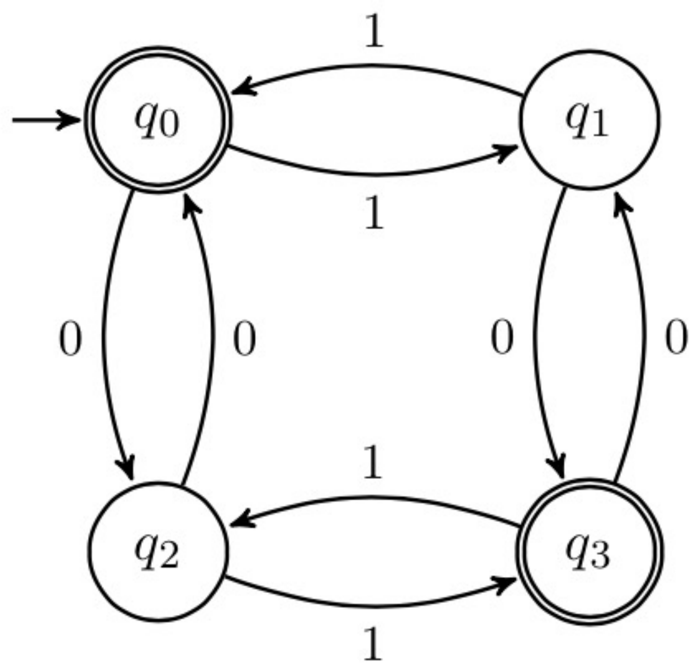
CSE 311: Foundations of Computing

Lecture 23: Finite State Machines



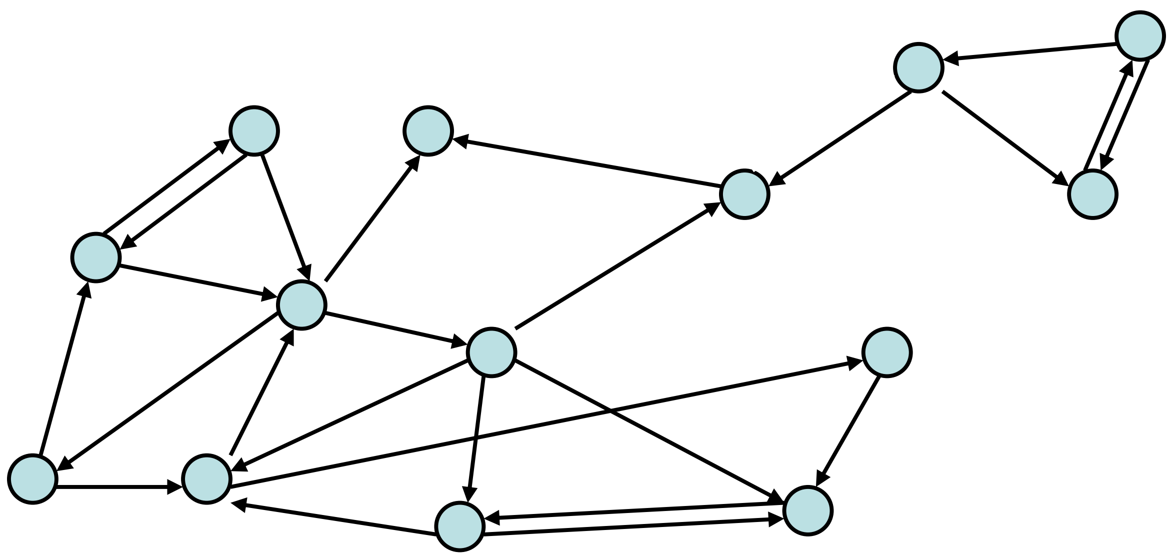
Last time: Relations & Composition

Let A and B be sets,
A **binary relation from** A **to** B is a subset of $A \times B$

Let A be a set,
A **binary relation on** A is a subset of $A \times A$

Last time: Directed Graphs

G = (V, E)   V – vertices
E – edges, ordered pairs of vertices



How Properties of Relations show up in Graphs

Let R be a relation on A.

R is **reflexive** iff $(a,a) \in R$ for every $a \in A$

R is **symmetric** iff $(a,b) \in R$ implies $(b, a) \in R$

R is **antisymmetric** iff $(a,b) \in R$ and $a \neq b$ implies $(b,a) \notin R$

R is **transitive** iff $(a,b) \in R$ and $(b, c) \in R$ implies $(a, c) \in R$

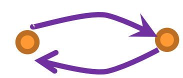
How Properties of Relations show up in Graphs

Let R be a relation on A.

R is **reflexive** iff $(a,a) \in R$ for every $a \in A$


at every node

R is **symmetric** iff $(a,b) \in R$ implies $(b, a) \in R$

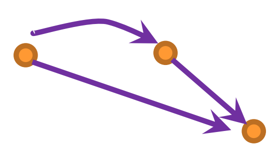

or


R is **antisymmetric** iff $(a,b) \in R$ and $a \neq b$ implies $(b,a) \notin R$


or

or


R is **transitive** iff $(a,b) \in R$ and $(b, c) \in R$ implies $(a, c) \in R$



Last time: Relation Composition

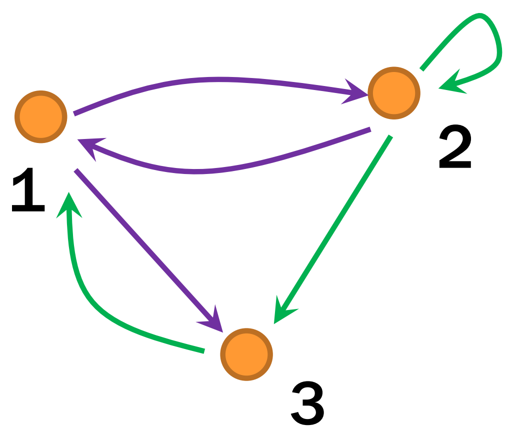
The composition of relation R and S, $R \circ S$ is the relation defined by:

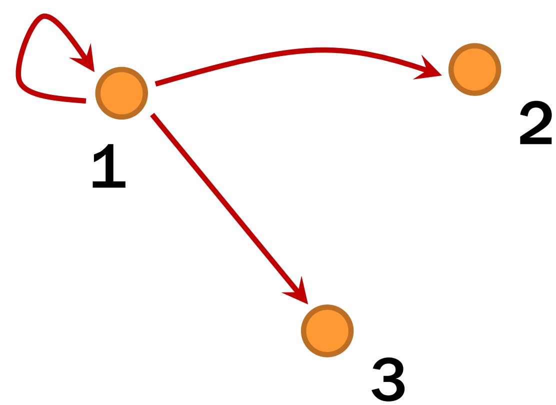
$R \circ S$ = $\{$ (a, c) | $\exists$ b such that (a,b)$\in R$ and (b,c)$\in S\}$

Last time: Relational Composition using Digraphs

If $\boldsymbol{S} = \{(2,2),(2,3),(3,1)\}$ and $\boldsymbol{R} = \{(1,2),(2,1),(1,3)\}$

Compute $\boldsymbol{R} \circ \boldsymbol{S}$

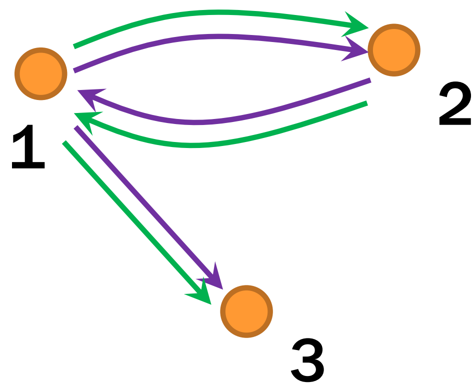


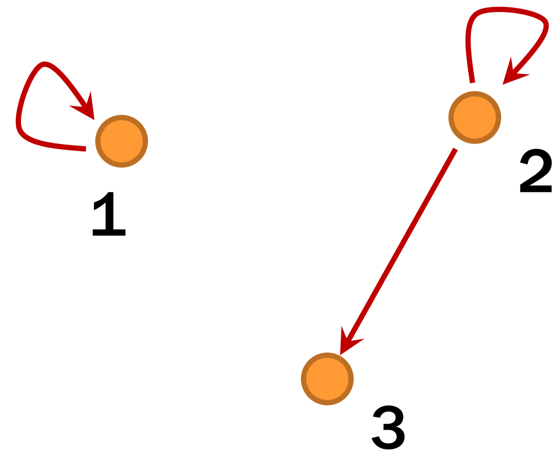


Relational Composition using Digraphs

If $\boldsymbol{R} = \{(\mathbf{1},\mathbf{2}),(\mathbf{2},\mathbf{1}),(\mathbf{1},\mathbf{3})\}$ and $\boldsymbol{R} = \{(\mathbf{1},\mathbf{2}),(\mathbf{2},\mathbf{1}),(\mathbf{1},\mathbf{3})\}$

Compute $\boldsymbol{R} \circ \boldsymbol{R}$





$(a,c) \in R \circ R = R^2$ iff $\exists b\,((a,b) \in R \land (b,c) \in R)$

Last time: Powers of a Relation

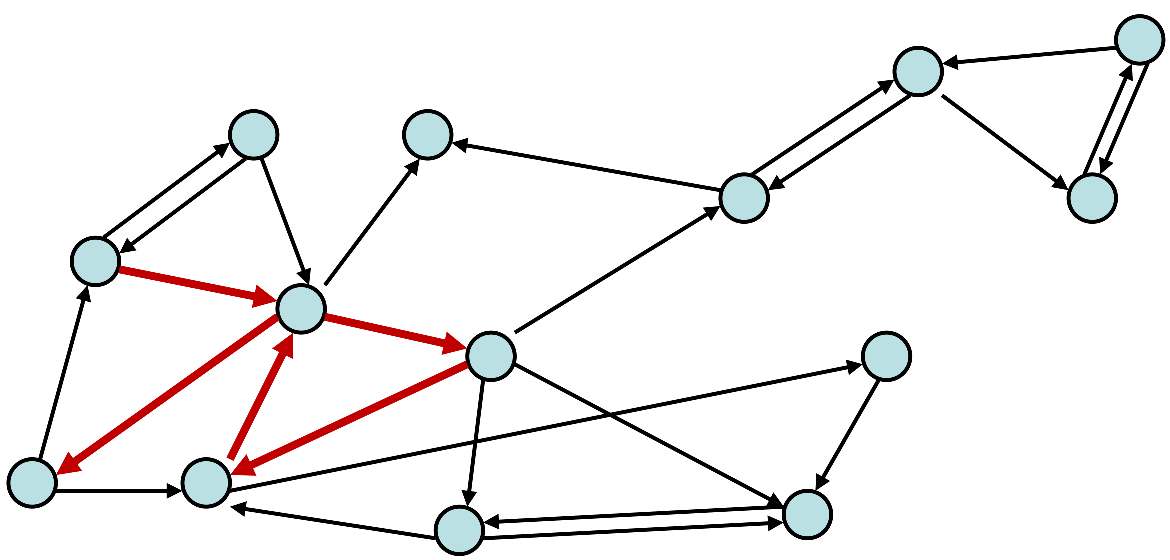
$\boldsymbol{R^0} \quad ::= \{(\boldsymbol{a}, \boldsymbol{a}) \mid \boldsymbol{a} \in \boldsymbol{A}\}$ “the equality relation on $\boldsymbol{A}$”

$\boldsymbol{R^{n+1}} \; ::= \boldsymbol{R^n} \circ \boldsymbol{R}$ for $\boldsymbol{n} \geq \boldsymbol{0}$

Last time: Paths in Directed Graphs

$G = (V, E)$ V – vertices
E – edges (relation on vertices)

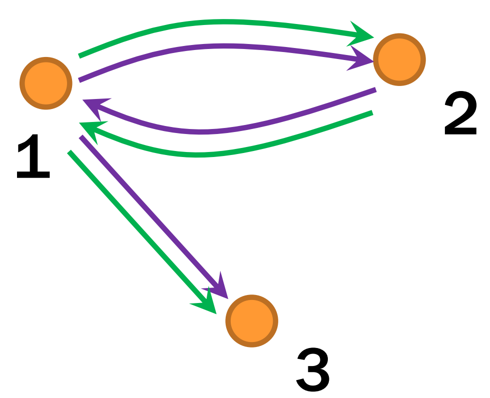
Path: $v_0, v_1, \ldots, v_k$ with each (v_i, v_{i+1}) in E

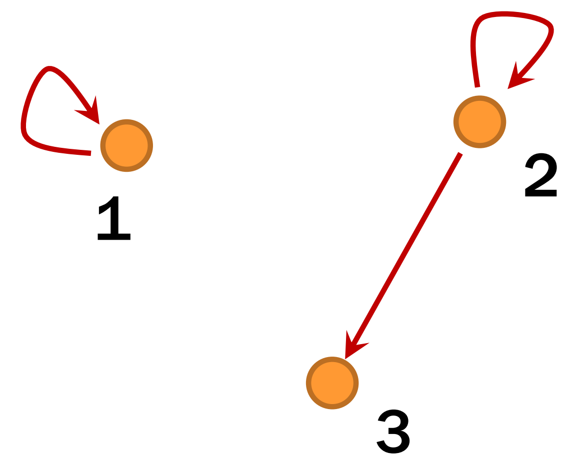


Relational Composition using Digraphs

If $\boldsymbol{R} = \{(1,2),(2,1),(1,3)\}$ and $\boldsymbol{R} = \{(1,2),(2,1),(1,3)\}$

Compute $\boldsymbol{R} \circ \boldsymbol{R}$





$(a,c) \in R \circ R = R^2$ iff $\exists b\ ((a,b) \in R \land (b,c) \in R)$

iff $\exists b$ such that a, b, c is a path

Paths in Relations

Def: The **length** of a path in a graph is the number of edges in it (counting repetitions if edge used > once).

Elements of R^0 correspond to paths of length 0.
Elements of $R^1 = R$ are paths of length 1.
Elements of R^2 are paths of length 2.
...

Paths in Relations

Def: The **length** of a path in a graph is the number of edges in it (counting repetitions if edge used > once).

Let $\boldsymbol{R}$ be a relation on a set $\boldsymbol{A}$.

There is a path of length $\boldsymbol{n}$ from **a** to **b** in the digraph for $\boldsymbol{R}$ if and only if (**a**,**b**) $\in \boldsymbol{R}^{\boldsymbol{n}}$

Connectivity In Graphs

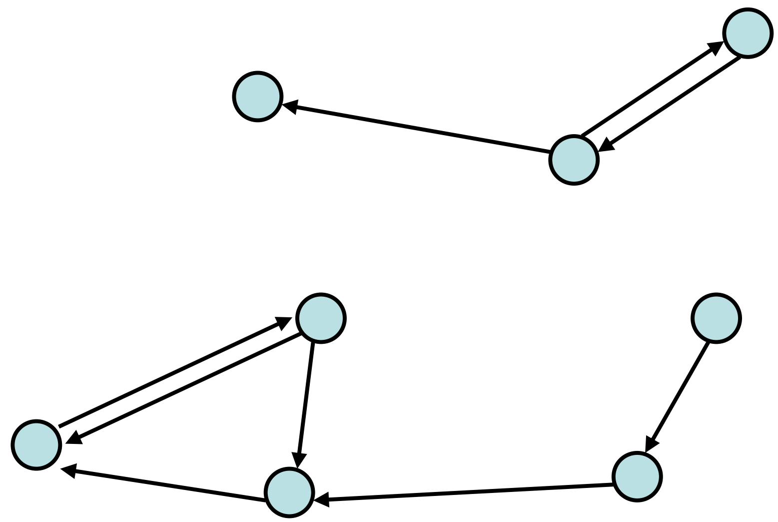
Def: Two vertices in a graph are **connected** iff there is a path between them.

Let $\boldsymbol{R}$ be a relation on a set $\boldsymbol{A}$. The **connectivity** relation $\boldsymbol{R}^*$ consists of the pairs (a,b) such that there is a path from a to b in $\boldsymbol{R}$.

$$\boldsymbol{R}^* = \bigcup_{k=0}^{\infty} \boldsymbol{R}^k$$

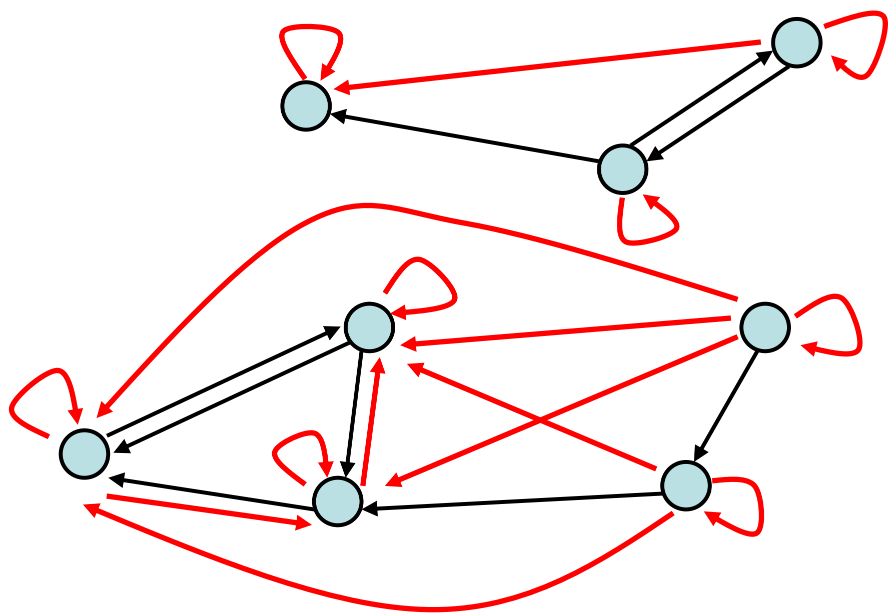
Note: Rosen text uses the wrong definition of this quantity. What the text defines (ignoring k=0) is usually called R^+

Transitive-Reflexive Closure



Add the **minimum possible** number of edges to make the relation transitive and reflexive.

Transitive-Reflexive Closure



Relation with the **minimum possible** number of **extra edges** to make the relation both transitive and reflexive.

The **transitive-reflexive closure** of a relation $\boldsymbol{R}$ is the connectivity relation $\boldsymbol{R}^*$

n-ary Relations

Let $\boldsymbol{A_1}, \boldsymbol{A_2}, \ldots, \boldsymbol{A_n}$ be sets. An $\boldsymbol{n}$**-ary** relation on these sets is a subset of $\boldsymbol{A_1} \times \boldsymbol{A_2} \times \cdots \times \boldsymbol{A_n}$.

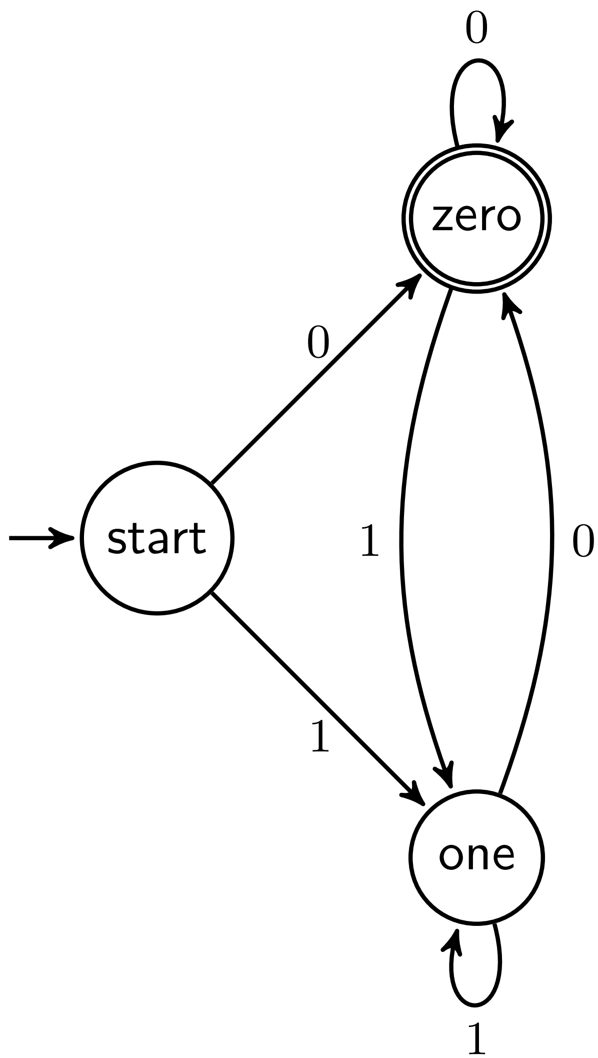
Relational Databases

STUDENT

Student_Name	ID_Number	Office	GPA
Knuth	328012098	022	4.00
Von Neuman	481080220	555	3.78
Russell	238082388	022	3.85
Einstein	238001920	022	2.11
Newton	1727017	333	3.61
Karp	348882811	022	3.98
Bernoulli	2921938	022	3.21

Back to Languages

Selecting strings using labeled graphs as “machines”

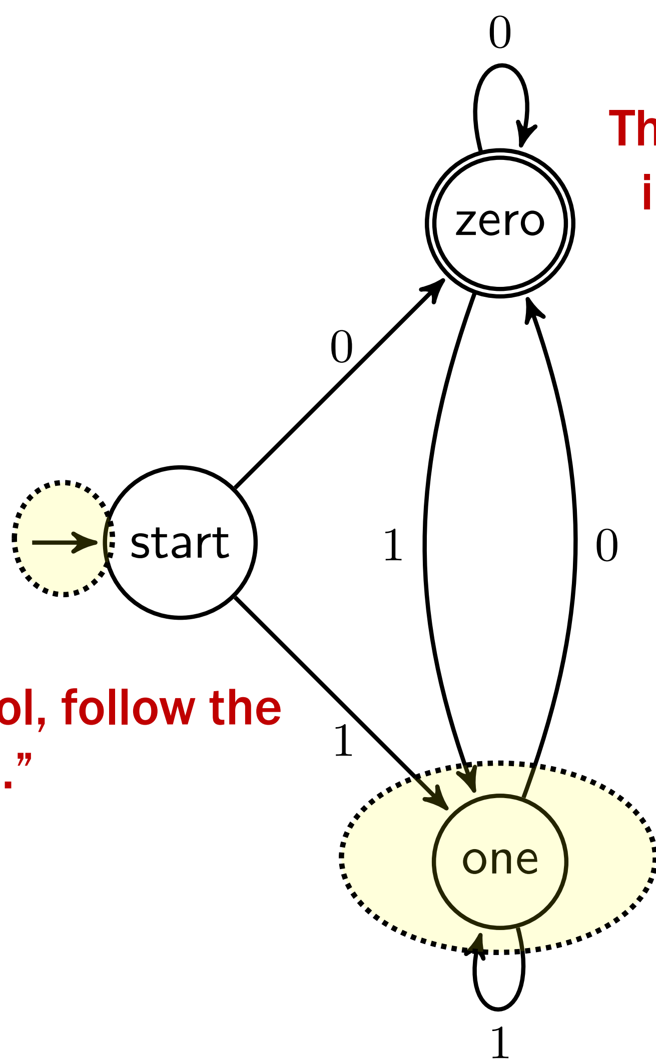


Finite State Machines

The “double circle” means “the input is good if it ends here”

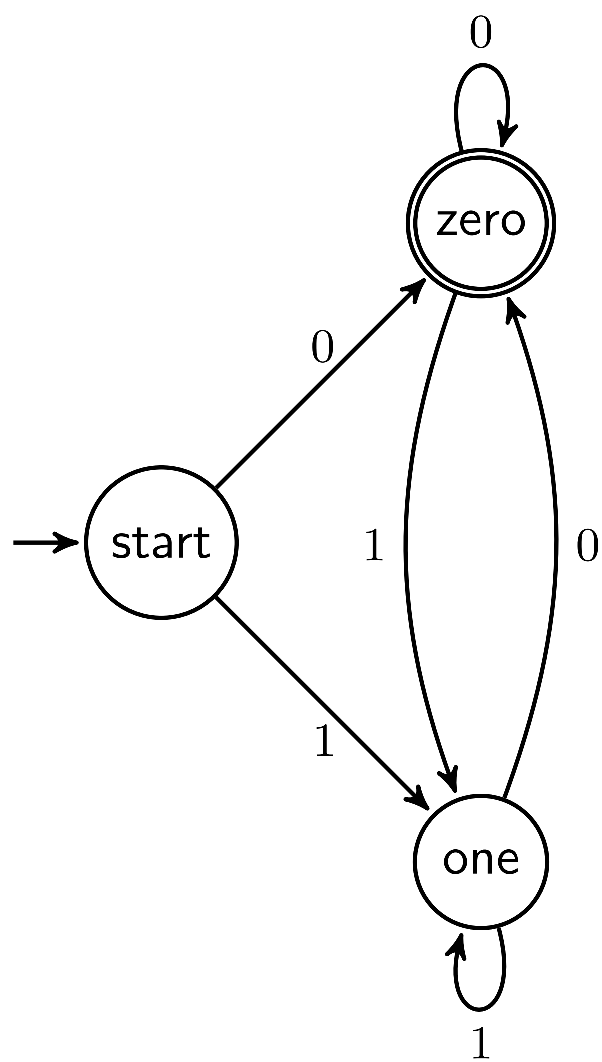
“Start here”

“If I get this symbol, follow the arrow…”

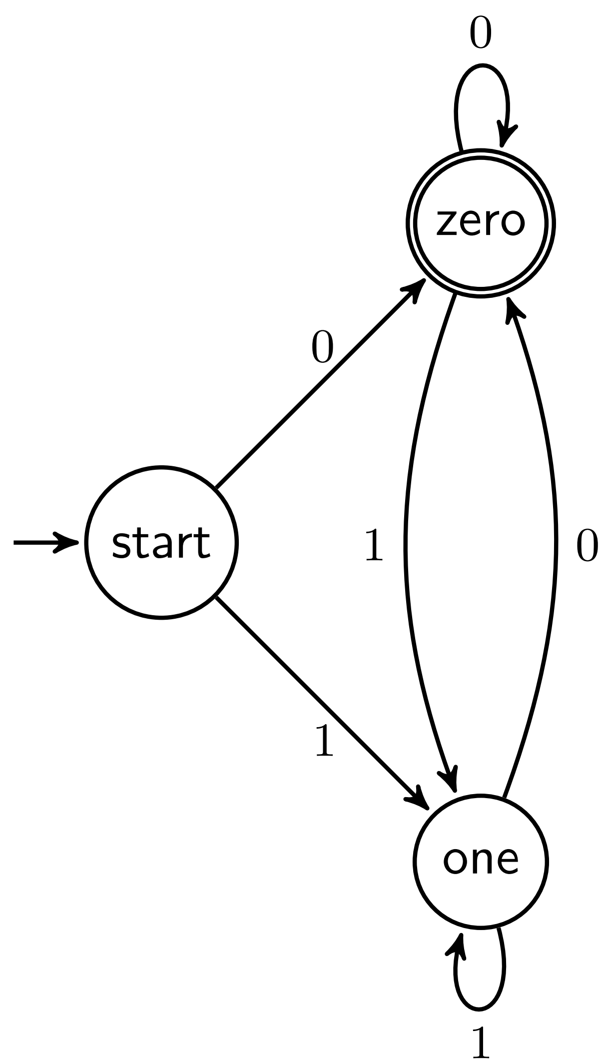


The circles are called “states”
We’re only in a single state at any point in time…

Which strings does this machine say are OK?



Which strings does this machine say are OK?

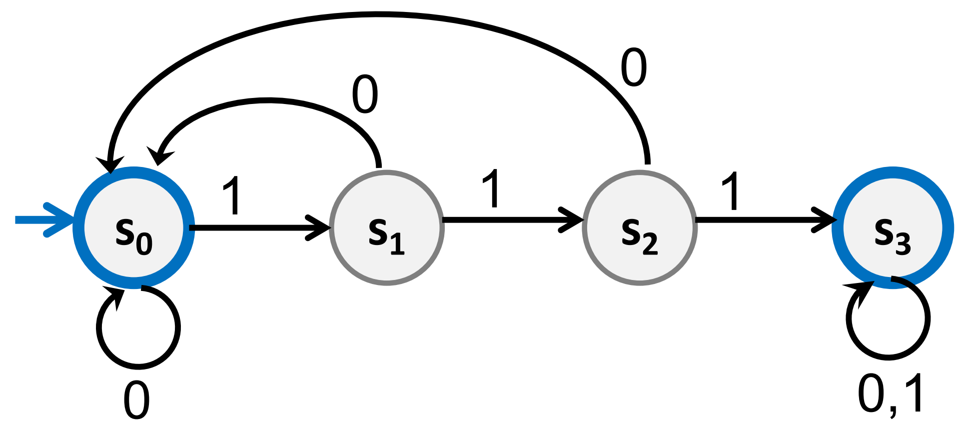


The set of all binary strings that end in 0

Finite State Machines

- States
- Transitions on input symbols
- Start state and final states
- The "language recognized" by the machine is the set of strings that reach a final state from the start

Old State	0	1
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_0	s_3
s_3	s_3	s_3

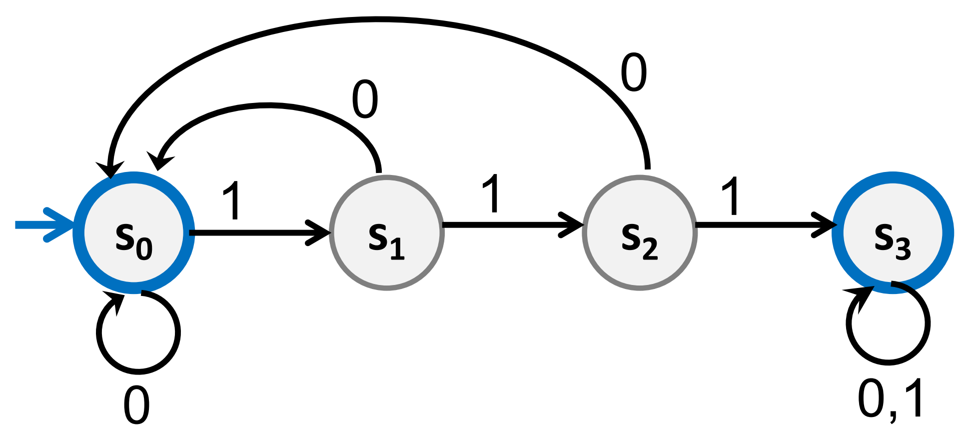


Finite State Machines

- Each machine designed for strings over some fixed alphabet Σ.

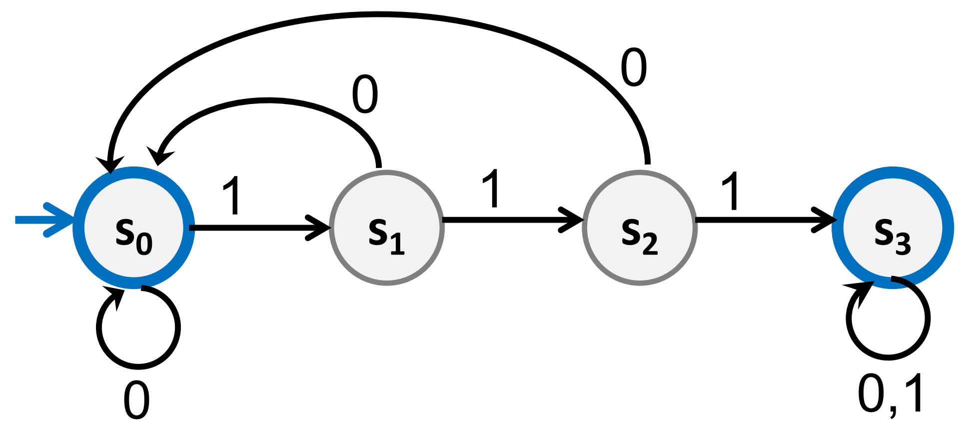
- Must have a transition defined from each state for ***every*** symbol in Σ.

Old State	0	1
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_0	s_3
s_3	s_3	s_3



What language does this machine recognize?

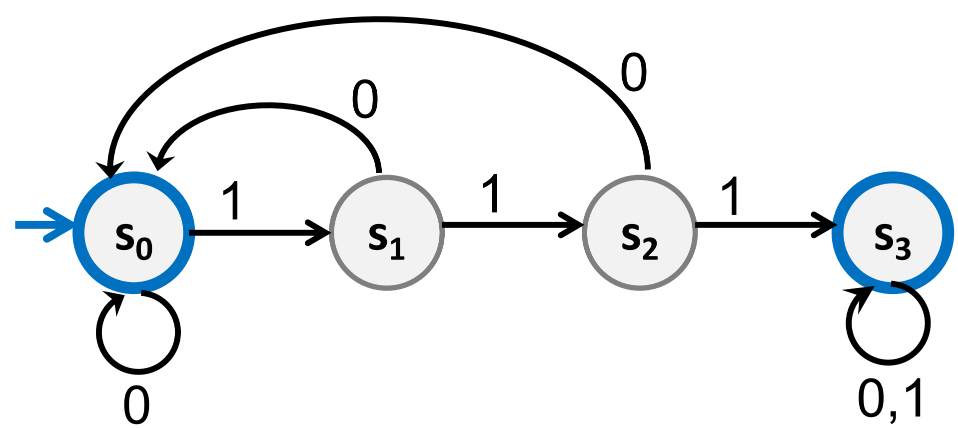
Old State	0	1
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_0	s_3
s_3	s_3	s_3



What language does this machine recognize?

The set of all binary strings that contain 111 or don't end in 1

Old State	0	1
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_0	s_3
s_3	s_3	s_3



Applications of FSMs (a.k.a. Finite Automata)

- Implementation of regular expression matching in programs like `grep`
- Control structures for sequential logic in digital circuits
- Algorithms for communication and cache-coherence protocols
  - Each agent runs its own FSM
- Design specifications for reactive systems
  - Components are communicating FSMs

Applications of FSMs (a.k.a. Finite Automata)

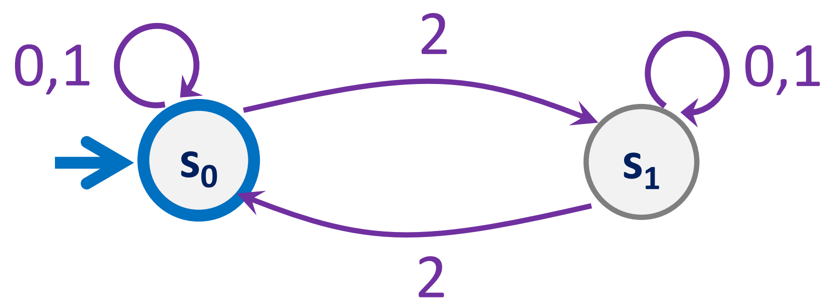
- Formal verification of systems
  - Is an unsafe state reachable?
- Computer games
  - FSMs implement non-player characters
- Minimization algorithms for FSMs can be extended to more general models used in
  - Text prediction
  - Speech recognition

Strings over {0, 1, 2}

M_1: Strings with an even number of 2's

Strings over {0, 1, 2}

M_1: Strings with an even number of 2's



Strings over {0, 1, 2}

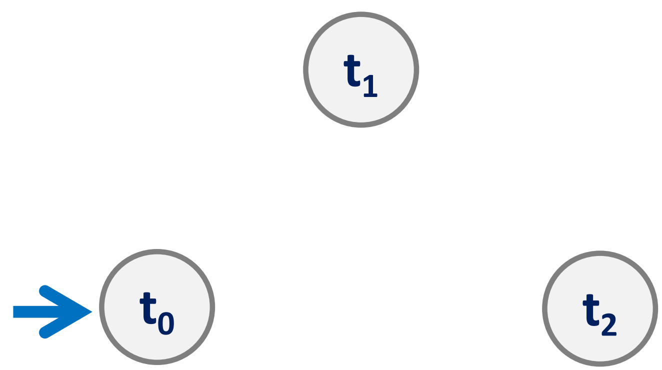
M_2: Strings where the sum of digits mod 3 is 0

FSM as abstraction of Java code

```
boolean sumCongruentToZero(String str) {
    int sum = 0;
    for (int i = 0; i < str.length(); i++) {
        if (str.charAt(i) == '2')
            sum = (sum + 2) % 3;
        if (str.charAt(i) == '1')
            sum = (sum + 1) % 3;
    }
    return sum == 0;
}
```

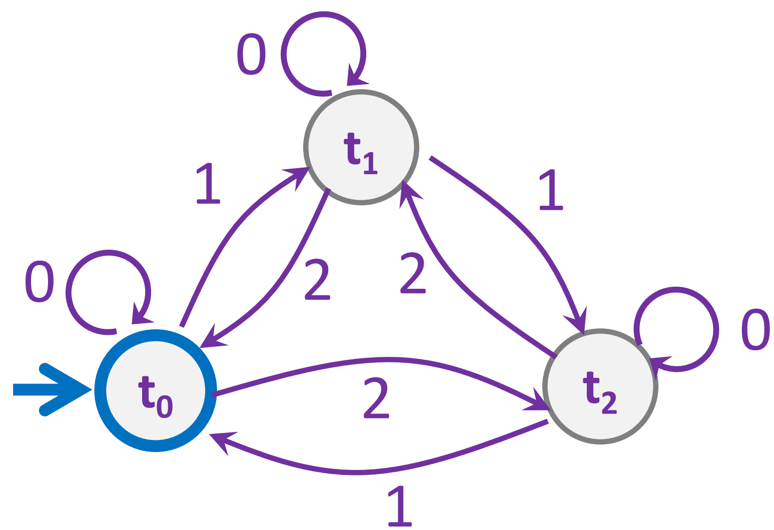
Strings over {0, 1, 2}

M_2: Strings where the sum of digits mod 3 is 0



Strings over {0, 1, 2}

M_2: Strings where the sum of digits mod 3 is 0



FSM as abstraction of Java code

```
boolean sumCongruentToZero(String str) {
    int sum = 0;  // state
    for (int i = 0; i < str.length(); i++) {
        if (str.charAt(i) == '2')
            sum = (sum + 2) % 3;
        if (str.charAt(i) == '1')
            sum = (sum + 1) % 3;
    }
    return sum == 0;
}
```

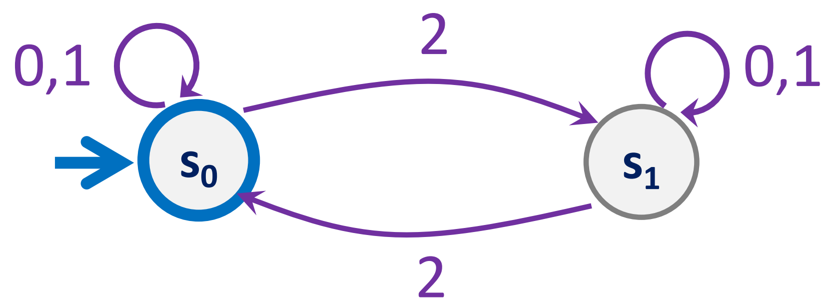
FSMs can model Java code with a **finite** number of **fixed-size** variables that makes **one pass** through input

FSM to Java code

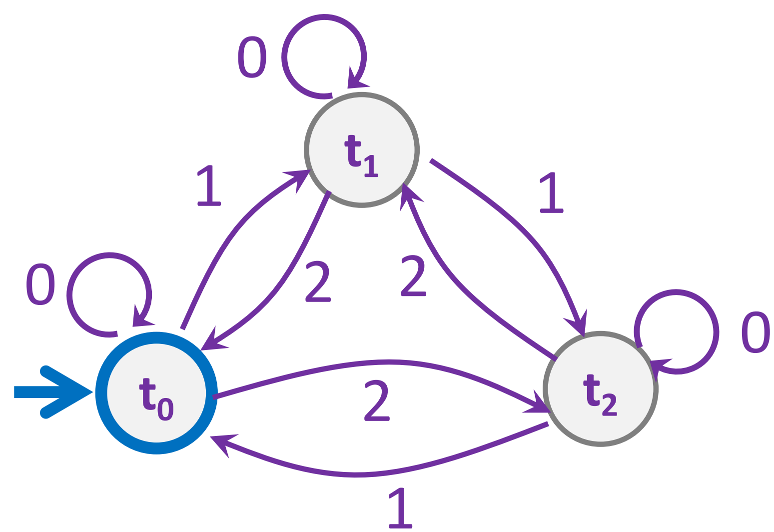
```java
int[][] TRANSITION = {...};

boolean sumCongruentToZero(String str) {
    int state = 0;
    for (int i = 0; i < str.length(); i++) {
        int d = str.charAt(i) - '0';
        state = TRANSITION[state][d];
    }
    return state == 0;
}
```

Strings over {0, 1, 2}

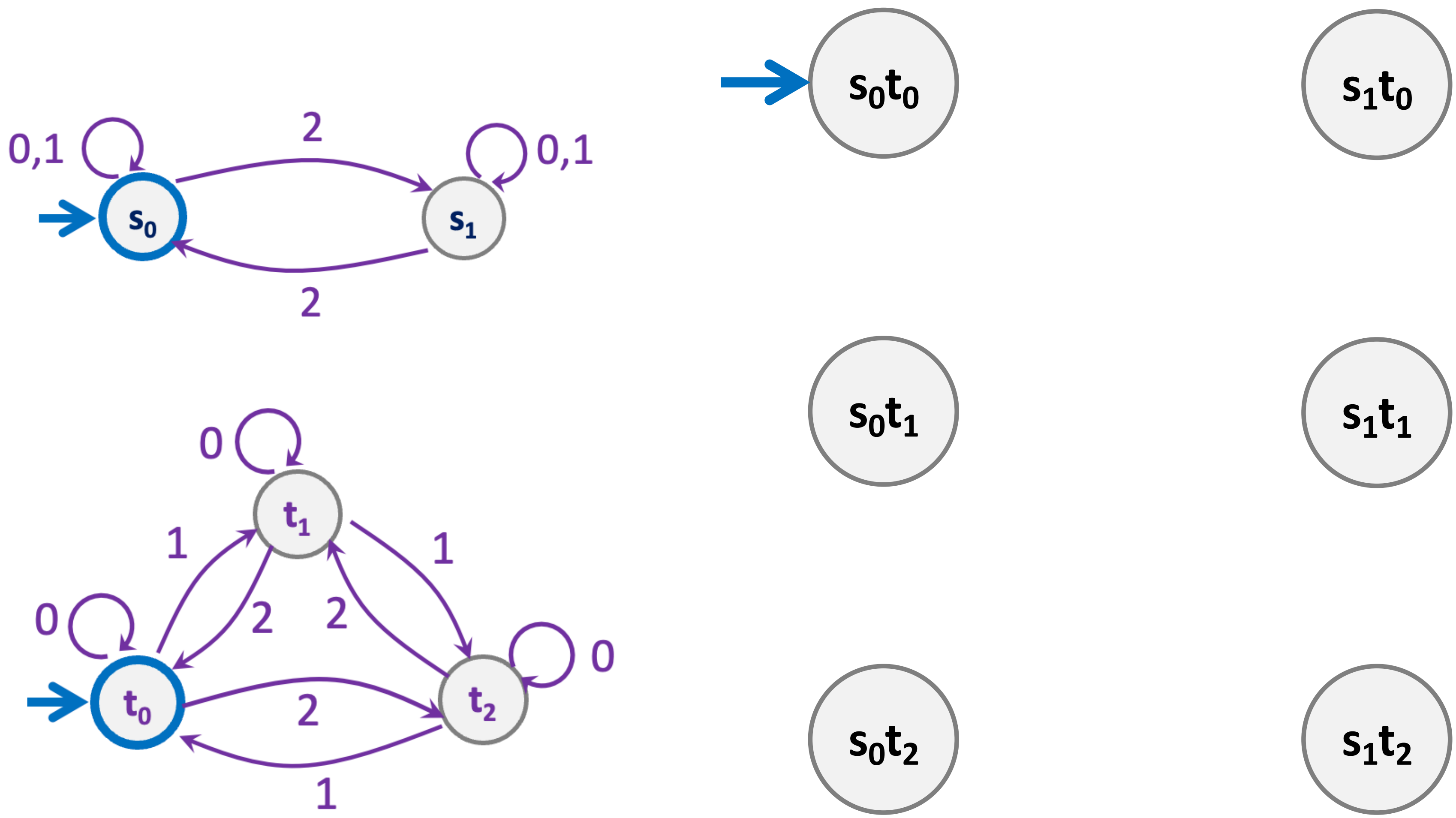
M_1: Strings with an even number of 2's



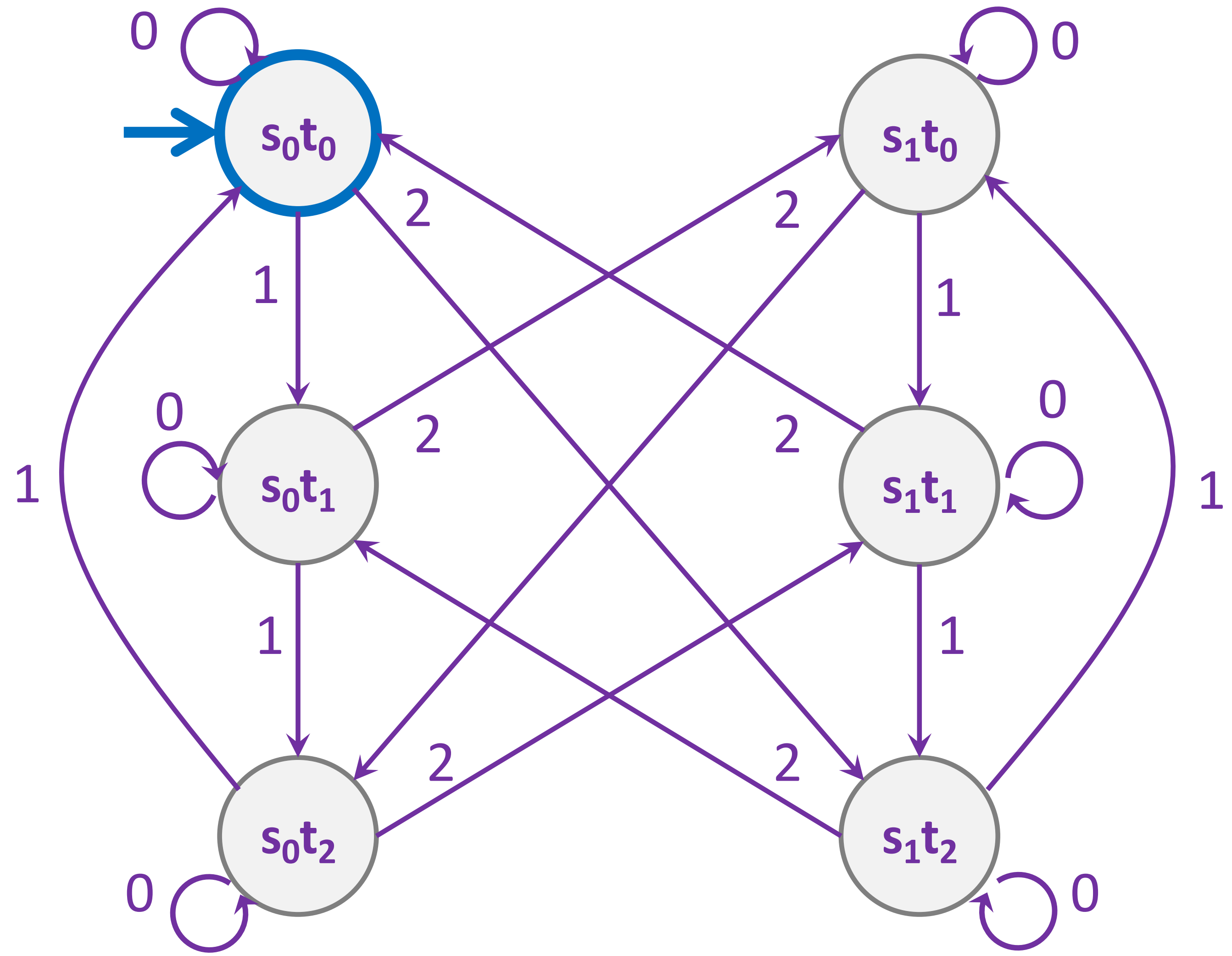
M_2: Strings where the sum of digits mod 3 is 0



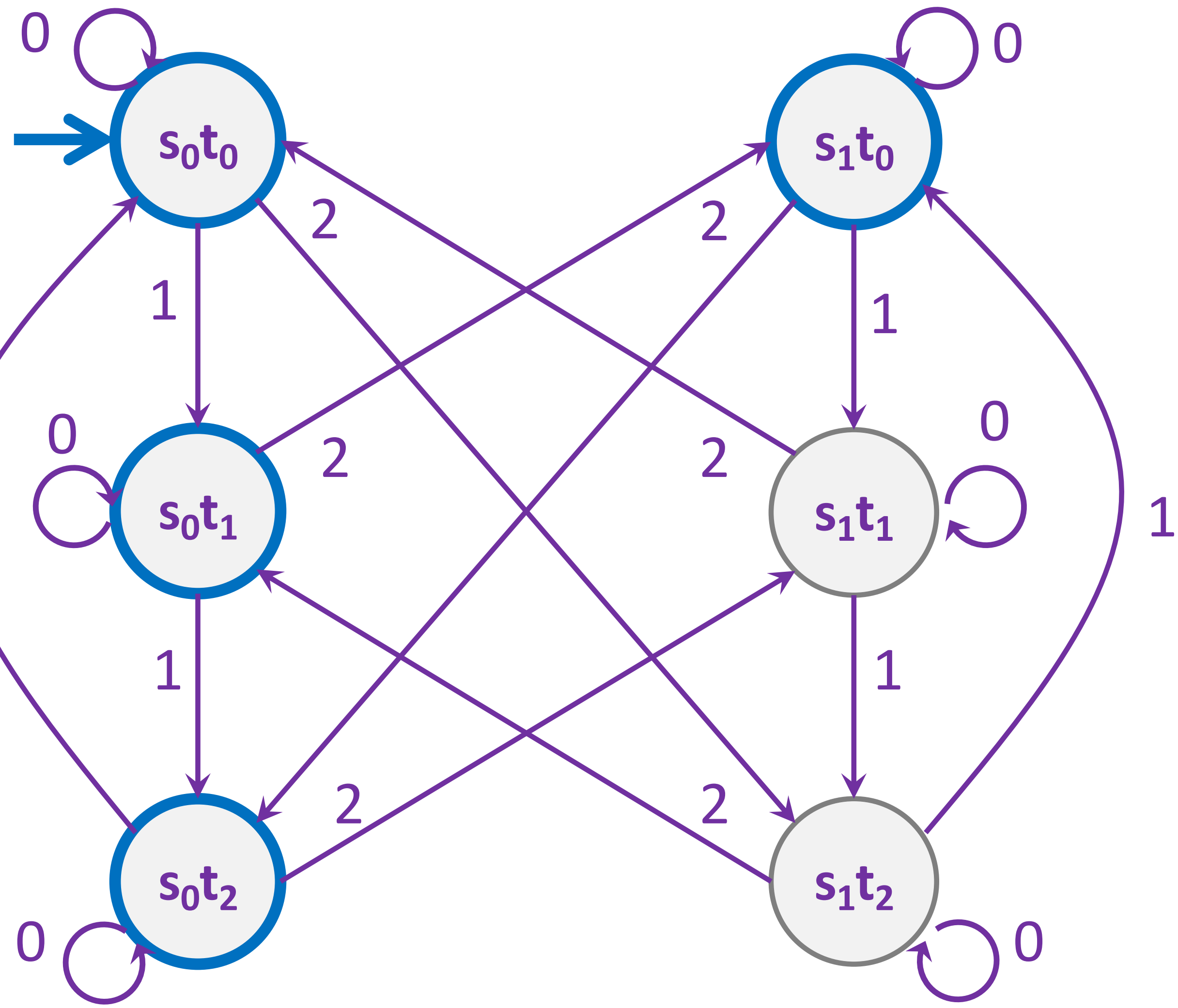
Strings over {0,1,2} w/ even number of 2's AND mod 3 sum 0



Strings over {0,1,2} w/ even number of 2's AND mod 3 sum 0



Strings over {0,1,2} w/ even number of 2's OR mod 3 sum 0



Strings over {0,1,2} w/ even number of 2's XOR mod 3 sum 0