. *$L(\varnothing) = \varnothing$ and $L(a) = \{a\}$ for each $a \in \Sigma$*
2. *If αα , b are regular expressions, then*
   $L(\alpha\beta) = L(\alpha)–L(\beta)$
   = all strings that can be formed by concatenating to some string from L(⟨) some string from $L(\beta)$.
   Note that if either α or β is $\varnothing$, then its language is $\varnothing$, so there is nothing to concatenate and the result is $\varnothing$.
3. *If α , β are regular expressions, then $L(\alpha \cup \beta) = L(\alpha) * L(\beta)$*
4. *3. If a is a regular expression, then L(a*) = L(a)**
5. *5. L((α)) = L(α)*

A language is ***regular*** *if and only if it can be described by a regular expression. A regular expression is always finite, but it may describe a (countably) infinite language.*

Solution:

In this solution, finding two occurrences in the strings *aba* and *bab*, having two refinements, either might occur first. Therefore, any of them (*aba or bab*) may read first, in this way the L definition doesn't require the two substrings of *aba* and *bab* to be non overlaying.

So, the regular expression is:

(a ∪ b)*aba (a∪b)*bab (a∪b)* ∪ (a∪b)* bab (a∪b)* aba (a∪b)*

d). State Diagram for nondeterministic FSM accepting of the following below language

d. i). (ba)(ab)* + bb**

Solution:

A non-deterministic finite automation allow us to try all the possible choices in parallel.

The easiest way to do this is to make a 2 state FSA for aa* and a 4 state one for *(ba)*(ab)**, then make a 7th state, the start state, that nondeterministically deductions which class an input string will fall into. The given string inputs are palindrome.

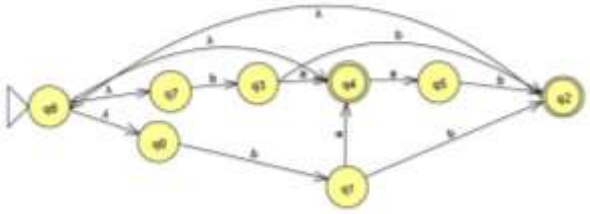
Simplify: (ba)*(ab)* ∪ bb*

*/ (L_1*L_2)+L3* = (L_1 ∗ L_2)L3* /*

=(ba)∪(ab)+bb*

/ Union is /

=(ba, ab, bb) or (bb,ba,ab)



NFA

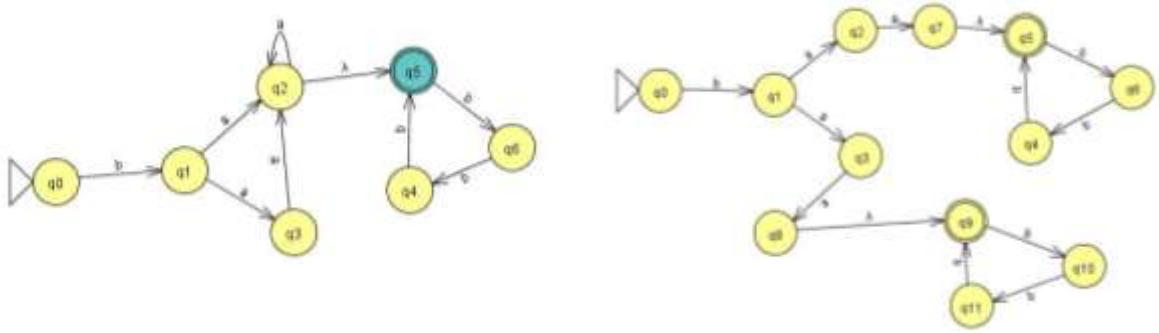


Fig. 37: 3 possible NFSM for the (ba)(ab)* + bb**

W	***Input***	***acceptance***
ba	q0, q1, q3, q4	acceptable
ab	q0,q4q5,q6	acceptable
qb	q0,q2,q7,q6	acceptable

Table Text Size

Input	Result
ba	Accept
ab	Accept
baab	Accept
bb	Accept
baabbb	Accept

*d. ii). (a*b*a*)**

First, we simplify: (a*b*a*) *

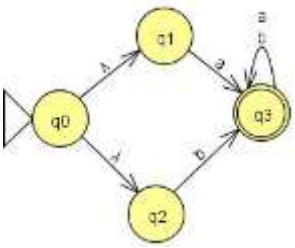
*/ (L_1*L_2*L_3*)* = ($L_1 * L_2 * L_3$)* /*

=(a∪b ∪ a)*

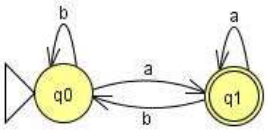
/ Union is idempotent /

=(a ∪ b)*

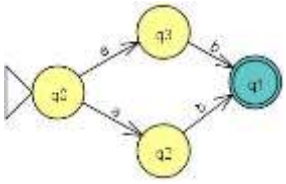
There is a simple 2 state NDFSM, which is the empty string and all strings.



NFA



DFA



NFA-Parlel

*Fig. 38: 2 possible NFSM for the (a*b*a*)**

W	***Input***	***acceptance***
a	q0, q1, q3	acceptable
b	q0, q2, q3	Acceptable
ab	Q0,q1,q3,q2,q3	accepetable

Table Text Size

Input	Result
a	Accept
b	Accept
ab	Accept

e). Construct the deterministic FSM

We can build the following machine by making the path from 1 to 2 to 3 for the *ba* option. The rest is for the second choice. The nondeterministic machine is generally built when we use the simple approach.

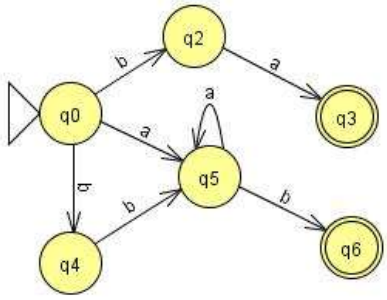


Fig. 39: FSM equivalent to NFSM

In this case, we could simplify machine if we wanted to and get rid of state 4 by adding a transition on *b* from 2 to 5.

Exercise 3

C31. DESCRIBE THE MEANING OF A FINITE STATE MACHINE

a) Case study of Air conditioning system using a state chart

This air conditioner control has four switches and one temperature sensor input. The AC system adjusts the temperature and accordingly turn on and off the compressor.

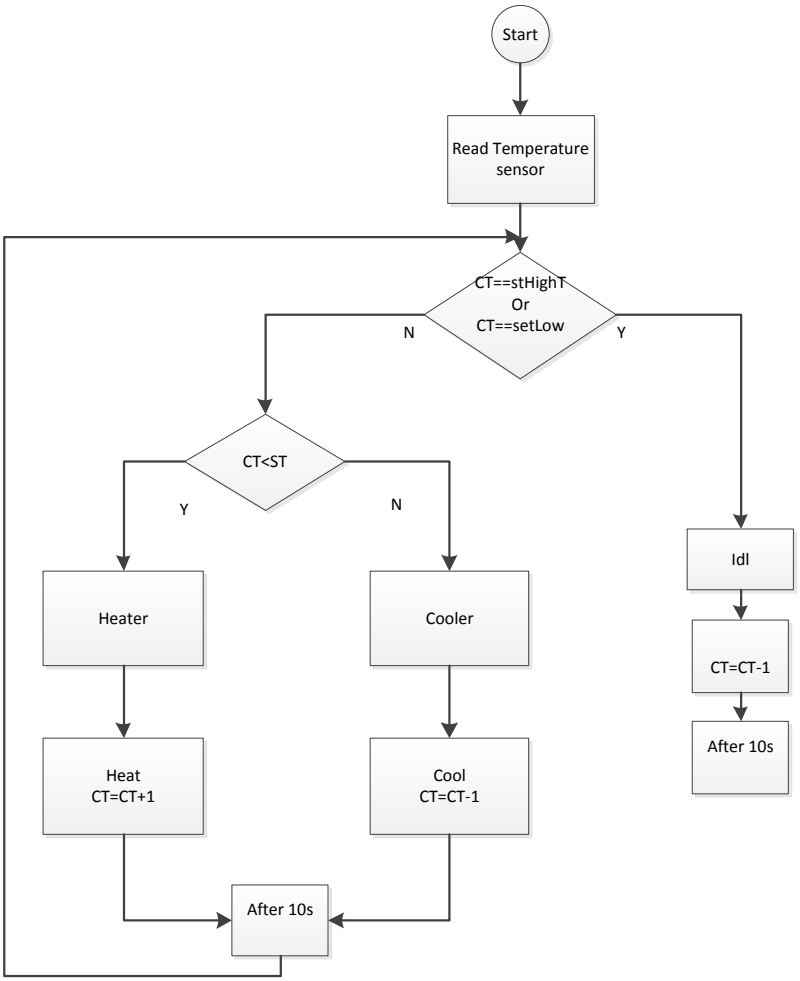
The function:

- Temperature Up: increase the temperature by 1 degree
- Temperature down: decrease the temperature by 1 degree
- Mode: mode will start and stop the AC.
- History Of state: will check the last history and when next time AC will turn on, the current temperature will assign the historical value to start.

The AC has an input state for the temperature from the sensor, and the sensor will calculate the actual temperature using current room temperature. If the current temperature (CT) > set temperature (ST), then the AC compressor will turn on, similarly if the CT< ST then the compressor will turn on.

The system has only one ST threshold, so the system have history, where TH_threshold =ST+2, TL_threshold =ST-2. If the CT>ST, the AC will turn on, until CT<TL; when CT<ST the AC will not turn on until CT>TH. The AC will go for ideal state if the temperature is too low or is too high.

Flow Chart:



The states and triggers:

	State	External changes	Nest State	Trigger	Description
1	On_Off	On_Off	Sensor	On_Off	Set point temperature

2	Sensor	Calculate the temperature:
		If CurrTemp>setTemp and CT< !=0 then
		{
		goto cooler
		}else
		{
		goto heater
		}
			Cooler / Heater	Entry	Read temperature and calculate the current temperature. If currTemp < setTemp then AC movers to Heater else moves to the cooler

3	Cooler	Open Close cool pipe Dipaly(currTemp)	Cool	Every	Get the instruction from the Sensor and activate cool compressor for 10 the second and diplay currTemp.
4	Heater	Open Close heat pipe Dipaly(currTemp)	Heat	Every	Get the instruction from the sensor and activate the heater for 10 second and diplay currTemp.
5	Heat	CurrTemp = CurrTemp +1	Operator	Every	Loop current temperature for every 1 second currentTemp=+1
6	Cool	CurrTemp = CurrTemp - 1	Operator	Every	Loop current temperature for every 1 second currentTemp=-1
7	Standby	Every 3 second currTemp=currTemp+1	Operator	Every	AC will go for ideal if the temperature reached to very low =5 and very high=25
8	History	setTemp=currTemp	On_Off	Hstry	

List of Time Out:

- Every 10s the cool and heat will turn off and AC will move to sensor back
- If Current temperature reaches too cool or too hot, then the AC will set time out for 10s and go for ideal state. In this state the temperature will be down.

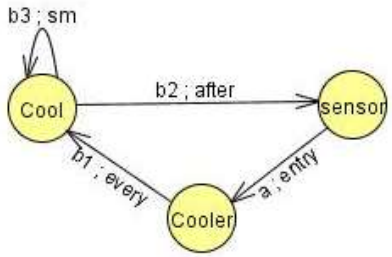
Scheduled Transaction Action:

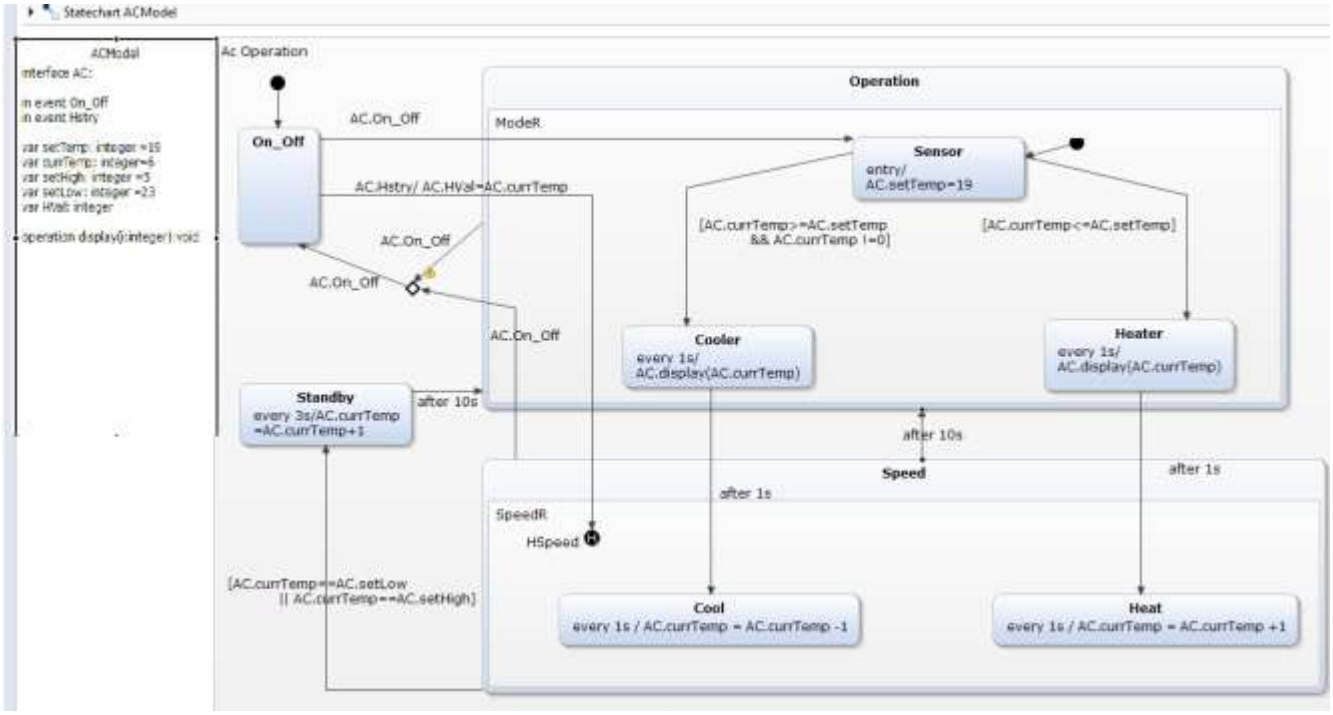
- The sensor will set the initial set the initial temperature in the room for the AC in order to manipulate.
- If current temperature is greater than the set temperature, then AC will trigger and open cooler pipe else AC will open heater pipe.
- Compressor allows 10 second for the hot/cool pips to be open.
- Cool or heat pipe will increase / decrease the temperature by 1 in every 1 second
- If cold reached to equal to very low temperature, then the AC will switch to standby state and pose AC for 10 seconds, where the current temperature would be down by 1 for every 3 second. This process would have acted as normal condition where room temperature would be down in case the AC would be in a pose condition.
- If heat reached to equal to very high temperature, then the AC will switch to standby state and pose AC for 10 seconds, where the current temperature would be down by 1 for every 3 second. This process would have acted as normal condition where room temperature would be down in case the AC would be in a pose condition.
- After 10 second the AC will turn back to the sensor and recalculate the temperature.
- The history would be maintained and transfer the current temperature into HVal by triggering the Hstry event.
- The AC will turn off and on by triggering the On_Off event at any stage.

The Model:

Generation of events e / a_1; . . . ; a_n

Time-out events team (e(w1),10/every) , team (e(w2),1/after), team (e(w3),1/every),

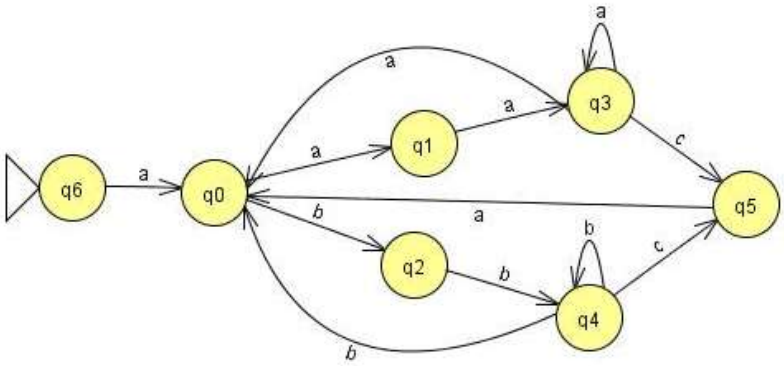




What happened:

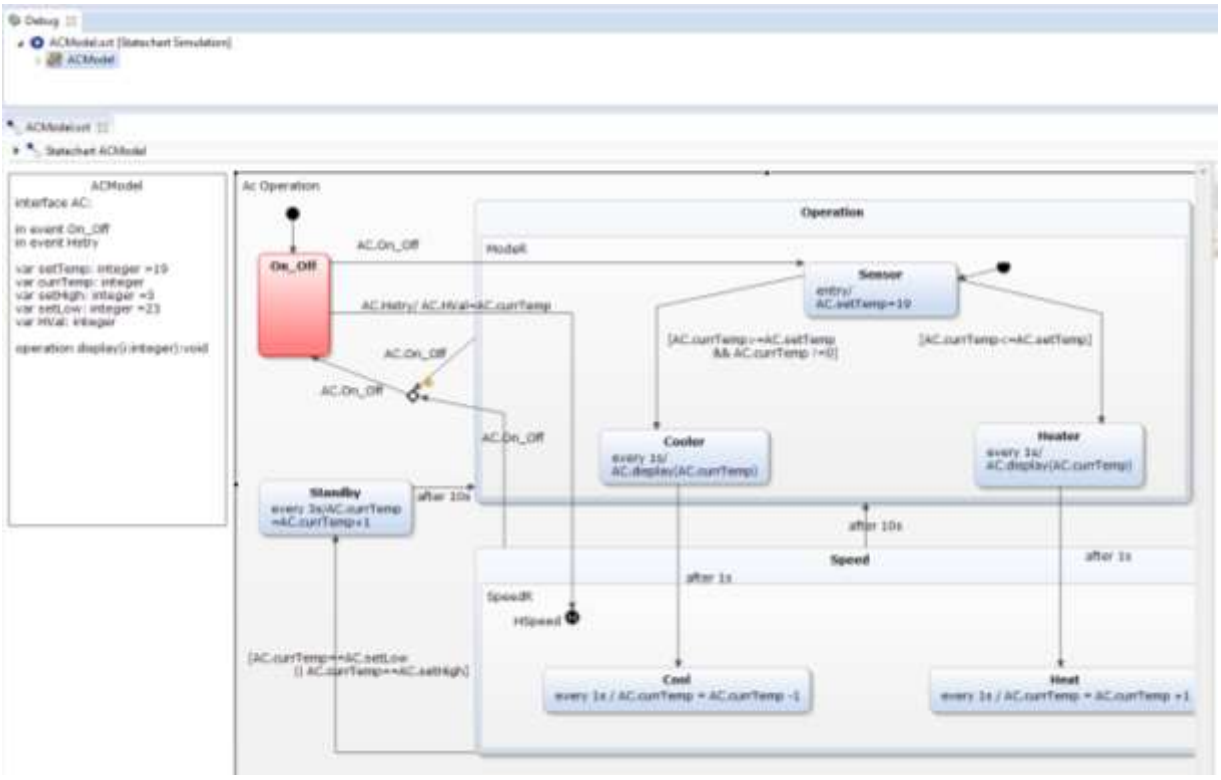
When the event On_Off button pressed, the AC switches to the sensor. Once the sensor state activated the *entry* event generated and set the initial set_temp to some value. However, the sensor then calculates the current temperature by analysis the set temperature and assigned the task to the cooler or heater accordingly. Once the AC switches to cooling or heat state, the operation *display ()* activates and display the current temperature. The cooler state opens the *cool* pipe after 1 second by triggering the *after*. The speed region will triggers the *after* event and switches to the operation region. Here sensor again activated and compare the current temperature to the set temperature. The AC may reach to the state *standby* if the room temperature reached to the lowest or highest, which is set in the initial state. The shallow history maintained by observing current temperature and activates while pressing Hstry event.

(¬ on_off (sensor),(cooler V heater), (cooler ^ cool), (heater ^ heat)

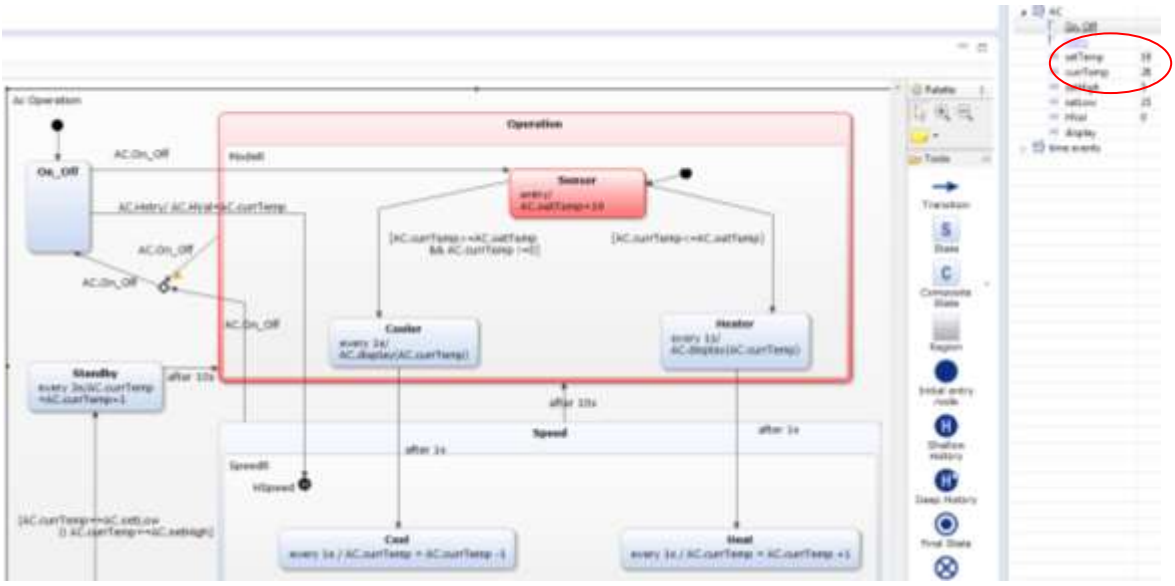


Tests and Simulation:

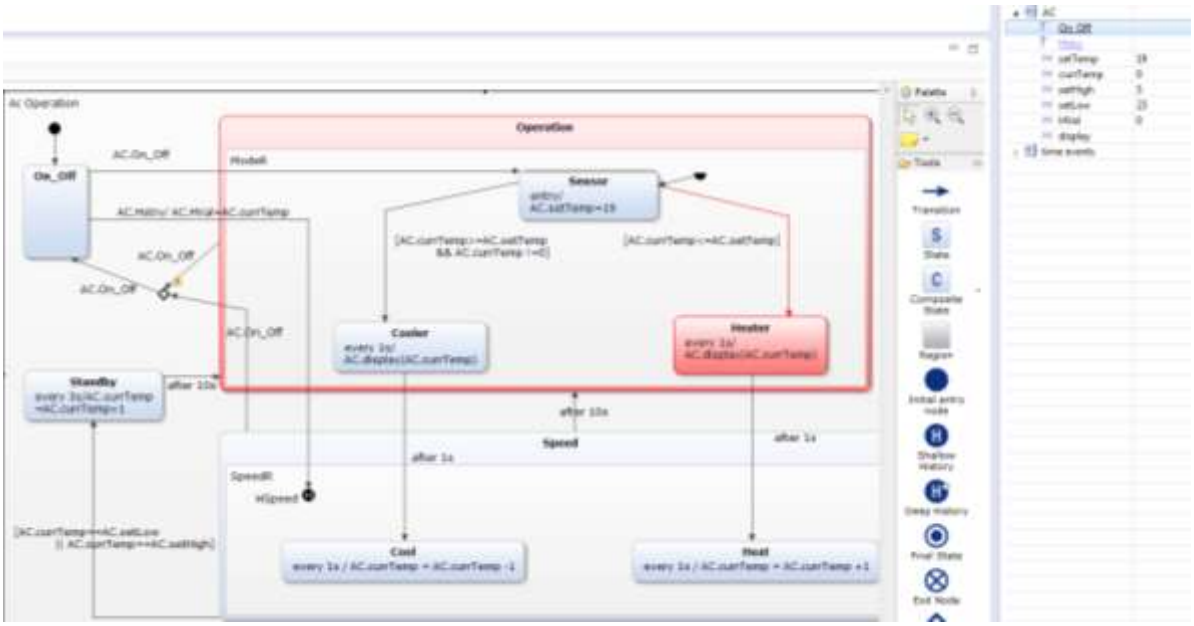
The interfaces has been set The statechart begin with the On_Of State where the initial variable set in the following below screen. The setHigh which is coolest temperature has been assigned 5 and the setLow integer assigned the 23 values as hottest temperature. The setTemp variable has been set as 19 for the room initially, that can be assigned at the initial state of the interface or can be assigned in the sensor state as an entry. However the On_Off event fired while press the button to start.



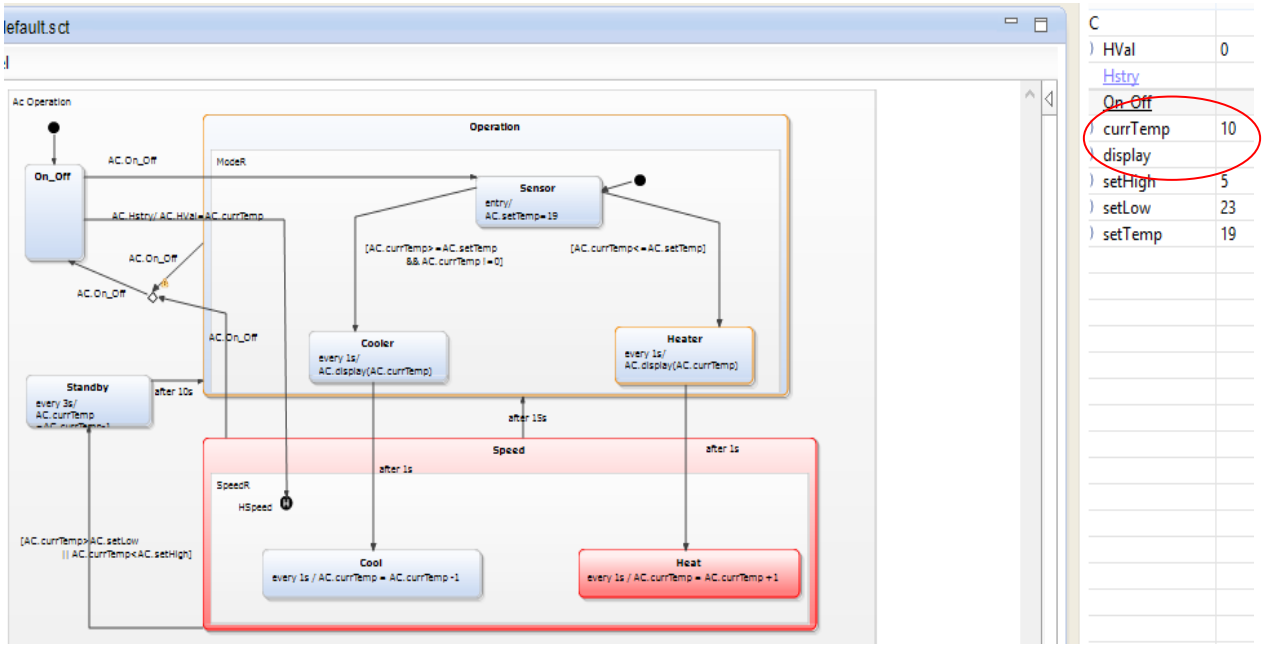
The AC_On event switched to the sensor state and enter the value 19 for the setTemp variable, that leads to begin the AC system. This setTemp variable set the required temperature and maintained the current temperature accordingly.



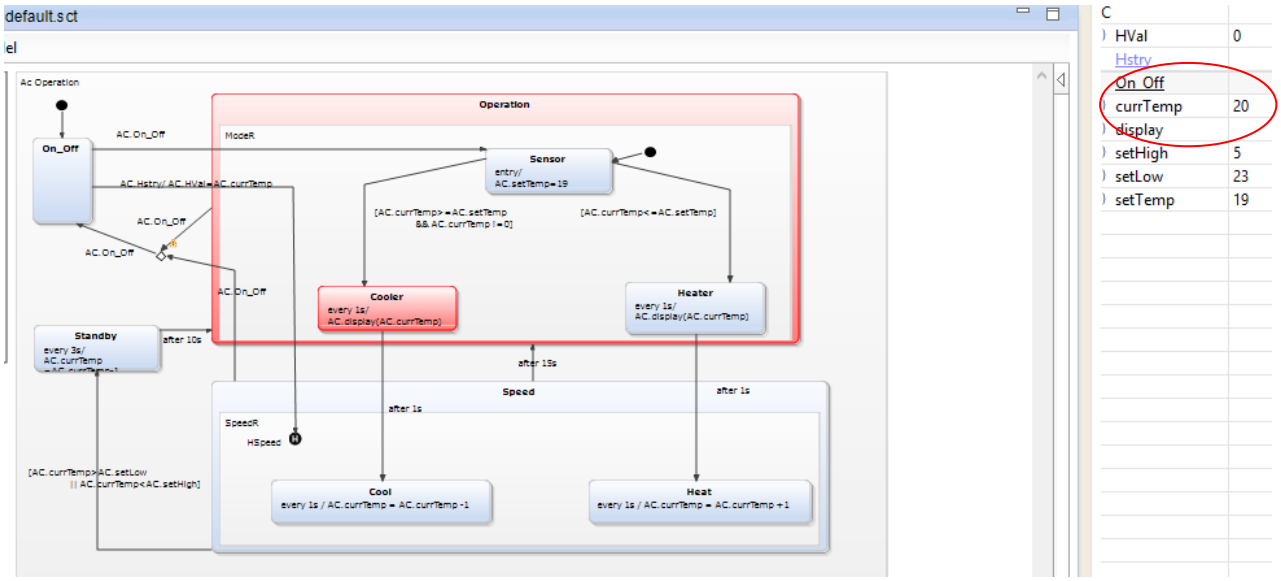
In the following below figure the variables are set as current temperature =6 and current temperature =19. However, a sensor checks the curTemp<=setTemp. Here, the currTemp (6) < setTemp i.e. 6<19 which means the room temperature is 6 and required heat. Therefore, the transaction begins to Heater state and call the *display* () operation to display the current temperature.



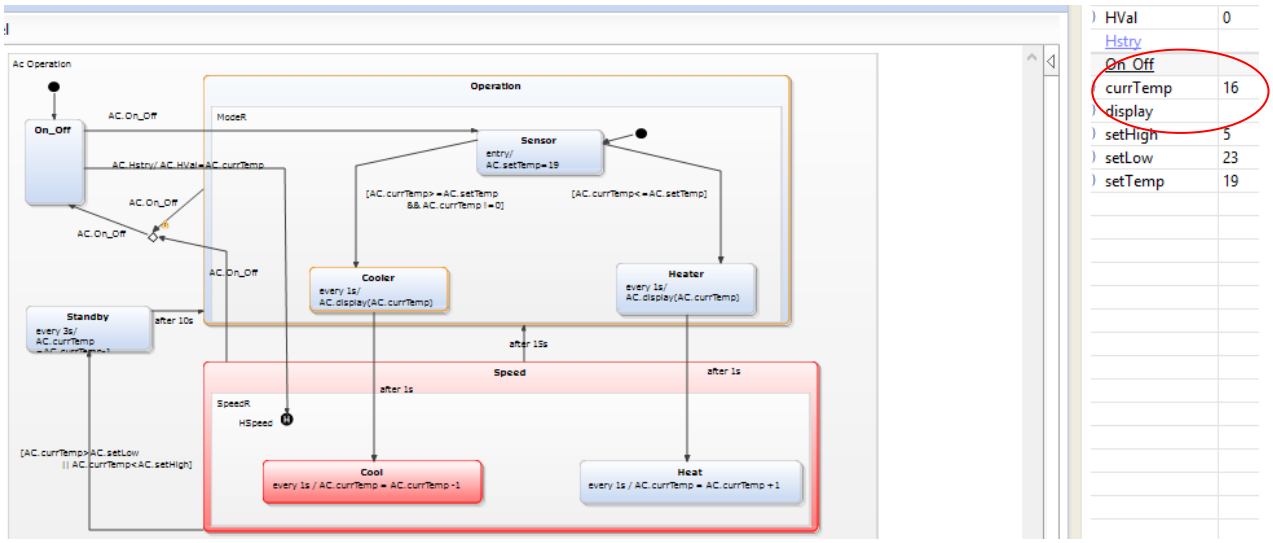
Thus, Heater state received instructions from sensor to open Heat pipe, so the transaction moved to Heat state and open the pipe for 10 seconds. The Heat state activated and increase the temperature by 1 in each second.



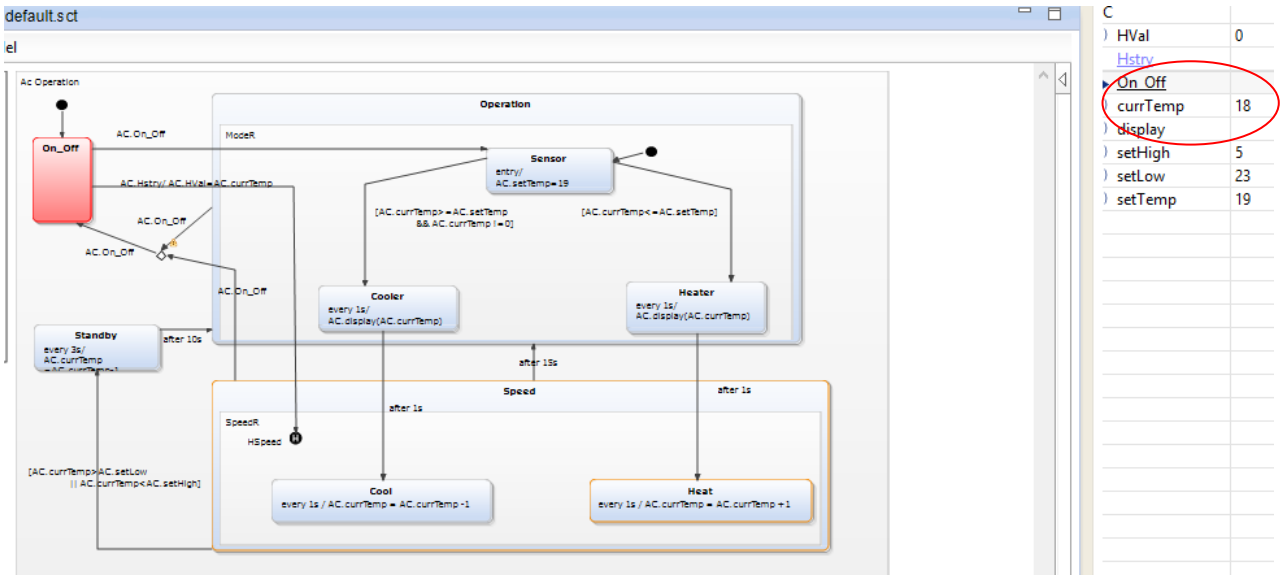
Finally, *After* trigger fired in about 10 second and AC switched from Speed region to operation region and again sensor check the status of two variables, where currTemp now increased to 20. Thus, the sensor assigned a task to the Cooler this time where current status of two variables are setTemp < currTemp.



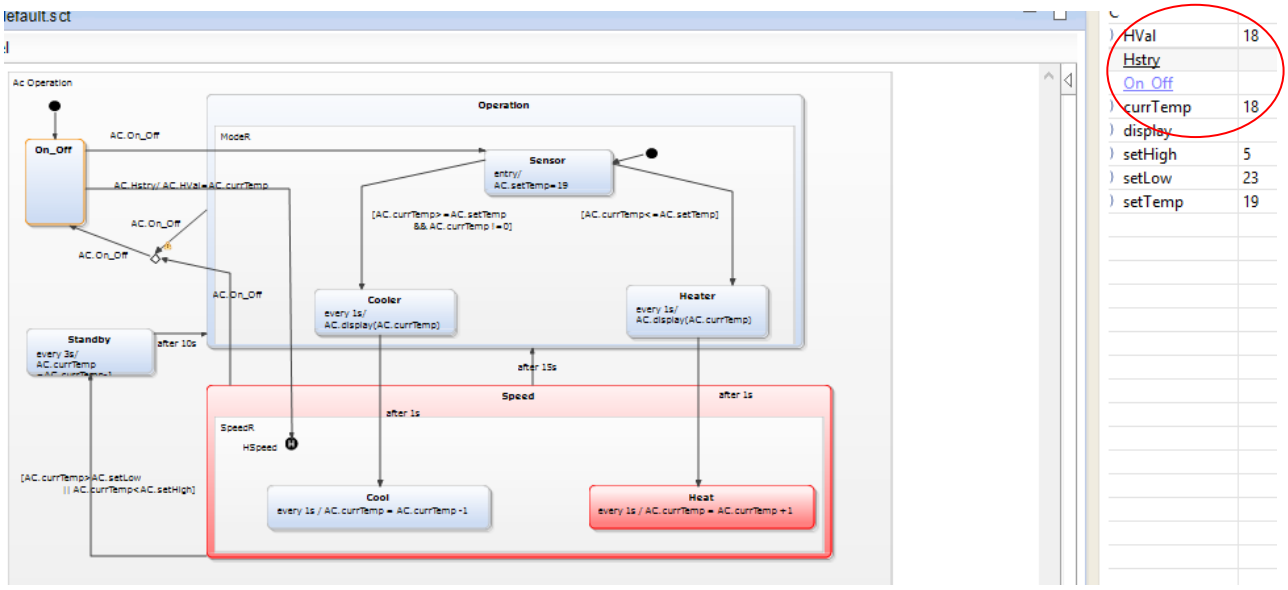
After 1 second the AC switched to the Speed region. Now the current temperature recorded 16, so, the temperature will be decreased by 1 in each second i.e. currTemp=-1. The following below figure is the next step.



The shallow history worked when we press the stop button and then press the Hstry event to recall the previous room temperature. In the following below simulation the currTemp =18 when off button pressed. The machine stopped now.



The Hstry event pressed to recall the previous history. The following below screen has temperature still 18. That means shallow history will recall the previous temperature and restart the function from the same point.



fine states

Describe states

Raw transaction

Define transition trigger

Dfine guard condition

References:

[1] Harry R. Lewis, Christos H. Papadimitriou, "Elements of the theory of computation, second edition"

[2] course slides.

[3] S. Michael, "Introduction to the Theory of Computation", https://books.google.com.bh/books?id=1aMKAAAAQBAJ&pg=PA39&lpg=PA39&dq=The+following+finite+state+machine+has+a+four-symbol+input+alphabet:+%7BReset,+0,+1,+2%7D.&source=bl&ots=iDSTjKOfbN&sig=uowAYll5BvOPtLkQ_xtSLohhP2k&hl=en&sa=X&ei=S44FVc_oIsi6ygPe6YCACg&ved=0CCEQ6AEwAQ#v=onepage&q=The%20following%20finite%20state%20machine%20has%20a%20four-symbol%20input%20alphabet%3A%20%7BReset%2C%200%2C%201%2C%202%7D.&f=false

[4] http://www.ics.uci.edu/~irani/f13-6B/FiniteAutomataRosen.pdf-VEND

[5] http://www3.cs.stonybrook.edu/~cse350/slides/automata2.pdf

[6] http://xlinux.nist.gov/dads/HTML/finiteStateMachine.html

[7] http://faculty.simpson.edu/lydia.sinapova/www/cmsc365/LN365_Lewis/L07-Problems.htm#pr3d

[8] W. Zhang, X.L. Zhang, "Design and Implementation of automatic vending machine, Based on the short massage payment" International Conference on Information and Communication, technology in Electrical Sciences, Neijiang, Sichuan, China.pp.978-981, 2010.