CS 3100, Models of Computation, Spring 2020, Lec 3

Ganesh Gopalakrishnan
School of Computing
University of Utah
Salt Lake City, UT 84112

bit.ly/3100s20Syllabus

Lecture 3, covering up to Chapter 4.6

Introducing DFA

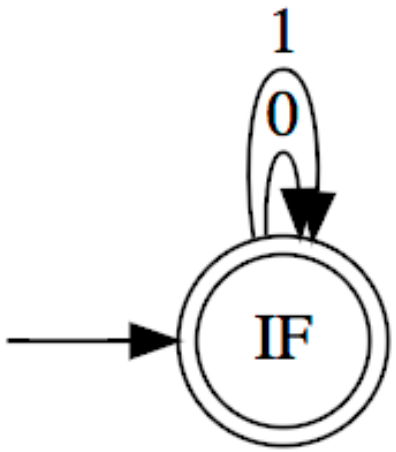
- Machines with a **finite set of states : "Q"**
- Is **defined for an alphabet** Sigma (usually {0,1})
- Has a **single initial state**
- Has **some number of final states "F"**
  - Could be none
  - Could be all of Q
- Has moves defined
  - **for every state q in Q**
  - **for every member of Sigma**
- **Helps define a language!**
  - Any string that takes the DFA from I to one of the states in "F" is in the DFA's language
  - **The DFA accepts such a string**
- **All the strings that go from I to F are in the language recognized by the DFA**

Is this a DFA?
Which strings are accepted by it?
What language is recognized by it?

In [12]:

```
1  # Do this in CH4-5-6
2
3  dotObj_dfa(md2mc('''
4
5  DFA
6
7  IF : 0|1 -> IF
8
9  '''))
```

Out[12]:

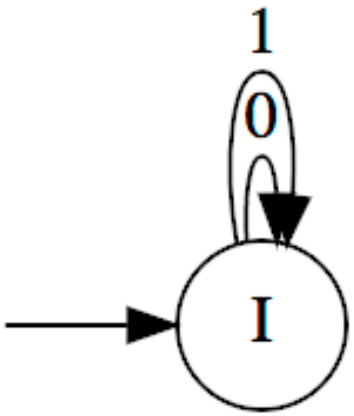


Is this a DFA?
Which strings are accepted by it?
What language is recognized by it?

In [11]:

```
1  # Do this in CH4-5-6
2
3  dotObj_dfa(md2mc('''
4
5  DFA
6
7  I : 0|1 -> I
8
9  '''))
```

Out[11]:

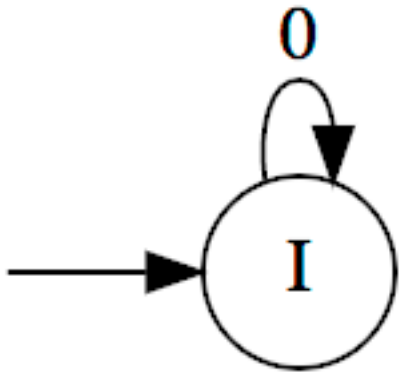


Is this even a DFA over {0,1} ?? Why or why not?

In [13]:

```
1  # Do this in CH4-5-6
2
3  dotObj_dfa(md2mc('''
4
5  DFA
6
7  I : 0 -> I
8
9  '''))
```

Out[13]:

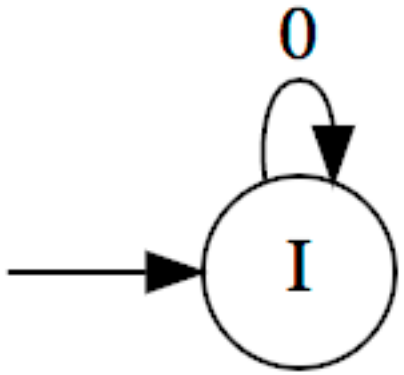


Is this a DFA over {0} ?

In [13]:

```
1  # Do this in CH4-5-6
2
3  dotObj_dfa(md2mc('''
4
5  DFA
6
7  I : 0 -> I
8
9  '''))
```

Out[13]:

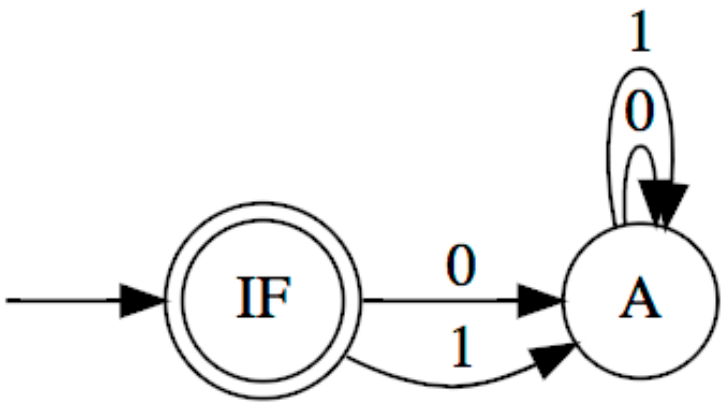


Is this a DFA?
Which strings are accepted by it?
What language is recognized by it?

In [14]:

```
# Do this in CH4-5-6

dotObj_dfa(md2mc('''

DFA

IF : 0|1 -> A

A  : 0|1 -> A

'''))
```

Out[14]:

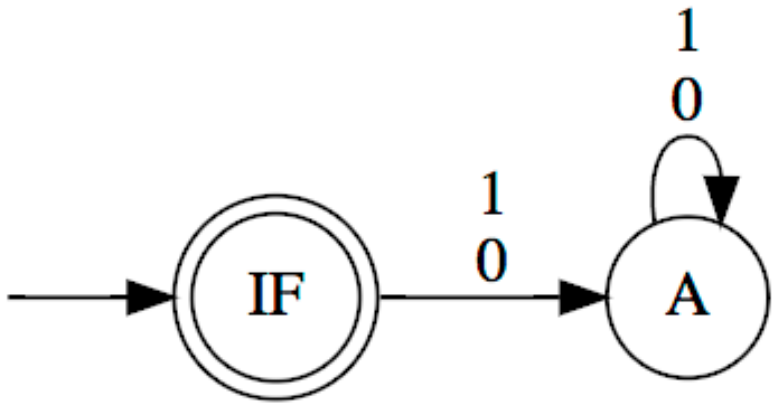


Is this a DFA?
Which strings are accepted by it?
What language is recognized by it?

In [15]:

```
1  # Do this in CH4-5-6
2
3  dotObj_dfa(md2mc('''
4
5  DFA
6
7  IF : 0|1 -> A
8
9  A  : 0|1 -> A
10
11 '''), FuseEdges=True)
```

Out[15]:

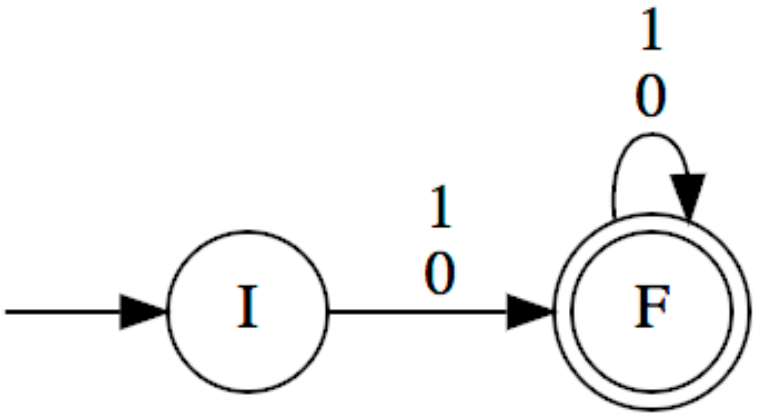


Is this a DFA?
Which strings are accepted by it?
What language is recognized by it?

In [16]:

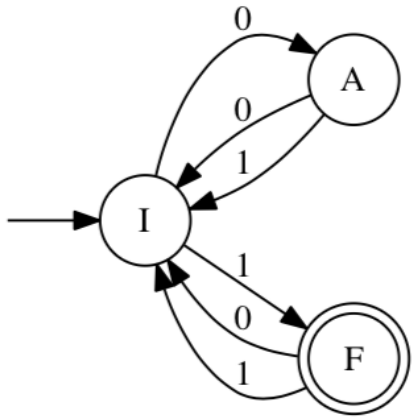
```
1  # Do this in CH4-5-6
2
3  dotObj_dfa(md2mc('''
4
5  DFA
6
7  I   : 0|1 -> F
8
9  F   : 0|1 -> F
10
11 '''), FuseEdges=True)
```

Out[16]:



Formal structure of a DFA, and an example

- Q is a *finite nonempty* set of states,
- Σ is a *finite nonempty* alphabet,
- $\delta : Q \times \Sigma \to Q$ is a *total* transition function,
- $q_0 \in Q$ is the initial state, and
- $F \subseteq Q$ is a *finite, possibly empty* set of final (or *accepting*) states.




```
{'Q': {'A', 'F', 'I'},
 'Sigma': {'0', '1'},
 'Delta': {('A', '0'): 'I',
           ('A', '1'): 'I',
           ('F', '0'): 'I',
           ('F', '1'): 'I',
           ('I', '0'): 'A',
           ('I', '1'): 'F'},
 'q0': 'I',
 'F': {'F'}}
```

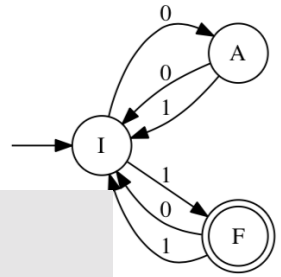
The language of a DFA is

" the strings that take the DFA from the initial state to one of the final states "

(if the initial state is also a final state, then the language also contains epsilon)

List five strings in this DFA's language (in numeric order)

step_dfa, run_dfa, accepts_dfa in code




```
def step_dfa(D, q, a):
    """Run DFA D from state q on character a.
    """
    assert(a in D["Sigma"])
    assert(q in D["Q"])
    return D["Delta"][(q,a)]
```

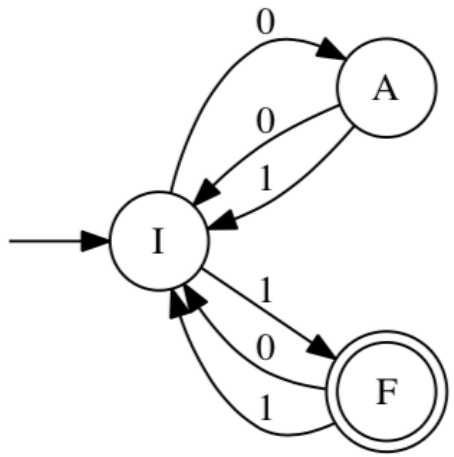
```
def accepts_dfa(D, w):
    """ Checks for DFA acceptance. Input : DFA D, string w.
        Output : Boolean (True|False).
    """
    return run_dfa(D, w) in D["F"]
```

```
def run_dfa(D, w):
    """In : D (consistent DFA)
          w (string over D's sigma, including "")
       Out: next state of D["q0"] via string w.
    """
    curstate = D["q0"]
    if w=="":
        return curstate
    else:
        return run_dfa_h(D, w[1:], step_dfa(D,curstate,w[0]))
```

```
def run_dfa_h(D, w, q):
    """Helper for run_dfa. Compute the next state attained
       by w running on D starting from state q.
    """
    if w=="":
        return q
    else:
        return run_dfa_h(D, w[1:], step_dfa(D, q, w[0]))
```

step_dfa, run_dfa, accepts_dfa in math

- Q is a *finite nonempty* set of states,
- Σ is a *finite nonempty* alphabet,
- $\delta : Q \times \Sigma \to Q$ is a *total* transition function,
- $q_0 \in Q$ is the initial state, and
- $F \subseteq Q$ is a *finite, possibly empty* set of final (or *accepting*) states.



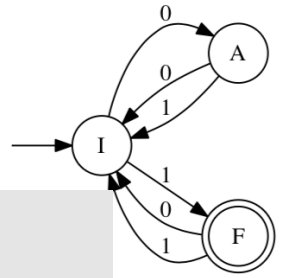
"step_dfa" $\delta(D, q, a)$

"run_dfa" $\hat{\delta}(D, q, ax) = \hat{\delta}(D, \delta(D, q, a), x)$

$\hat{\delta}(D, q, \varepsilon) = q.$

"accepts_dfa" $\hat{\delta}(D, q_0, w) \in F.$

step_dfa, run_dfa, accepts_dfa in both!




```python
def step_dfa(D, q, a):
    """Run DFA D from state q on character a.
    """
    assert(a in D["Sigma"])
    assert(q in D["Q"])
    return D["Delta"][(q,a)]
```

$\delta(D,q,a)$

```python
def accepts_dfa(D, w):
    """ Checks for DFA acceptance. Input : DFA D, string w.
        Output : Boolean (True|False).
    """
    return run_dfa(D, w) in D["F"]
```

$\hat{\delta}(D,q_0,w) \in F.$

```python
def run_dfa(D, w):
    """In : D (consistent DFA)
          w (string over D's sigma, including "")
       Out: next state of D["q0"] via string w.
    """
    curstate = D["q0"]
    if w=="":
        return curstate
    else:
        return run_dfa_h(D, w[1:], step_dfa(D,curstate,w[0]))
```

$\hat{\delta}(D,q,\varepsilon) = q.$

```python
def run_dfa_h(D, w, q):
    """Helper for run_dfa. Compute the next state attained
       by w running on D starting from state q.
    """
    if w=="":
        return q
    else:
        return run_dfa_h(D, w[1:], step_dfa(D, q, w[0]))
```

$\hat{\delta}(D,q,ax) = \hat{\delta}(D,\delta(D,q,a),x)$

Design a DFA for “ends with 0101”

Express mathematically as a language
Design in Jove syntax, keeping the relevant piece of the input in the state name
  i.e. S010 means “seen 010 so far”

Design a DFA for “contains 0101”

Express mathematically as a language
Design in Jove syntax, keeping the relevant piece of the input in the state name
  i.e. S010 means “seen 010 so far”

Some prefer to draw the DFA out also. But still, name states consistently.

Design a DFA for {0,1} where every block of length 3 contains exactly two 1’s

Design a DFA for strings over {0,1} whose numeric value is a multiple of 3