ECE 321 Software Requirements Engineering

Lecture 10: Finite state machines

Mid-term course and instruction feedback

- Thanks for giving it! Will try to address everything
- Main things I will try to improve for this year:
  - Make the lectures as interactive as possible
  - Make content less dry/boring
- Main things I will improve for next year:
  - 2x50 mins lecture instead of 1x110
  - Change first two reading assignments

Last week: descriptive specification

- Algebraic specification
- Set of values and their operations
- Axioms to define the relations between the operations

Substring specification (version 1)

- For Substring(“student”, 3) == “dent”:

```
Substring(Create, i) = “”;
Substring(Add(s, c), i) =
    if(i >= Length(s)) then “”;
    else Add(Substring(s, i), c);
```

Substring specification (version 2)

- For Substring(“student”, 3) == “stu”:

```
Substring(Create, i) = “”;
Substring(Add(s, c), i) =
   if(Length(s) < i) then
      Add(Substring(s, i), c);
   else Substring(s, i);
```

This week: operational specification

- Describe the system in terms of control aspects (not data)
- Many formalisms exist:
  - Entity-Relationship Diagrams (semi-formal)
  - Finite State Machines (formal)
  - Data Flow Diagrams (semi-formal)
  - Petri-Nets (formal)

Finite State Machines (FSMs)

- $M = \{Q, I, T\}$, where
  - Q is a finite set of *states*
  - I is a finite set of *inputs*
  - T is a *transition function*
    - Defines what is the next state for an input and a state
    - $T : Q \times I \rightarrow Q$
    - T can be a partial function

What are FSMs suitable for?

- Describing systems
  - That have a finite set of states
  - That can go from one state to another as a consequence of some event (input)
- Notes:
  - FSMs with more than 7 nodes are hard to read
  - We need tools to analyze larger FSMs
    - e.g. SPIN model checker

An example FSM

- States $Q = \{q_1, q_2, q_3\}$
- Inputs $I = \{i_1, i_2, i_3, i_4\}$
- Transition function T:

	i_1	i_2	i_3	i_4
q_1	q_2			q_1
q_2		q_1	q_3	
q_3		q_3	q_1	

Interpreting the transition function

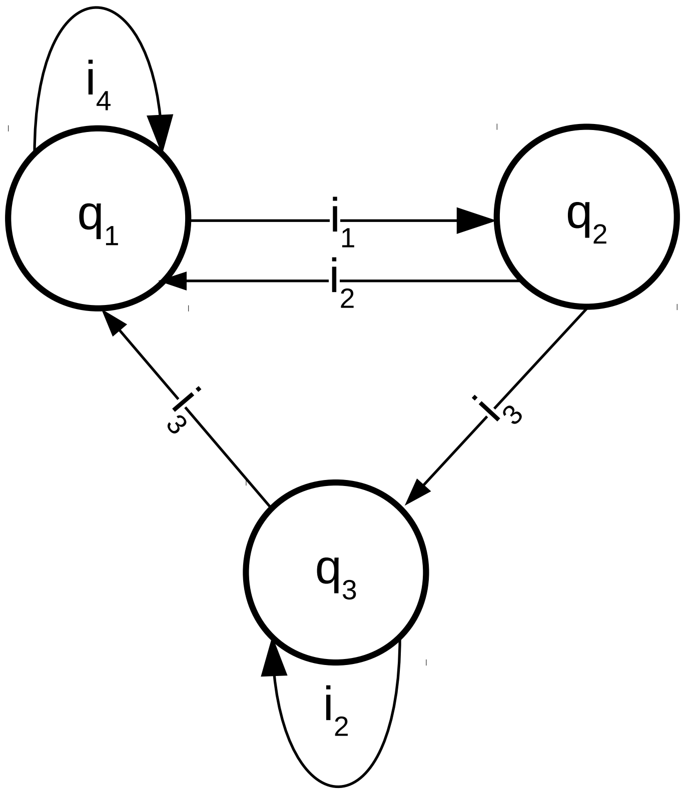
- Very simple to understand execution model
  - Machine is in some state
  - Input causes state change according to *T*

	i_1	i_2	i_3	i_4
q_1	q_2			q_1
q_2		q_1	q_3	
q_3		q_3	q_1	

We can also show an FSM as a graph

- Nodes represent states
- Edges are directed and labeled with inputs
  - Edge labeled *i* goes from state q_1 to state q_2
    - Iff (if and only if) $T(q_1, i) = q_2$

Drawing our example FSM



	i_1	i_2	i_3	i_4
q_1	q_2			q_1
q_2		q_1	q_3	
q_3		q_3	q_1	

Partial transition functions

- Some (state, input) pairs are undefined
- If we try to execute them
  - They are ignored or generate error
  - System does not change the state

	i_1	i_2	i_3	i_4
q_1	q_2			q_1
q_2		q_1	q_3	
q_3		q_3	q_1	

Describing execution

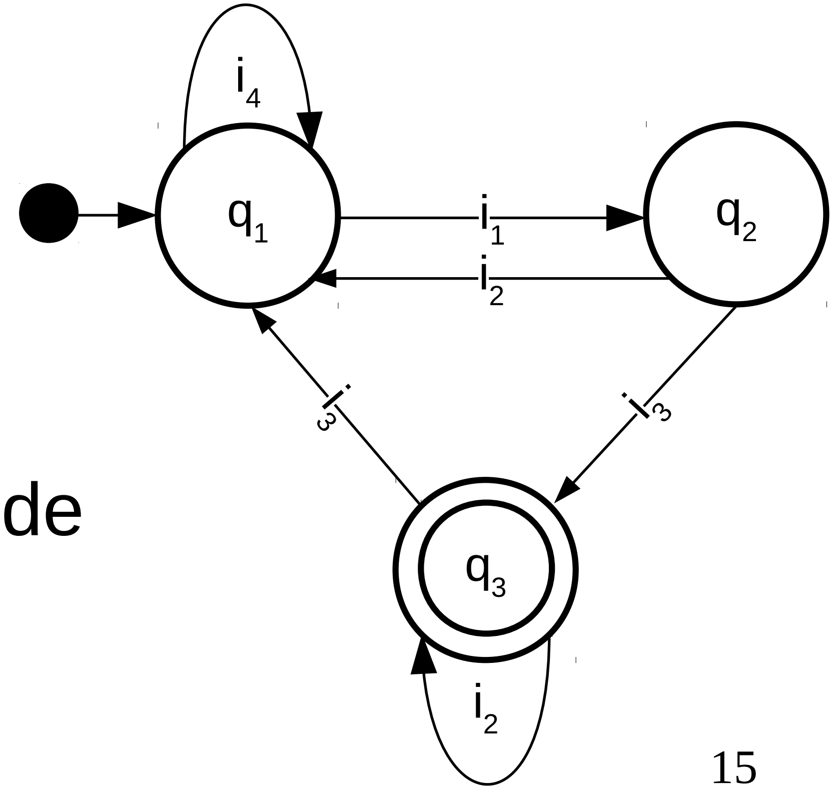
- We can use sequences of states or inputs

	i_1	i_2	i_3	i_4
q_1	q_2			q_1
q_2		q_1	q_3	
q_3		q_3	q_1	

- $q_1\ q_2\ q_1\ q_1\ q_2\ q_3$ ....
  - Or $i_1\ i_2\ i_4\ i_1\ i_3$

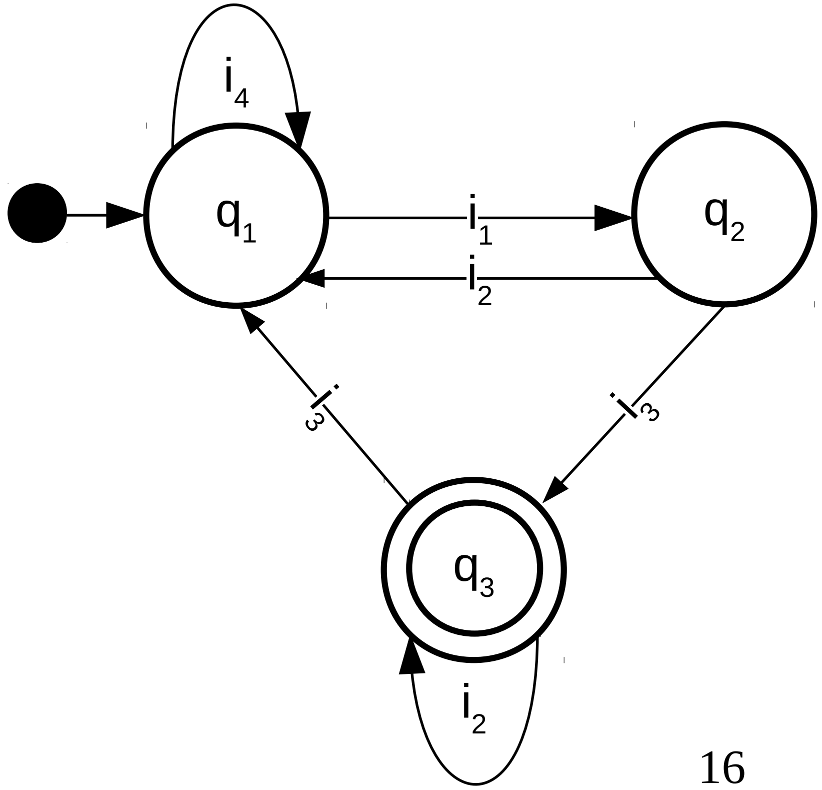
Initial states and stop states

- Initial state q_1
  - Indicate with black dot and arrow
- Stop state q_3
  - Denoted by double circled node



Initial states and stop states

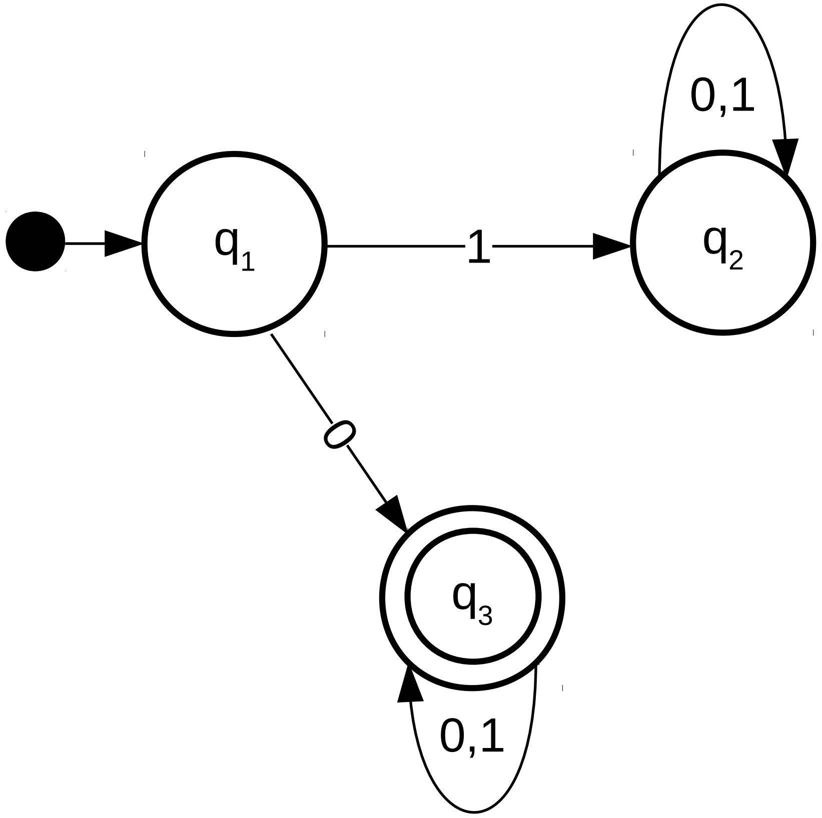
- Are these executions accepted?
- i_1 i_3 i_3 i_4 i_4 i_1 i_3
  - Yes
- i_1 i_3 i_3
  - No
- i_1 i_1 i_3 i_1
  - Yes/no (depending on whether we error or ignore undefined inputs)



State machines can be used to accept strings in a language

- e.g., The language of binary strings that start with ‘0’
  - Accepted: 0, 0001, 010101, 01111 etc.
  - Not accepted: 1, 1110, 1000, etc.
- How would such a FSM look?

FSM that accepts strings that start with 0



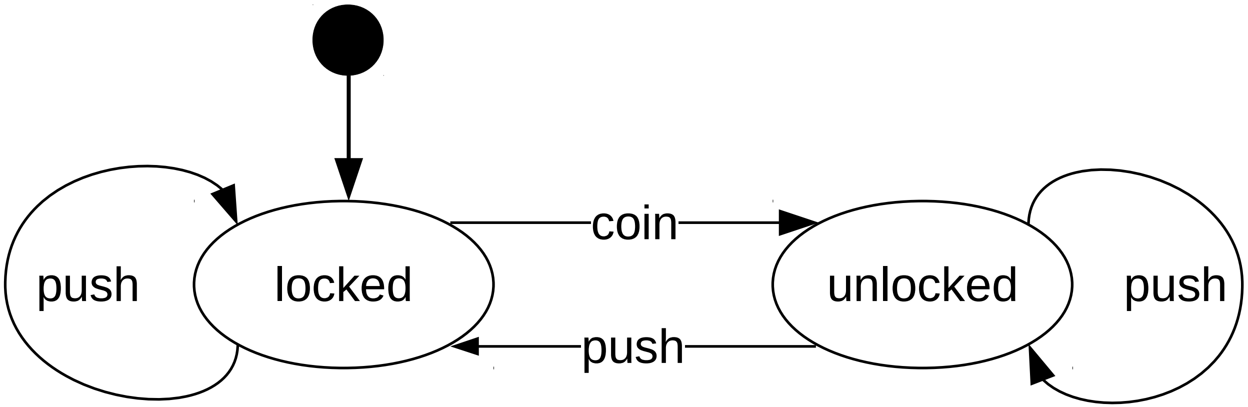
Another example FSM: Turnstile

States Q = *{locked, unlocked}*
Inputs I = *{insert coin, push}*
Initial state = *locked*

Another example FSM: Turnstile

States Q = *{locked, unlocked}*
Inputs I = *{insert coin, push}*
Initial state = *locked*

	coin	push
locked	unlocked	locked
unlocked	unlocked	locked

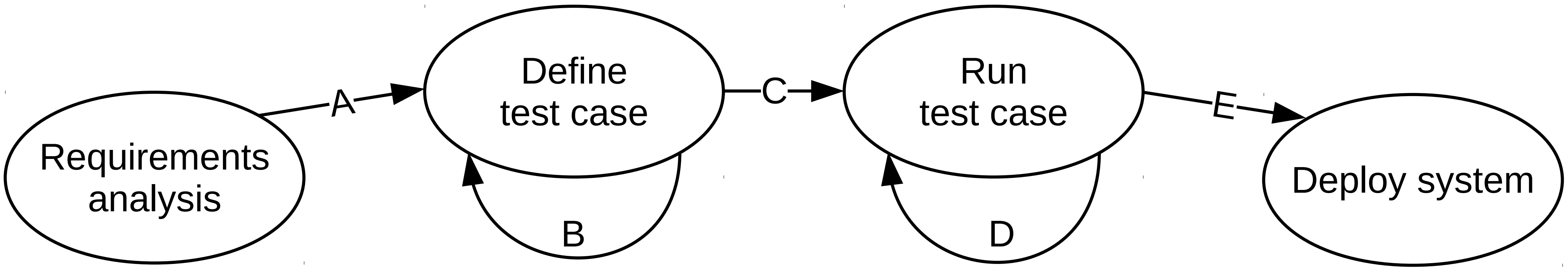


Another example application of an FSM: GUI design

- Represent windows and their states as nodes, user inputs as edges
- Allows to analyze the design and correct flaws
  - Are some windows hard to reach?
  - Does the flow of interaction make sense?

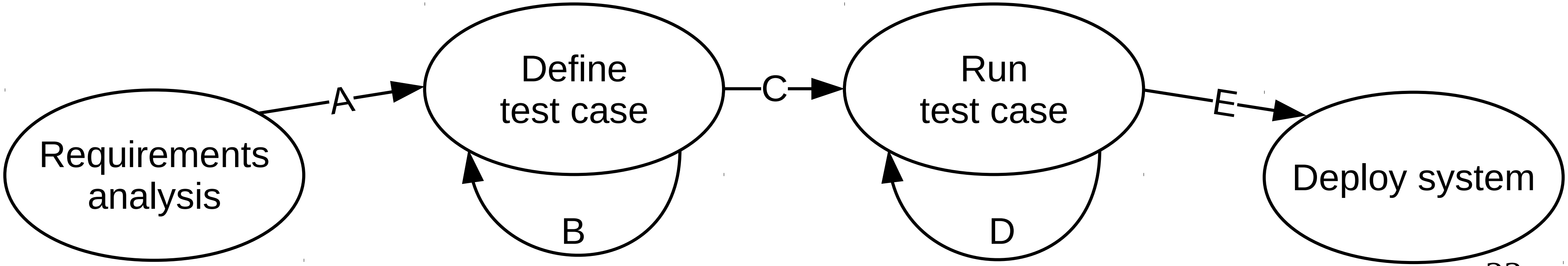
Another example FSM: Test execution

- Model software testing procedure using FSM



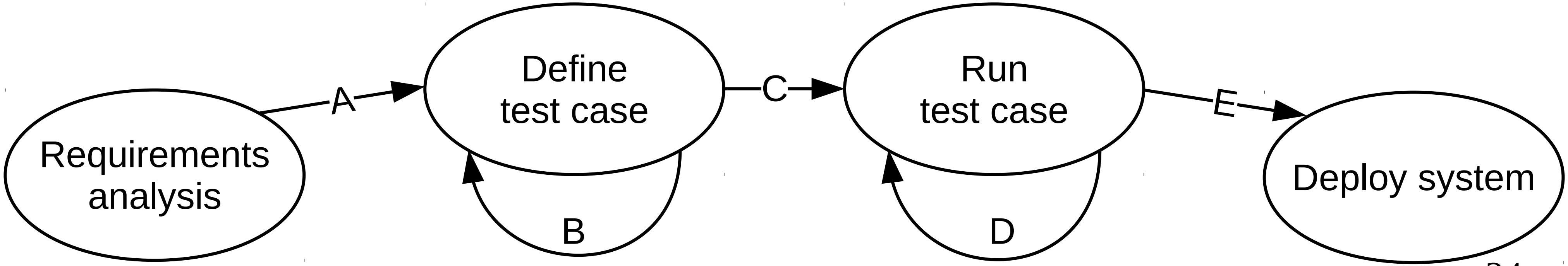
Possible test executions

- A B B C D D E
  - OK
- A B B C D D D E
  - One test case done twice
- A B B C D E
  - One test case skipped

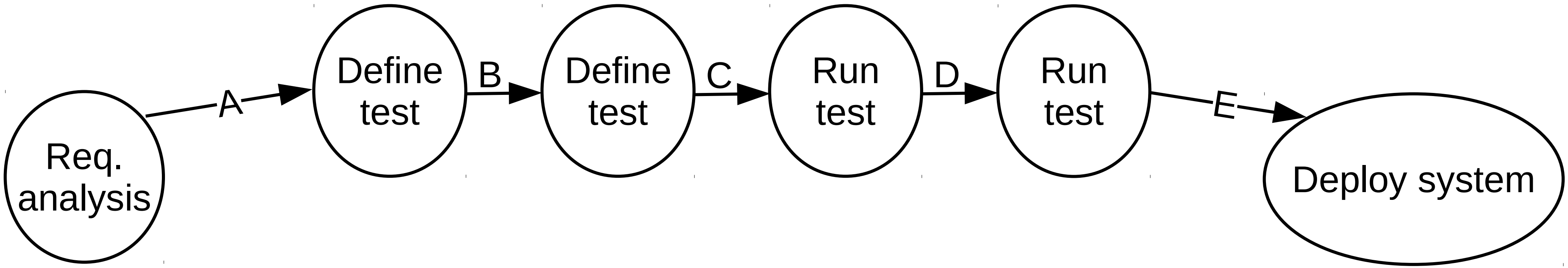


Possible test executions

- We want to have the same number of B and D
  - This model cannot guard against this
- How can we fix that?



Making sure all defined tests are executed

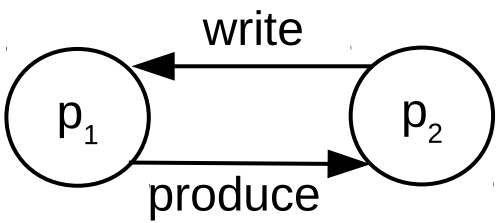


- Create separate states for each creation and execution of a test case
- What is wrong with this approach?
  - Fixes the number of test cases
  - State space explosion effect

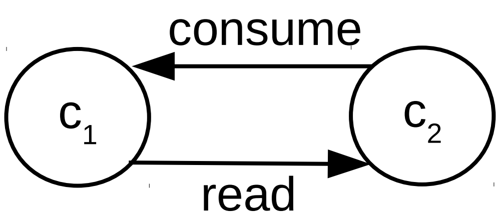
Demonstrating state space explosion

- Example: producer-consumer system
  - Producer **p** produces messages and puts them into a 2-slot buffer
  - Consumer **c** reads messages and removes them from the same buffer
  - If the buffer is full producer waits until consumer removes something
  - If the buffer is empty consumer waits until producer inserts a message

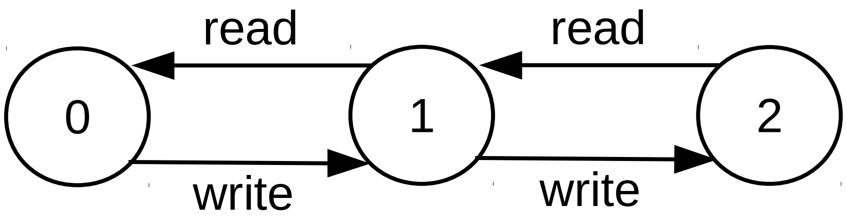
FSM for each of the objects



Producer



Consumer



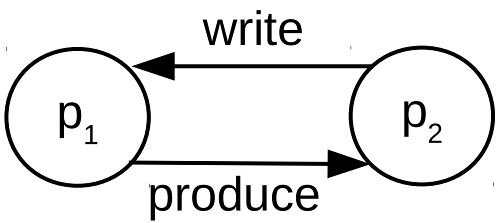
Buffer

Creating a combined FSM

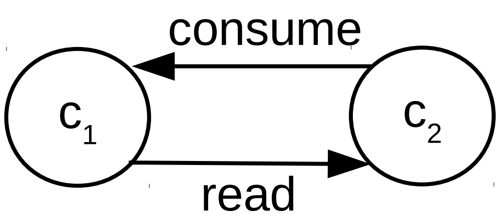
- The three FSMs are pieces of a single synchronized system
- How many states do we have in the combined FSM?
  - Cartesian product of the component state sets:

    2 x 2 x 3 = 12 states
  - Leads to state explosion (and this is a trivial system)

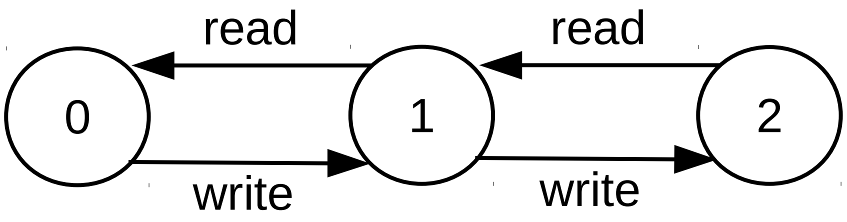
Let’s create the combined FSM



Producer



Consumer



Buffer

Step 1: identifying the states and input

- States *Q*
  - All combinations of states

    $\{<0, p_1, c_1>, <0, p_1, c_2>, <0, p_2, c_1>, <0, p_2, c_2>, ...\}$

- Inputs *I*
  - All possible inputs in the FSMs

    {produce, write, consume, read}

Step 2: identifying the transitions

	produce	write	consume	read
$0, p_1, c_1$				
$0, p_1, c_2$				
$0, p_2, c_1$				
$0, p_2, c_2$				
$1, p_1, c_1$				
$1, p_1, c_2$				
$1, p_2, c_1$				
$1, p_2, c_2$				
...				

Step 2: identifying the transitions

	produce	write	consume	read
0, p_1, c_1	0, p_2, c_1	-	-	-
0, p_1, c_2	0, p_2, c_2	-	0, p_1, c_1	-
0, p_2, c_1	-	1, p_1, c_1	-	-
0, p_2, c_2	-	1, p_1, c_2	0, p_2, c_1	-
1, p_1, c_1	1, p_2, c_1	-	-	0, p_1, c_2
1, p_1, c_2	1, p_2, c_2	-	1, p_1, c_1	-
1, p_2, c_1	-	2, p_1, c_1	-	0, p_2, c_2
1, p_2, c_2	-	0, p_1, c_2	1, p_2, c_1	-
...				

Step 3: Drawing the FSM

- ... a great exercise for at home :)

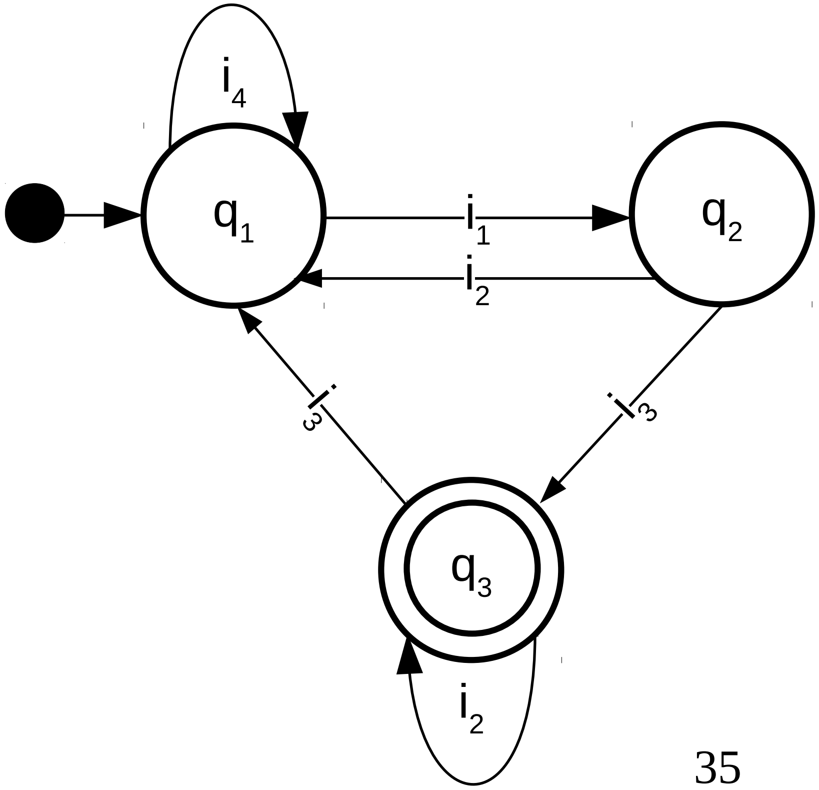
- For example, use GraphViz
  - https://www.graphviz.org/
  - Open source tool for drawing graphs

Advantages of FSMs

- Simple and intuitive model
  - Especially the graphical representation
- Support tools exist for FSM models
  - https://en.wikipedia.org/wiki/List_of_model_checking_tools
  - Transformers: generate code from a model
  - Analyzers: analyze system using FSM model

Which questions can analyzers answer?

- Starting in q_1 can I reach q_3?
- Am I guaranteed to reach q_3 from q_1?
- Can I have two i_2 actions in a row?



Disadvantages of FSMs 1/3

- Limited computational power
  - FSM has no ‘memory’
    - Remember the test case example?
  - Other options
    - Push-Down Automata (FSM + stack (memory))
    - Linear Bounded Automata (constrained size memory)
    - Turing Machine (unlimited memory)

More disadvantages of FSMs 2/3

- State space explosion
  - For larger problems
    - Complete description of a system requires many states
  - For composed/combined FSMs
    - Require number of states that is a multiplication of the number of states in the component FSMs

Even more disadvantages of FSMs 3/3

- Inherently synchronous
  - FSM has a single, global state
  - We cannot express a situation when consumer and producer perform operations at the same time