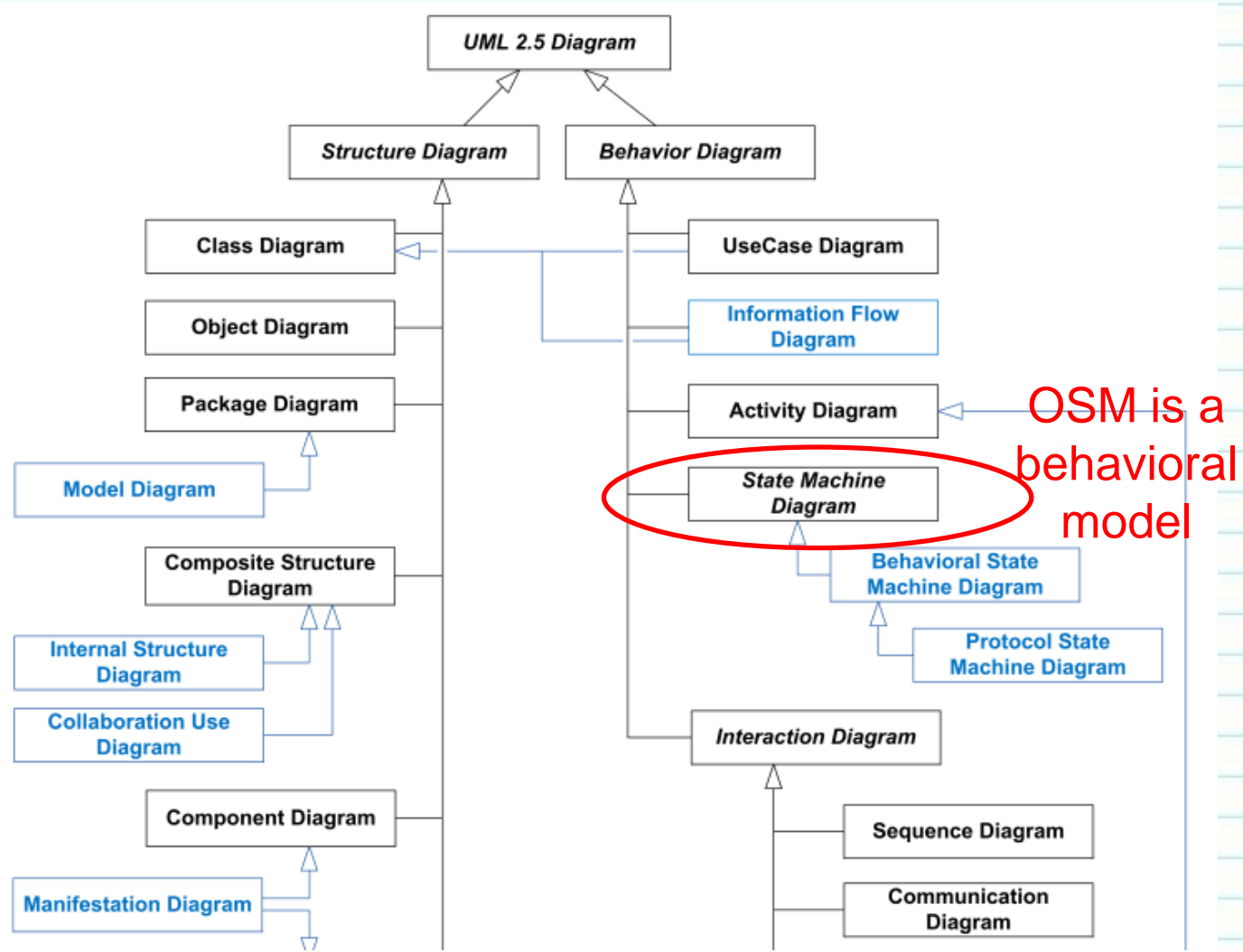


Chapter 13 – Object State Modeling

Dr John H Robb, PMP, SEMC

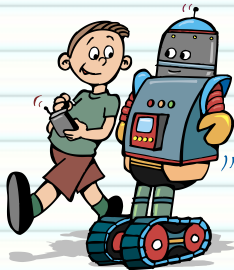
UT Arlington Computer Science and Engineering

OSM in the Methodology Context



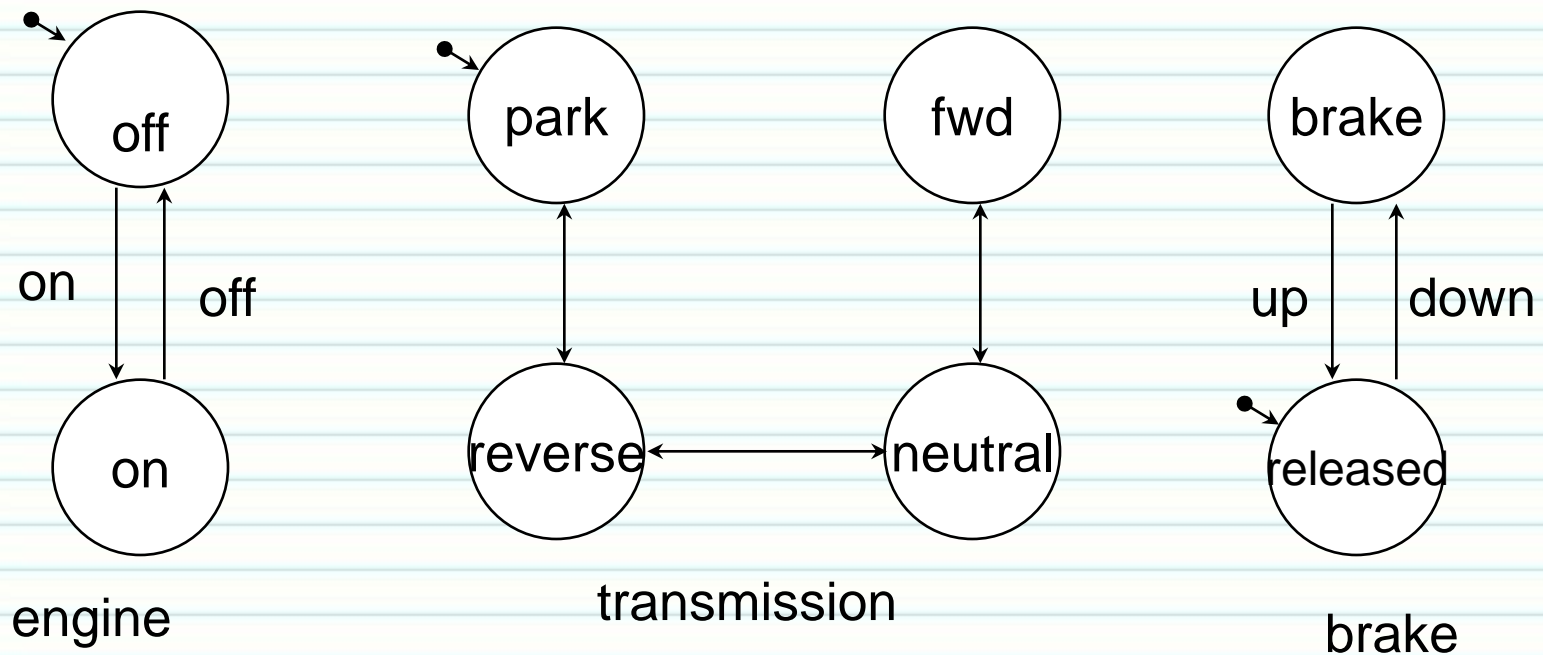
Object State Modeling

- Identification, modeling, analysis, design, and specification of state-dependent reactions of objects to external stimuli.
 - What are the external stimuli of interest?
 - What are the states of an object?
 - How does one characterize the states to determine whether an object is in a certain state?
 - How does one identify and represent the states of a complex object?
 - How does one identify and specify the state-dependent reactions of an object to external stimuli?
 - How does one check for desired properties of a state behavioral model?



State Behavior Modeling

- State dependent behavior is common in software systems.



What is the state dependent behavior of a car?

Basic Definitions

- An *event* is some happening of interest or a request to a subsystem, object, or component.
- A *state* is a named abstraction of a subsystem/ object condition or situation that is entered or exited due to the occurrence of an event.

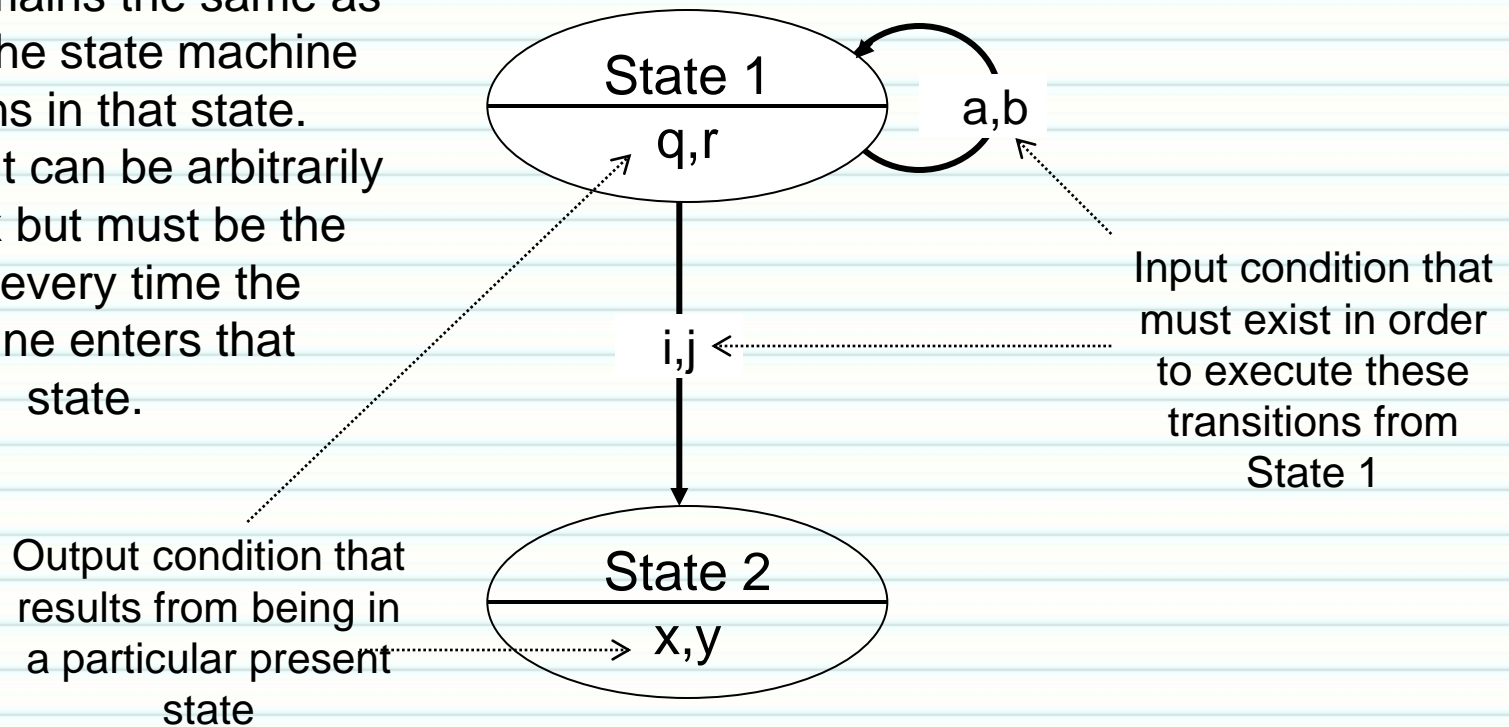
Moore and Mealy Machines

- There are two types of graphical depictions of state machines, they differ in the way that outputs are produced.
- **Moore Machine:**
 - Outputs are independent of the inputs-outputs are effectively produced from within the state of the state machine.
- **Mealy Machine:**
 - Outputs can be determined by the present state alone, or by the present state and the present inputs-outputs are produced as the machine makes a transition from one state to another.

Moore Machine Diagrams

The Moore State Machine output is shown inside the state bubble, because the output remains the same as long as the state machine remains in that state.

The output can be arbitrarily complex but must be the same every time the machine enters that state.



Mealy Machine Diagrams

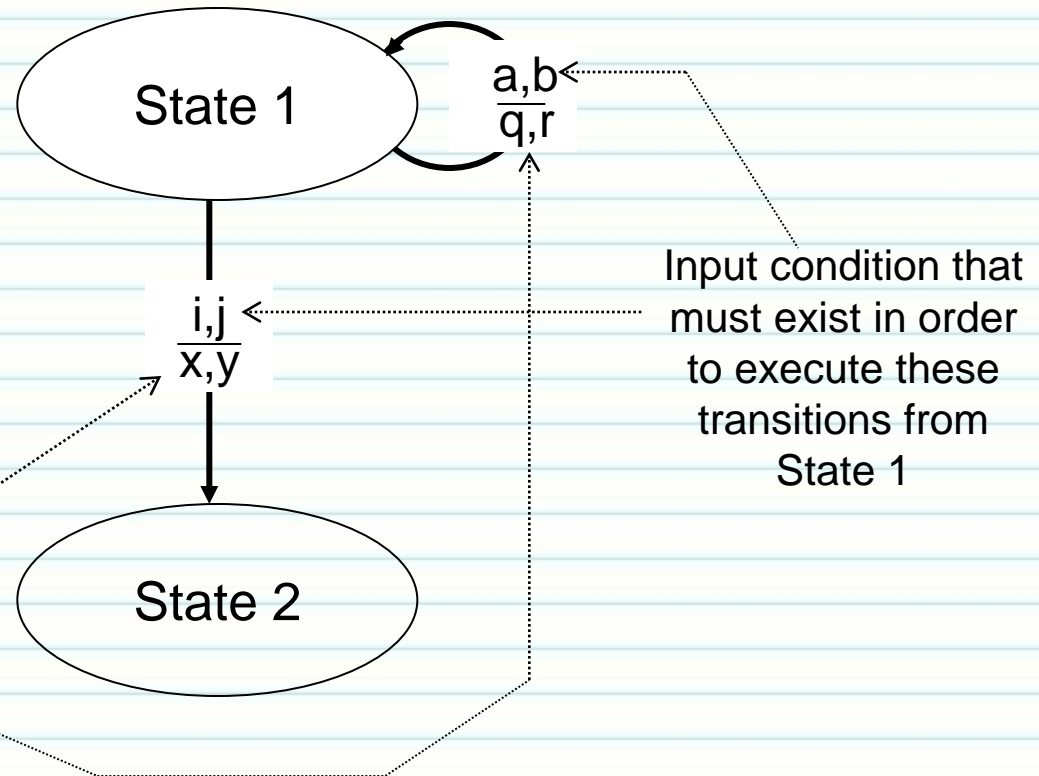
The Mealy State Machine generates outputs based on:

- ♦ The Present State, and
- ♦ The Inputs to the M/c.

So, it is capable of generating many different patterns of output signals for the same state, depending on the inputs present on the event.

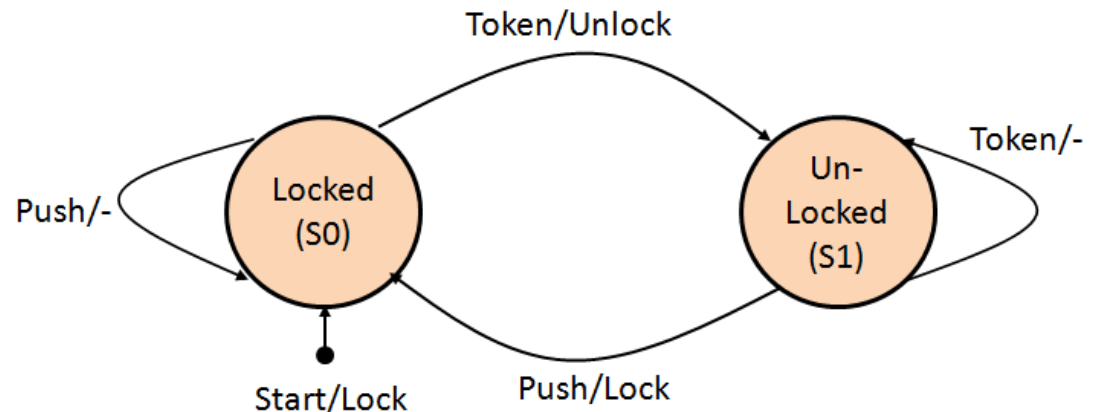
Outputs are shown on transitions since they are determined in the same way as is the next state.

Output condition that results from being in a particular present state



Simple System

- Turnstile system
- A turnstile is used to control access to subways, is a gate with rotating arms (typically 3) near waist height, one across the entryway.
- Initially the arms are locked, preventing entry.
- Depositing a token in a slot on the turnstile unlocks the arms, allowing a single customer to enter. After the customer enters by rotating through the arms, the arms are locked again until another token is inserted.

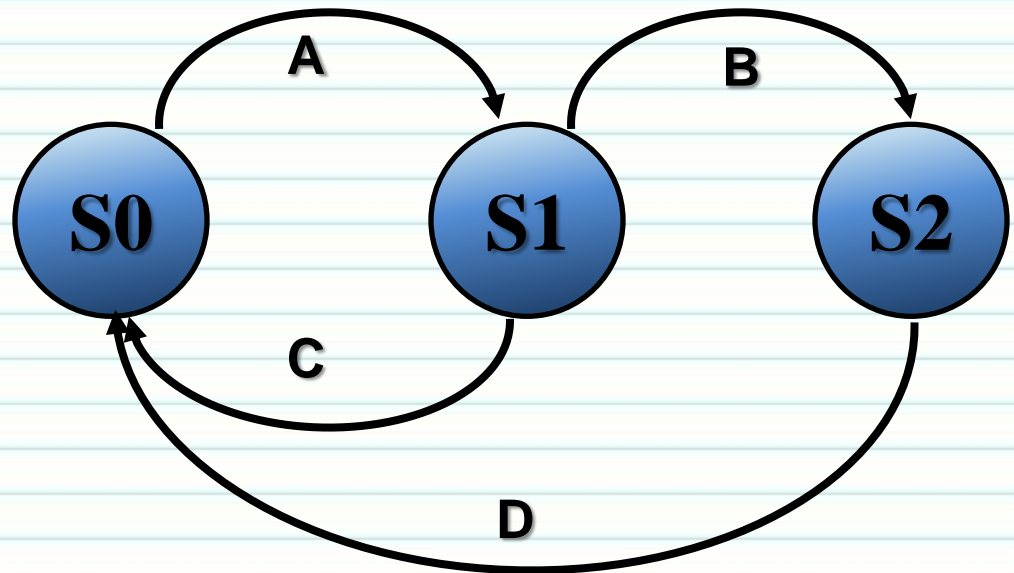


A state machine must have a start state!

Exercises (1)

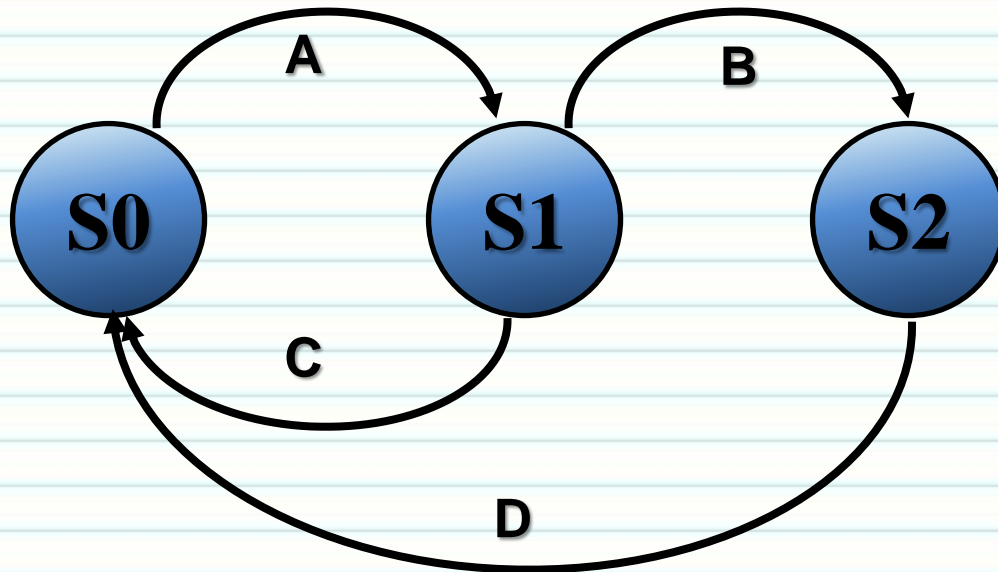
1. What is missing from this state diagram? Simple answer!
2. Given the following state transition diagram – which of the test cases below will cover the following series of state transitions? S1 S0 S1 S2 S0

1. C, A, B, D
2. C, A, C, D
3. C, A, D, C
4. C, A, C, D



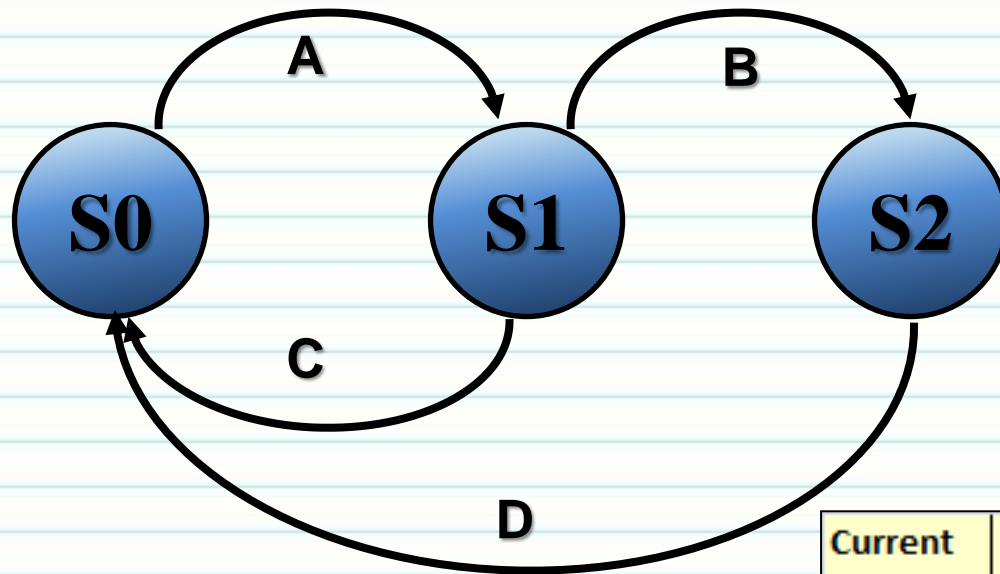
Exercises (2)

- Develop the state transition table for the following state diagram



Exercises (2)

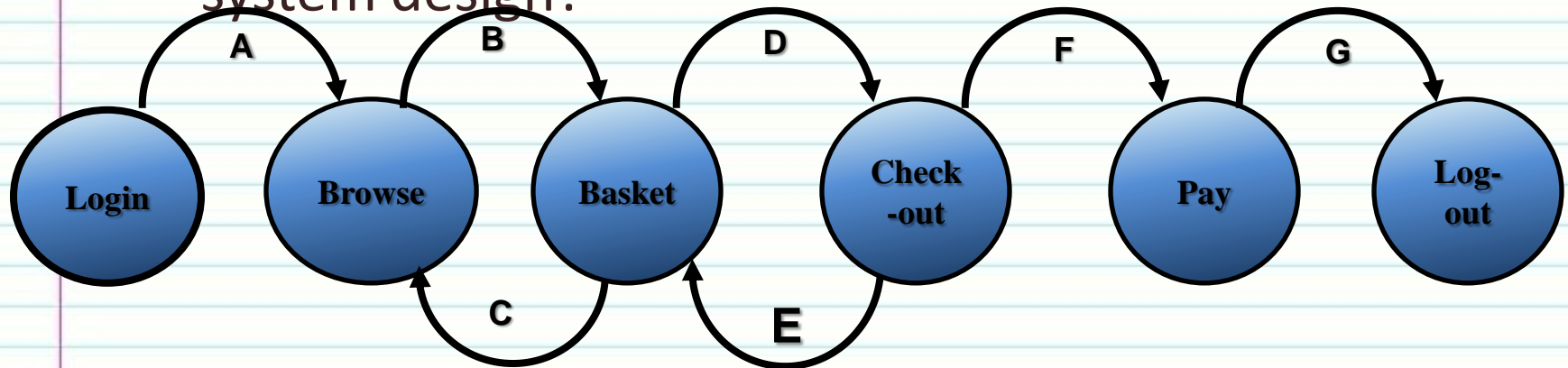
- Develop the state transition table for the following state diagram



Current State	Event	Action	Next State
Start		-	S1
S0	A	-	S1
S1	C	-	S0
	B	-	S2
S2	D	-	S1

Exercises (3)

2. Given the following state transition diagram which of the following series of state transitions contains an INVALID transition which may indicate a fault in the system design?

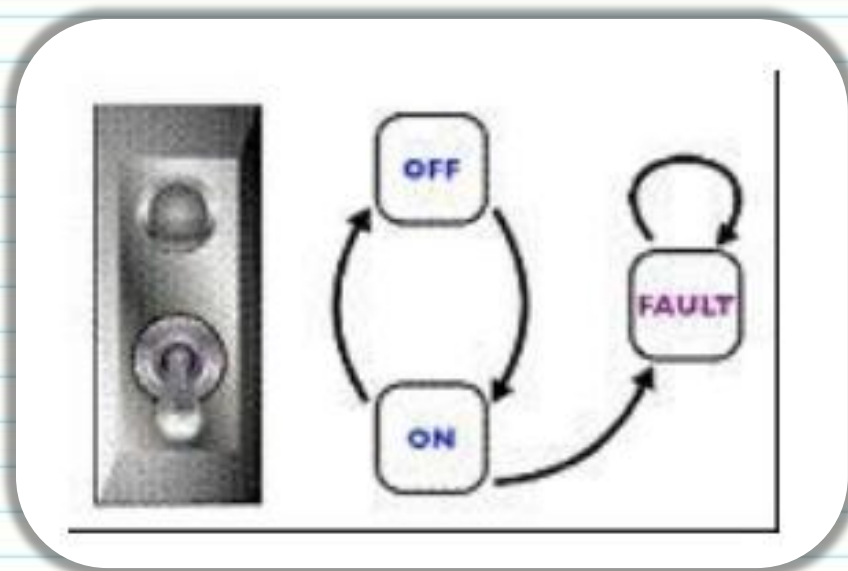


- A.** Login Browse Basket Checkout Basket Checkout Pay Logout
- B.** Login Browse Basket Checkout Pay Logout
- C.** Login Browse Basket Checkout Basket Logout
- D.** Login Browse Basket Browse Basket Checkout Pay Logout

Exercises (4)

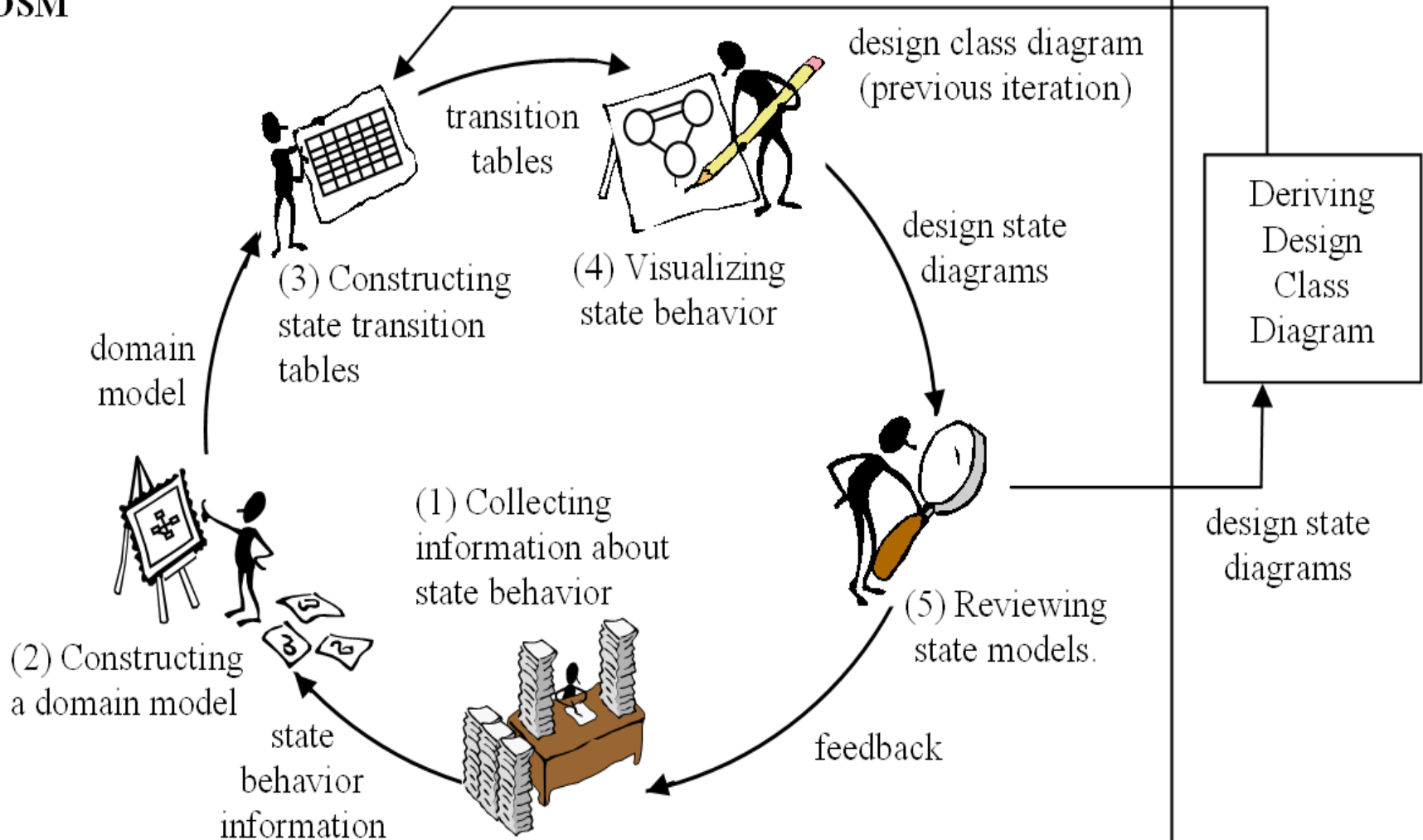
3. Consider the following state transition diagram of a switch. Which of the following represents an invalid state transition?

- a) OFF, ON, Fault
- b) ON, OFF, Fault
- c) Fault, Fault, On

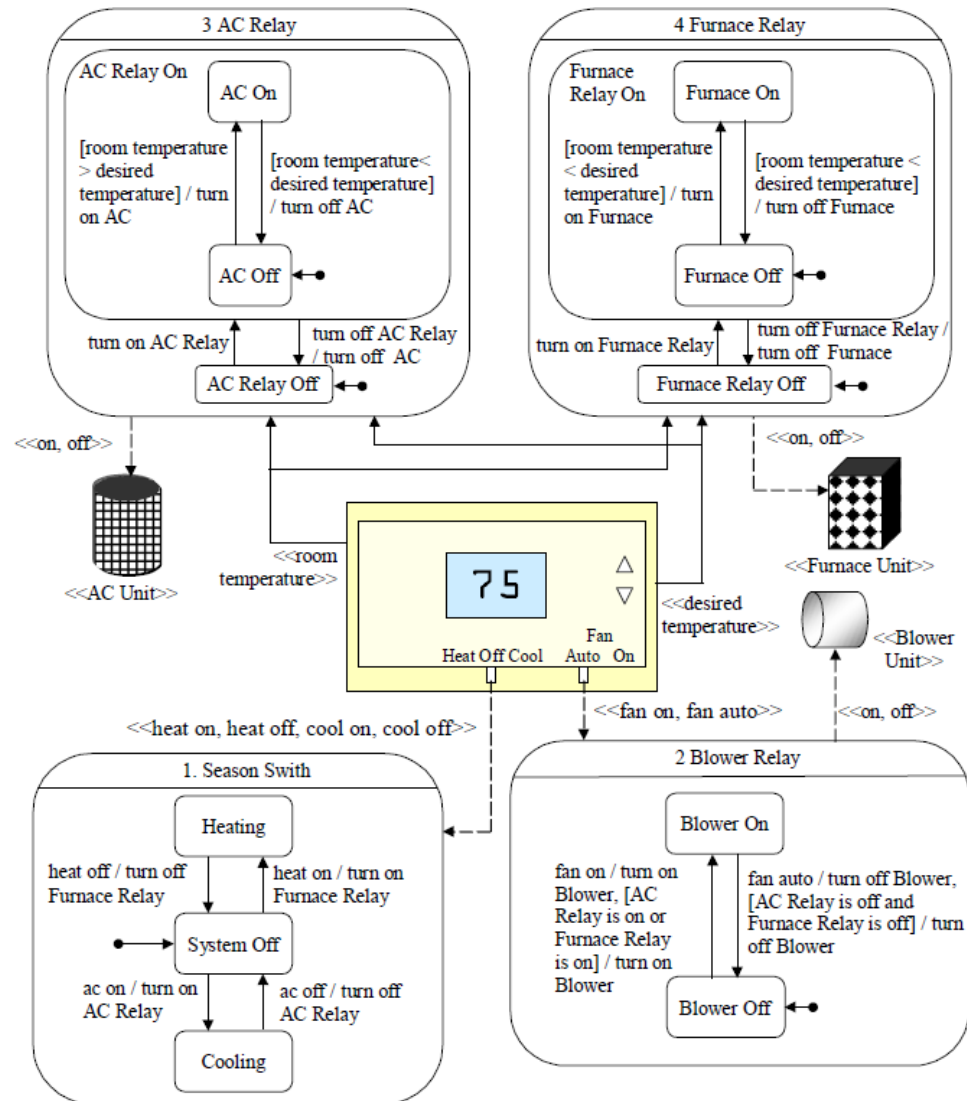


Object State Modeling Steps

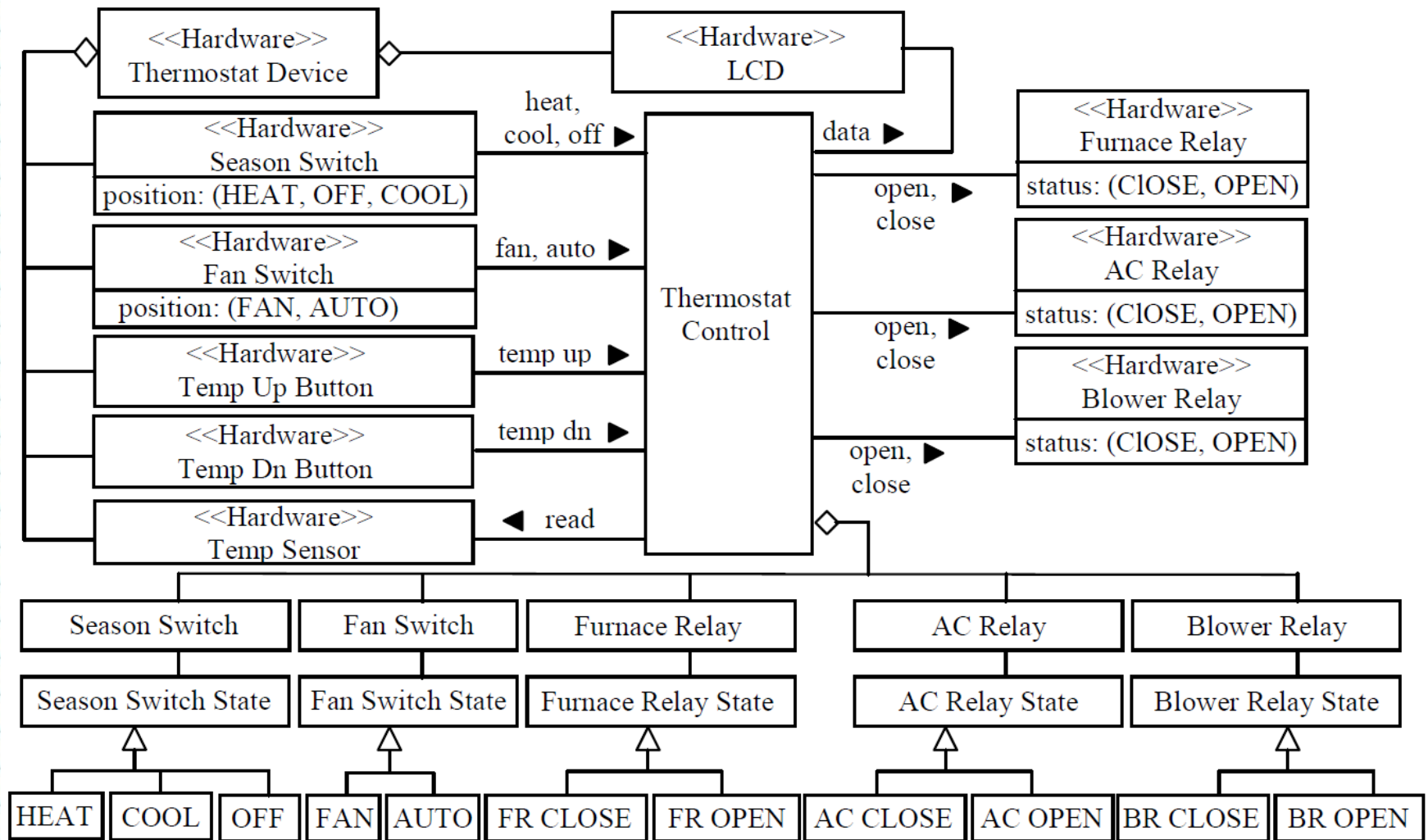
OSM





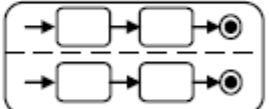
Thermostat System Diagram



Thermostat Domain Model



State Diagram Notations (UML)

	Notion	Notation
Basic State Diagram	Simple state: A state that does not contain other states. A state is a condition or situation of a subsystem / object.	
	Initial state: A pseudo state to start with.	●
	Final state: A state that indicates the completion of the state machine.	⦿
	Transition: A transition from one state to another caused by an event.	→
	Transition label: It indicates that when event e occurs and the guard condition expr is true, then the state transition takes place and the response actions a1 ; a2 are executed.	e [expr] /a1 ; a2
Advanced Features	Composite sequential state (CSS): A state that is composed of one region of states related by state transitions. At most one of the states can be active at any given point in time. Specialization substates and their parent state are visualized with CSS.	
	Composite concurrent state (CCS): A state that is composed of more than one region of states related by state transitions. Two or more states, each from a different region, can be active at the same time. Component substates and their aggregate are visualized with CCS.	
	Shallow history state: It indicates to return to the most recently active substate of its containing state but not the substates of that substate.	⦿
	Deep history state: It indicates to return to the most recent configuration of active substates of its containing state and recursively all of the substates of the containing state.	⦿*

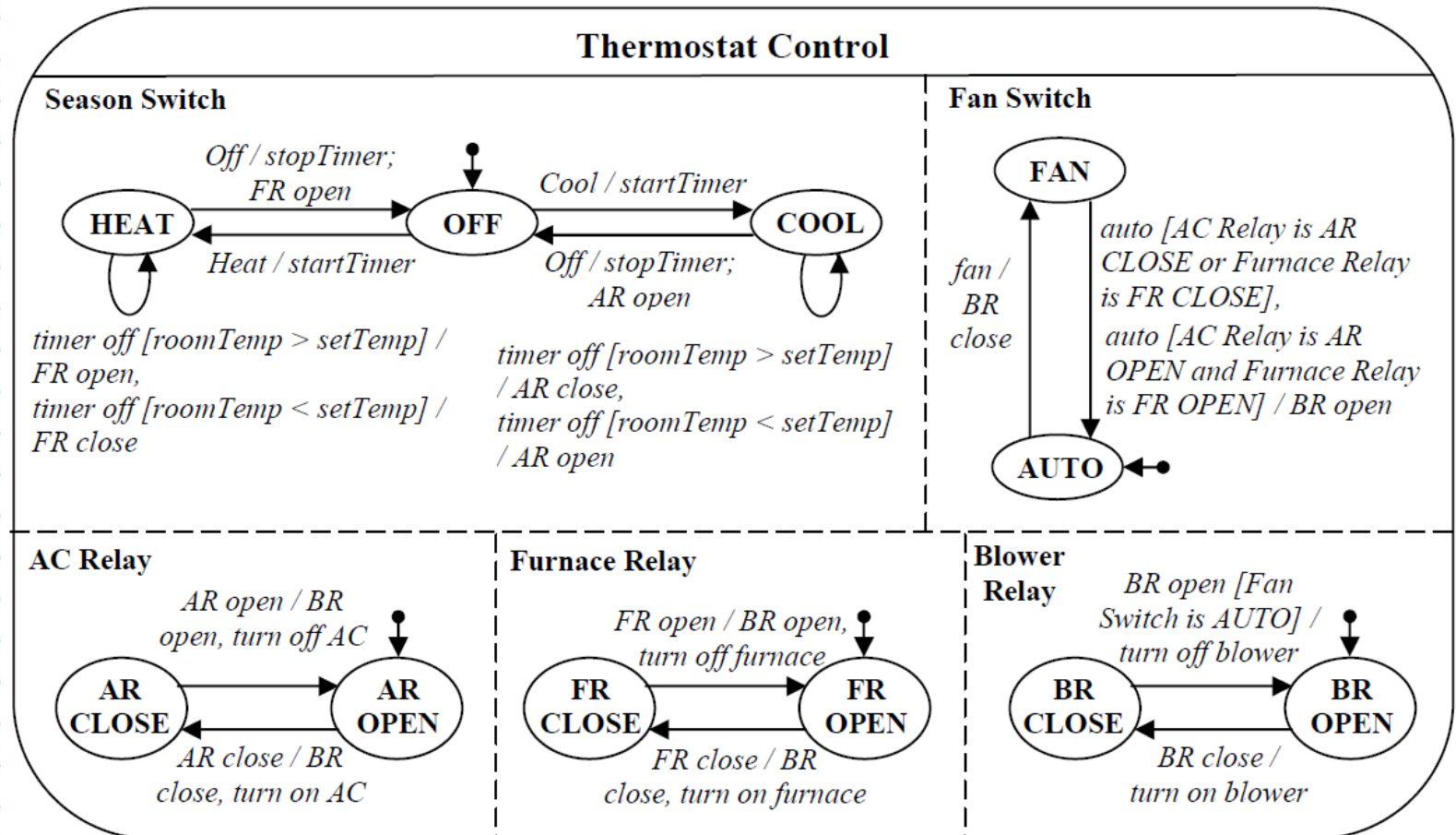
UML Statechart

- A UML state chart is actually a Harel statechart. A very expanded notation of the simpler Mealy or Moore state diagrams.
- Like state transition diagrams, UML state diagrams are directed graphs - nodes denote states and connectors transitions
- **Event** – is a type not a specific occurrence
 - Token is an event for the turnstile, but each specific token is an instance of the Token event. For the Push event, if the Push event occurred at a specific time it would be an instance of the Push event. An event instance outlives the occurrence that caused it. A specific event instance can go through three stages. The event instance is
 1. **Received** (on the event queue).
 2. **Dispatched** to the state machine, at which point it becomes the current event.
 3. **Consumed** when the state machine finishes processing the event instance.
- A **state** is much like the previous but captures the specific aspects of system behavior in a more focused manner (a single state variable)

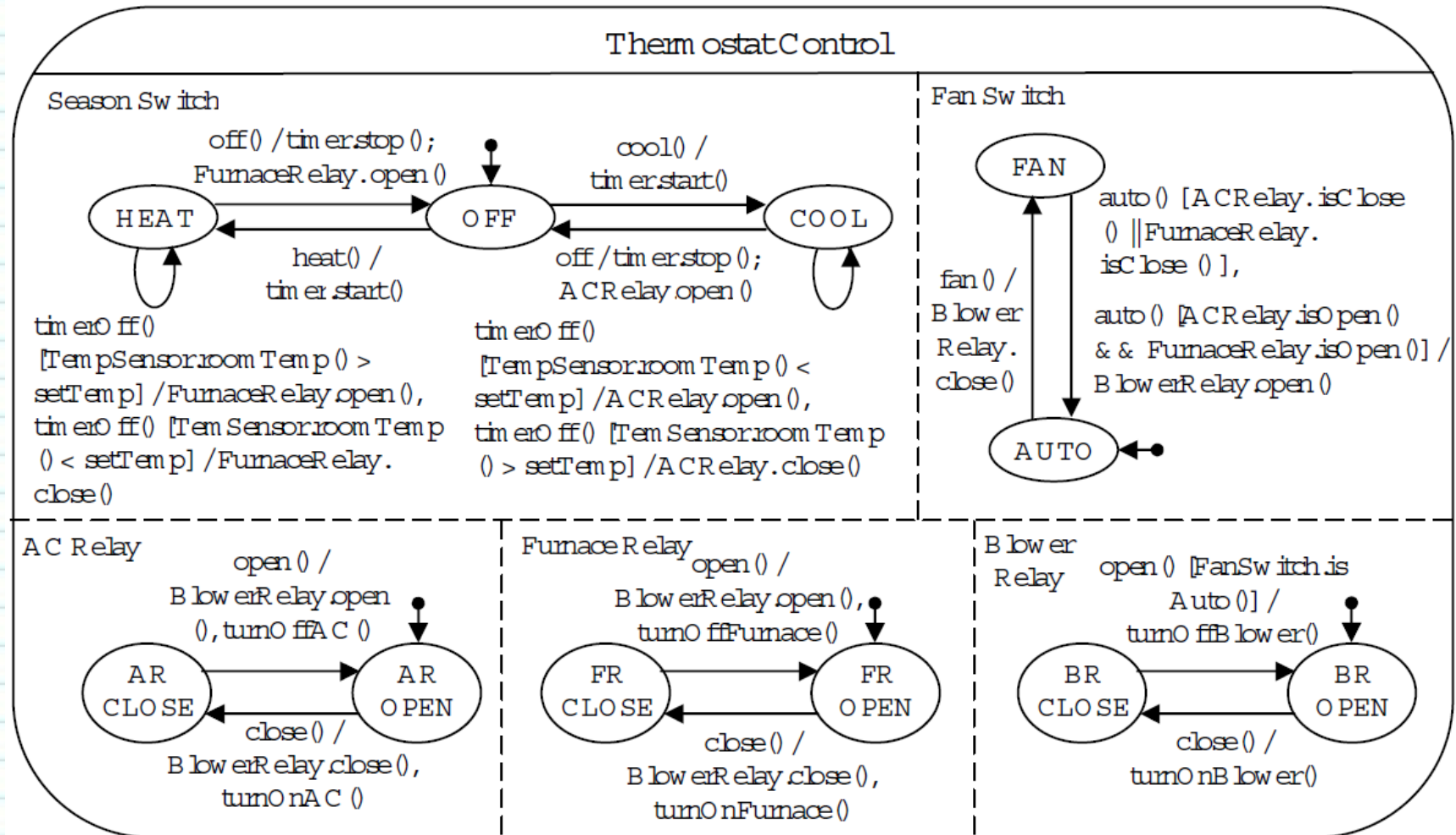
UML Statechart (cont.)

- **Guards** - Boolean expressions evaluated dynamically based on the value of event parameters. They affect the behavior only when they evaluate to TRUE and disable them when evaluated to FALSE.
- **Actions** – the dispatching of an event causes the state machine to performing actions.
- **Transitions** - switching from one state to another. The event that caused the transition is called the trigger.
- **Run to completion** - Incoming events cannot interrupt the processing of the current event and must be stored until the state machine can process them. This approach avoids any concurrency issues within a single state machine.
- My preference is to keep state machines in UML simple and represent them as either Mealy or Moore machines

Converting to State Diagram



Converting Texts to Function Calls

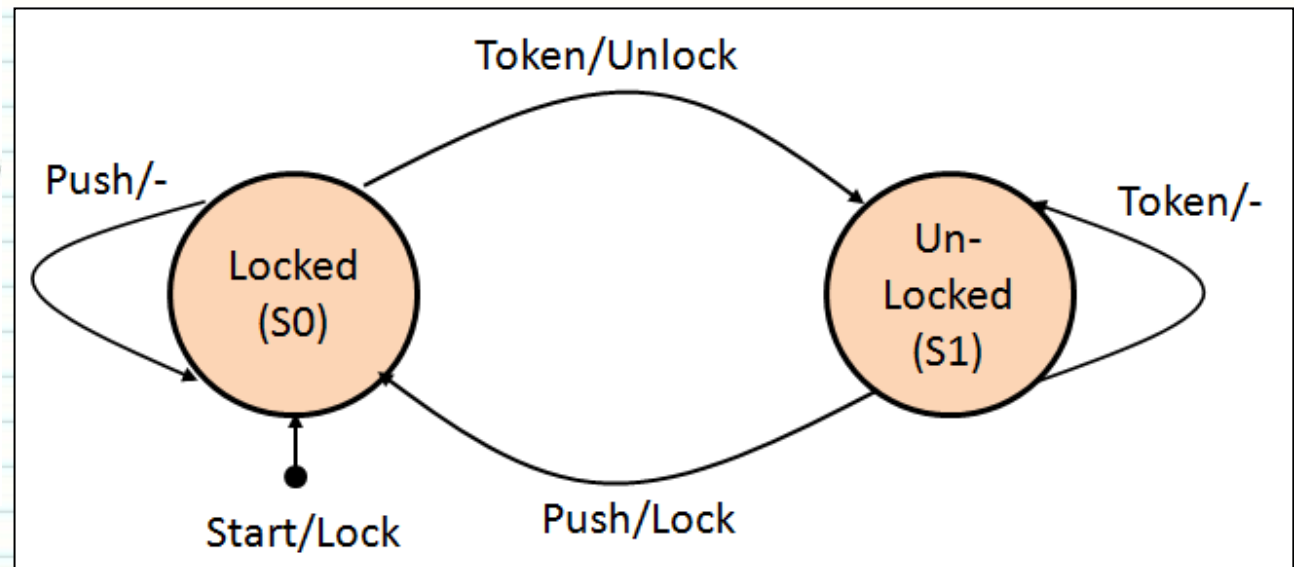


Implementing State Behavior

- Conventional approaches:
 - nested switch approach
 - using a state transition matrix
 - using method implementation

Exercises (5)

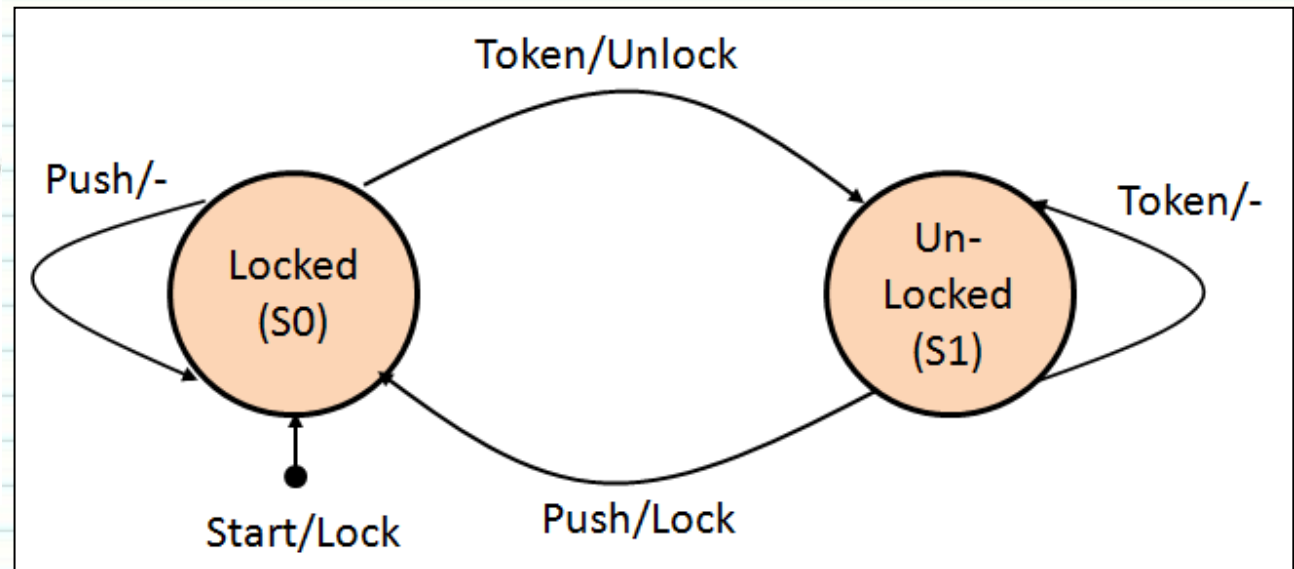
4. For the examples on the next slides perform the following:
- Draw a state transition diagram
 - Make a state transition table from the diagram
 - Define logical test cases



Conventional State Transition Matrix

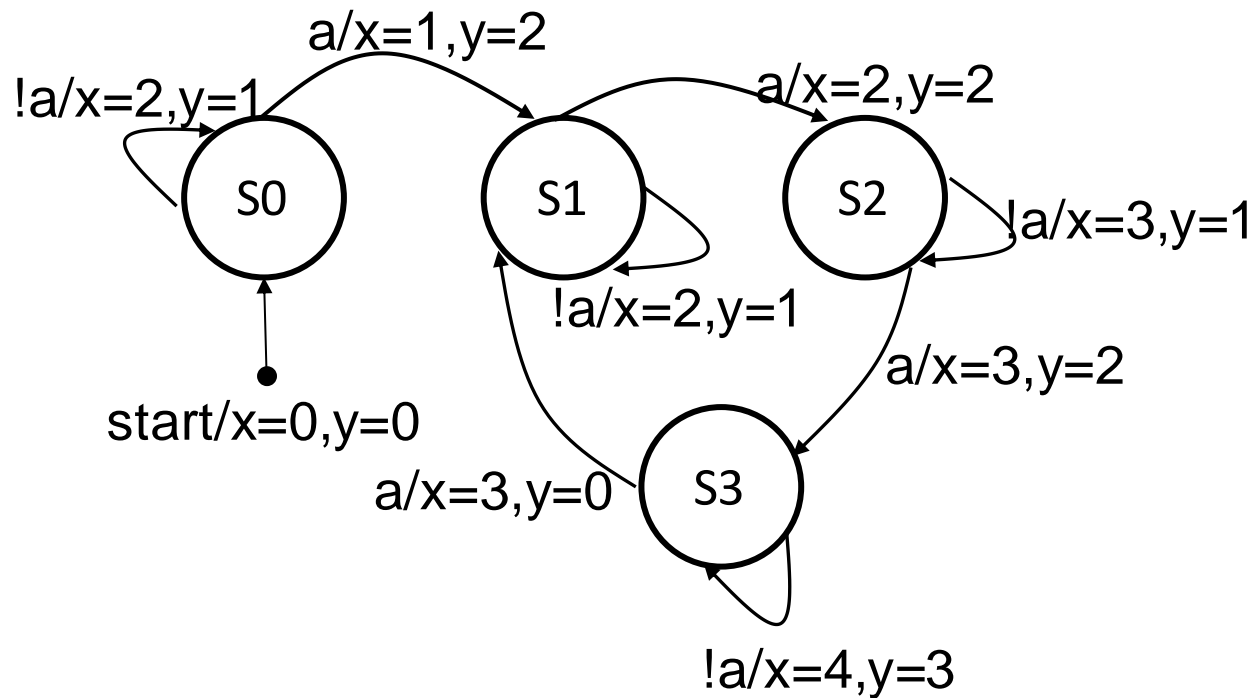
Current State	Input	Next State	Output
Start	-	Locked (S0)	Lock
Locked (S0)	Token	Unlocked (S1)	Unlock
Locked (S0)	Push	Locked (S0)	-
Unlocked (S1)	Token	Unlocked (S1)	-
Unlocked (S1)	Push	Locked (S0)	Lock

State Transition Matrix
or table will have at
least 4 columns



Example Diagram

- Example diagram - to determine next state we need to know
 - current state
 - input(s) value(s)



Example Diagram Solution

- Switch solution - we will use two state variables v1 and v2 to represent state
- Why do we need two variables?

State	v1	v2
S0	0	0
S1	0	1
S2	1	0
S3	1	1

- We will represent the state diagram with the following table

Current state	Inputs			Outputs				Next State
	V1	V2	a	x	y	V1	V2	
Start	-	-	-	0	0	0	0	S0
S0	0	0	TRUE	1	2	0	1	S1
S0	0	0	FALSE	2	1	0	0	S0
S1	0	1	TRUE	2	2	1	0	S2
S1	0	1	FALSE	2	1	0	1	S1
S2	1	0	TRUE	3	2	1	1	S3
S2	1	0	FALSE	3	1	1	0	S2
S3	1	1	TRUE	3	0	0	1	S1
S3	1	1	FALSE	4	3	1	1	S3

State Code - Switch Solution

```
package Problem_1;

public class StateTableClass {

    public void nextState (int v1, int v2, boolean a, StateTableOutputData outputs) {

        switch (v1*2 + v2) {

            case 0: if (a) {
                        outputs.setX(1);outputs.setY(2);outputs.setV1(0);outputs.setV2(1); }
                    else {
                        outputs.setX(2);outputs.setY(1);outputs.setV1(0);outputs.setV2(0); }

                    break;

            case 1: if (a) {
                        outputs.setX(2);outputs.setY(2);outputs.setV1(1);outputs.setV2(0); }
                    else {
                        outputs.setX(2);outputs.setY(1);outputs.setV1(0);outputs.setV2(1); }

                    break;

        }

    }

}
```

State Code - Switch Solution

```
case 2: if (a) {  
    outputs.setX(3);outputs.setY(2);outputs.setV1(1);outputs.setV2(1); }  
    else {  
        outputs.setX(3);outputs.setY(1);outputs.setV1(1);outputs.setV2(0); }  
  
    break;
```

```
case 3: if (a) {  
    outputs.setX(3);outputs.setY(0);outputs.setV1(0);outputs.setV2(1); }  
    else {  
        outputs.setX(4);outputs.setY(3);outputs.setV1(1);outputs.setV2(1); }  
  
    break;
```

```
// this is required to address the CWE issue with switches without defaults - there is no  
// correct value to set the outputs, but some value needs to be set to each output so the  
// corresponding test case will pass.
```

```
default: outputs.setX(0);outputs.setY(0);outputs.setV1(0);outputs.setV2(0); }  
  
}
```

```
}
```

Table Solution

- We will use two state variables v1 and v2 to represent state as before, we will also convert the input a into an integer and use the following table approach

Current state	Inputs			Outputs				Next State	index
	V1	V2	a	x	y	V1	V2		
Start	-	-	-	0	0	0	0	S0	
S0	0	0	1	1	2	0	1	S1	1
S0	0	0	0	2	1	0	0	S0	0
S1	0	1	1	2	2	1	0	S2	3
S1	0	1	0	2	1	0	1	S1	2
S2	1	0	1	3	2	1	1	S3	5
S2	1	0	0	3	1	1	0	S2	4
S3	1	1	1	3	0	0	1	S1	7
S3	1	1	0	4	3	1	1	S3	6

Table Solution (cont.)

- The code is short and elegant

```
public class StateTableClass {
```

```
    public void nextState (int v1, int v2, boolean a, StateTableOutputData outputs) {
```

```
        int x[]={1,2,2,2,3,3,3,4}, y[]={2,1,2,1,2,1,0,3},
```

```
            v1_tab[]={0,0,1,0,1,1,0,1},v2_tab[]={1,0,0,1,1,0,1,1};
```

```
        int index, aInt=a? 0:1;
```

```
        index=v1*4+v2*2+aInt;
```

```
        outputs.setX(x[index]);
```

```
        outputs.setY(y[index]);
```

```
        outputs.setV1(v1_tab[index]);
```

```
        outputs.setV2(v2_tab[index]);
```

```
    }
```

```
}
```

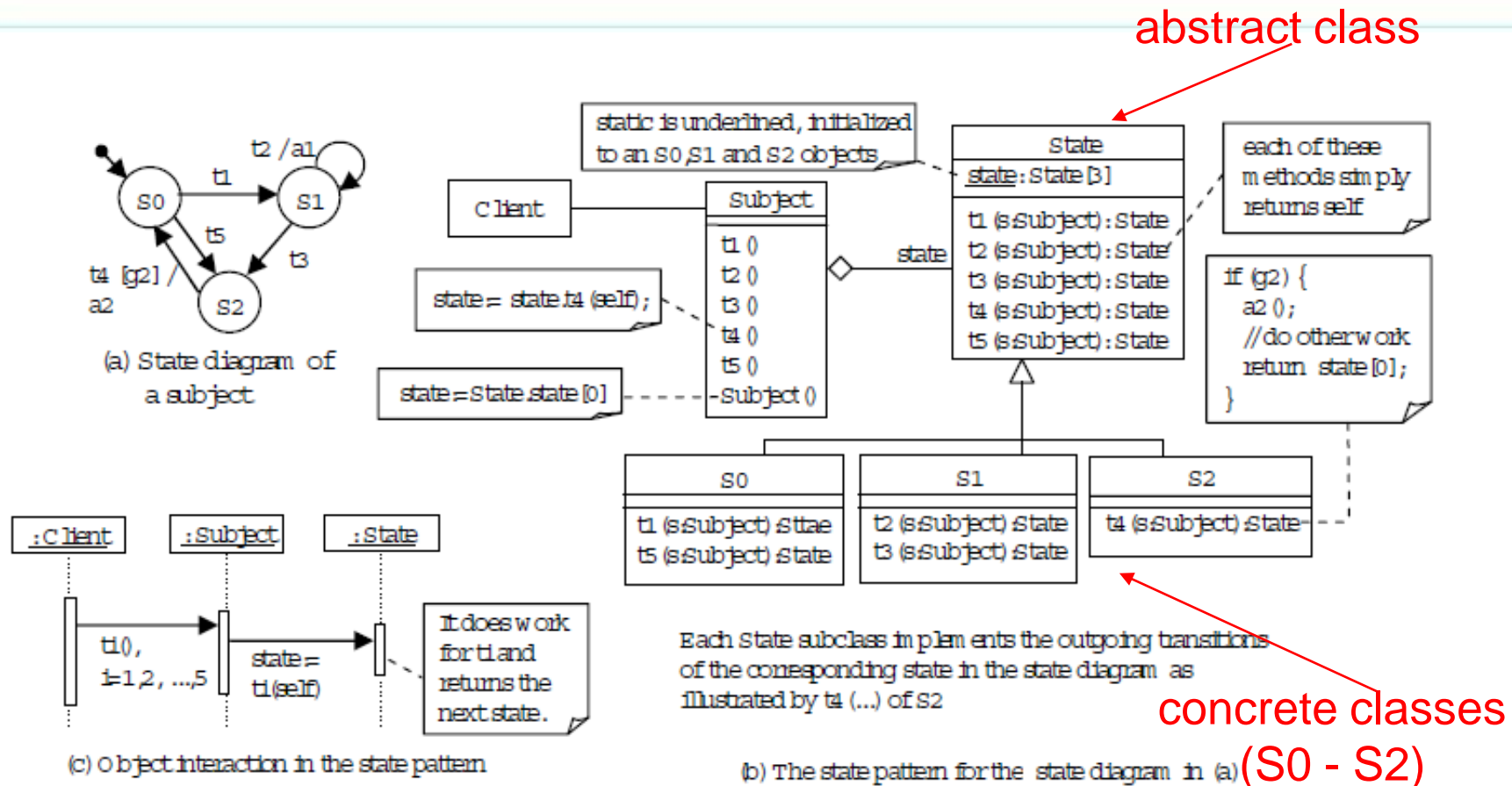
Table Solution (cont.)

- The table solution can get cumbersome when the number of states is large or the number of states and inputs are large
 - May end up with a sparse state
 - Best for small states with fewer inputs

Using Method Implementation

- State behavior of a class is implemented by member functions that denote an event in the state diagram.
- The current state of the object is stored in an attribute of the class.
- A member function evaluates the state variable and the guard condition, executes the appropriate response actions, and updates the state variable.

State Pattern



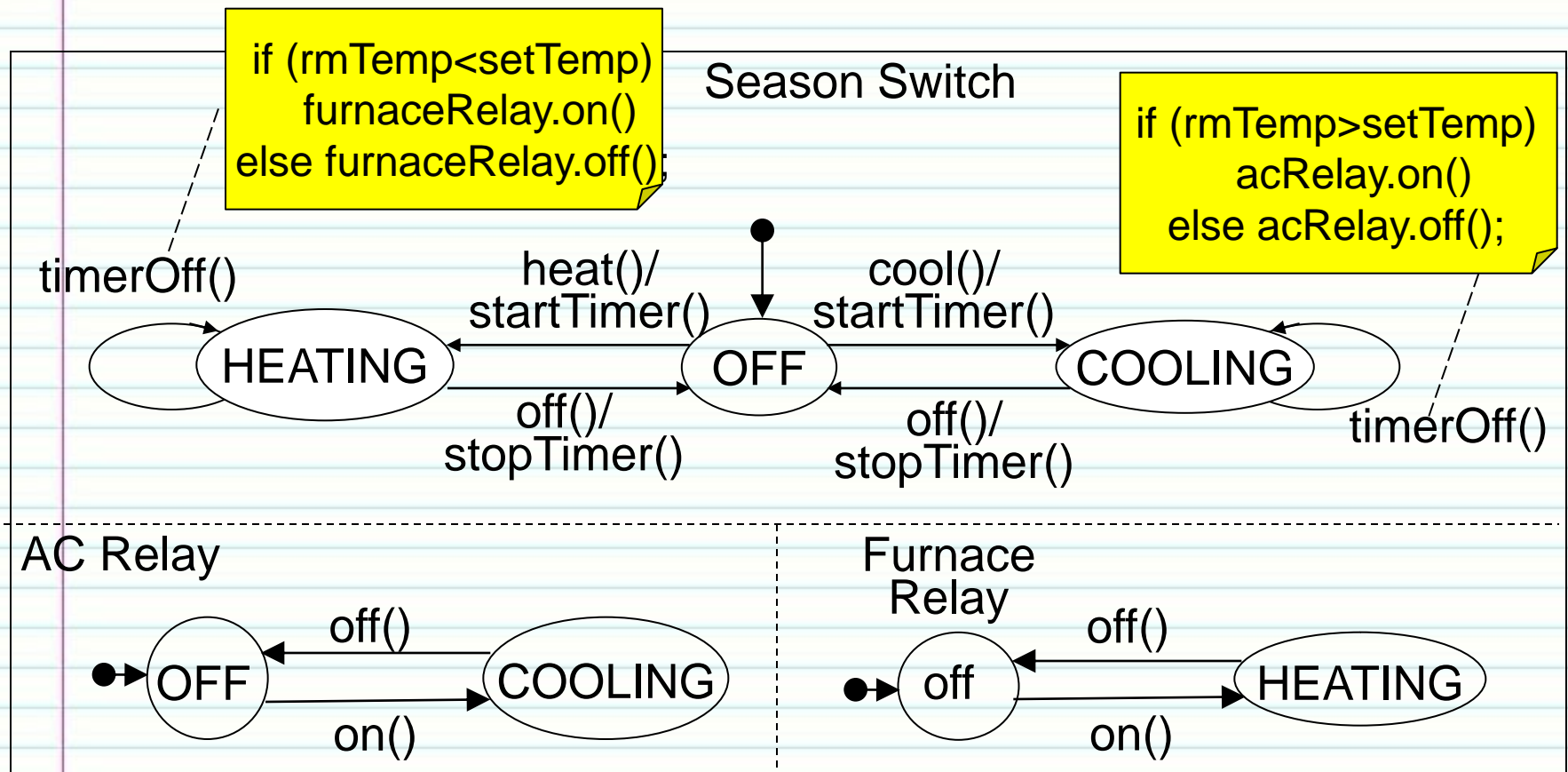
Subject: is the thing that is being modeled (vending machine).
Client is the user of the state model.

Benefits of State Pattern

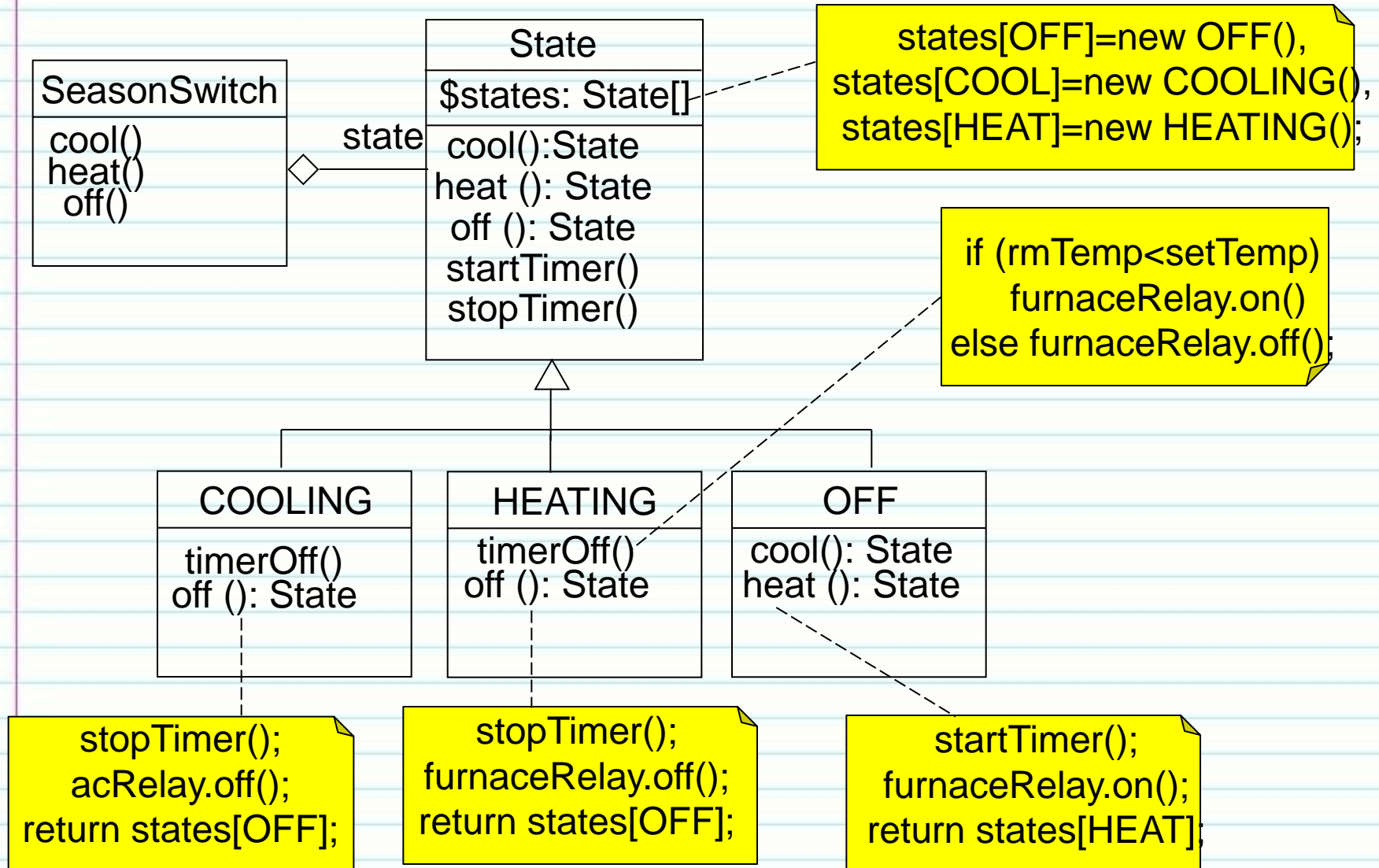
- Significantly reduces the cyclomatic complexity of the state handling code.
- Easy to add states --- simply add state subclasses and implement the relevant operations.
- Easy to add transitions --- simply add operations to the State class and implement the operations in relevant State subclasses.
- Easy to understand, implement, test and maintain.

Thermostat State Diagram

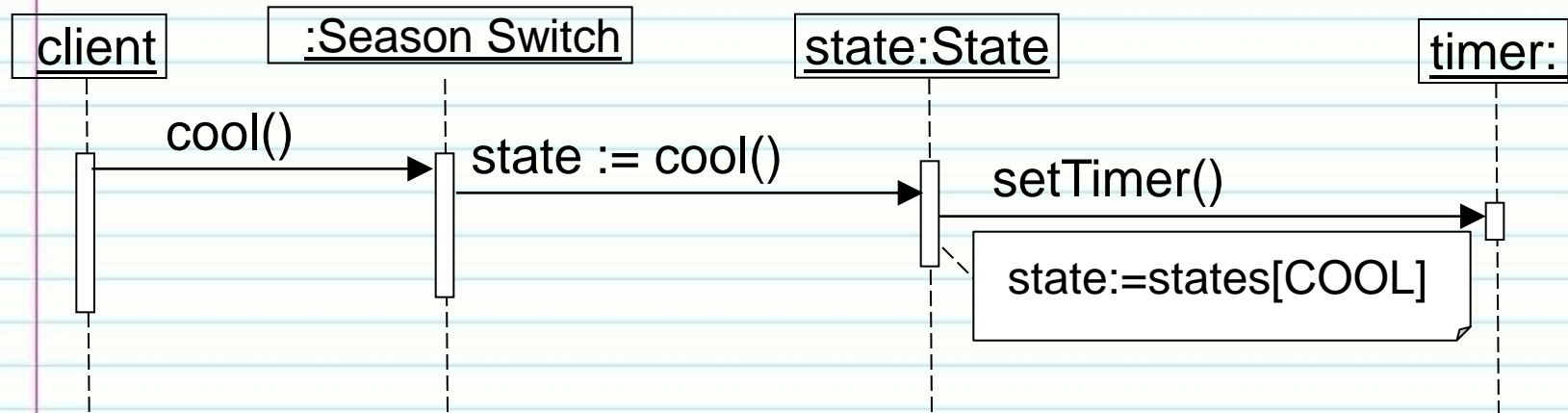
Apply the state pattern to the season switch state diagram of a thermostat.



The Season Switch State Pattern



Object Interaction in Season Switch State Pattern



Awareness of the State Design Pattern

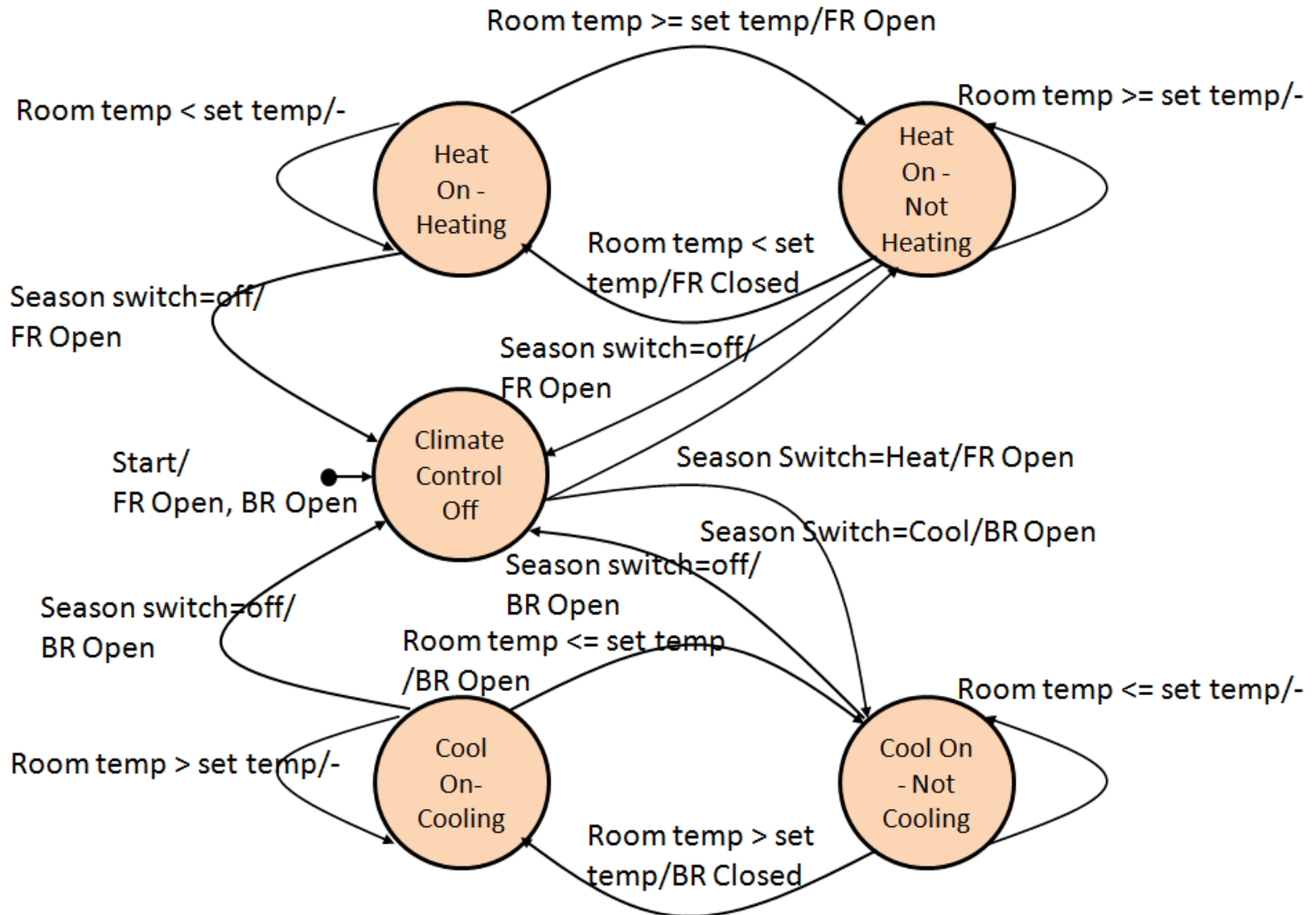
- Be aware this pattern exists and it is a solution
- Most state models have some basic, repeated problems
 1. Avoid states when possible - most people overuse states. States should only be used when history is important.
 2. If a state does not represent an obvious behavioral state of the system/software it is not a state.
 3. When using states make them as simple as possible.
 - a. Use an enumeration instead of a state when possible (e.g., off, cool, heat)
 - b. Don't use states to calculate values - how much change needs to be returned is a calculation, not a state.

Alternative Approach State Diagram (cont.)

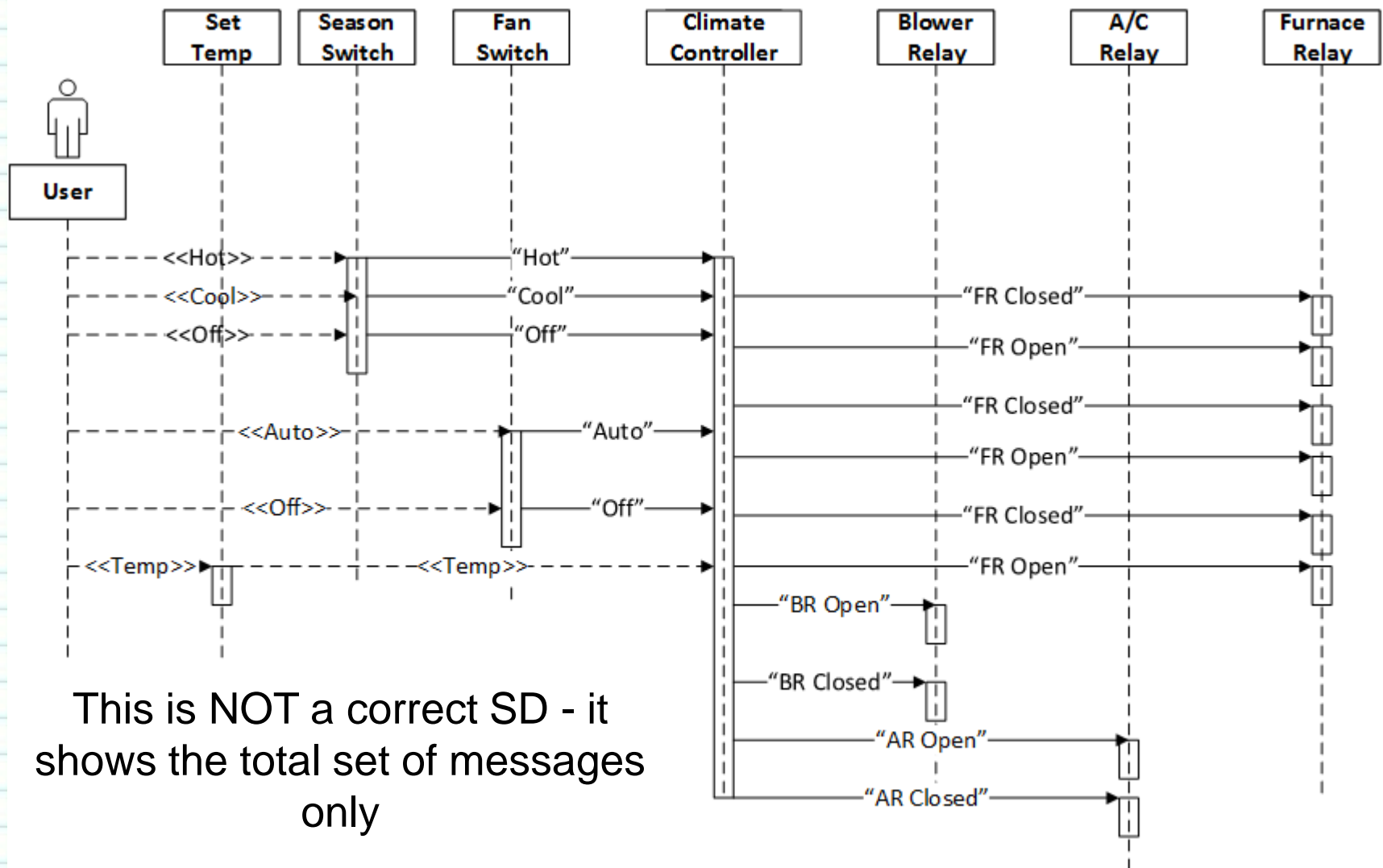
- Using this approach the AR, FR, and BR are now enumerations only - they have no state information.
- BR = AR Closed | FR Closed | Fan=On
- The state information is contained in the single diagram and will be implemented using the Controller pattern.
- Each of the devices: AR, FR, BR are objects on the sequence diagram.
- The Controller manages these devices. The objects are dumb and simply turn on and off switches.
- Three "GUI" objects - the Set Temp, the Season switch and the Fan Switch that go on the Sequence Diagram.
- The following slide shows the state diagram for the Controller which now manages each of the objects.

Alternative Approach State Diagram

Thermostat Climate Control (Heat/Cool)



Climate Controller Analysis Sequence Diagram



Guidelines for State Design Pattern

- Know the pattern - others use it so you need to be able to recognize it and converse about it
- Software Engineering is about making trade-offs
- Possible choices
 1. Switch,
 2. nested if,
 3. table,
 4. State pattern, or
 5. Controller approach
- Which is easier to change? Which is easier to understand? Which provides the best isolation?