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

Let A be a set,
**A binary relation on** A **is a subset of** A × 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 $A$.

- **Reflexive**: $R$ is reflexive iff $(a,a) \in R$ for every $a \in A$
- **Symmetric**: $R$ is symmetric iff $(a,b) \in R$ implies $(b,a) \in R$
- **Antisymmetric**: $R$ is antisymmetric iff $(a,b) \in R$ and $a \neq b$ implies $(b,a) \not\in R$
- **Transitive**: $R$ 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:

$$R \circ S = \{ (a, c) \mid \exists b \text{ such that } (a, b) \in R \text{ and } (b, c) \in S \}$$
If $S = \{(2, 2), (2, 3), (3, 1)\}$ and $R = \{(1, 2), (2, 1), (1, 3)\}$
Compute $R \circ S$
Relational Composition using Digraphs

If \( R = \{(1, 2), (2, 1), (1, 3)\} \) and \( R = \{(1, 2), (2, 1), (1, 3)\} \)
Compute \( R \circ R \)

\[
(a, c) \in R \circ R = R^2 \iff \exists b ((a, b) \in R \land (b, c) \in R)
\]
Last time: Powers of a Relation

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

\[ R^{n+1} := R^n \circ R \quad \text{for } n \geq 0 \]
Last time: Paths in Directed Graphs

\[ G = (V, E) \]
\[ V \text{ – vertices} \]
\[ E \text{ – edges} \quad \text{(relation on vertices)} \]

**Path:** \( v_0, v_1, ..., v_k \text{ with each } (v_i, v_{i+1}) \text{ in } E \)
Relational Composition using Digraphs

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

$$(a, c) \in R \circ R = R^2 \iff \exists b \ ( (a, b) \in R \land (b, c) \in R)$$
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 $R$ be a relation on a set $A$.
There is a path of length $n$ from $a$ to $b$ in the digraph for $R$ if and only if $(a,b) \in R^n$.
Connectivity In Graphs

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

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

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

Note: Rosen text uses the wrong definition of this quantity. What the text defines (ignoring $k=0$) is usually called $R^+$.
Add the minimum possible number of edges to make the relation transitive and reflexive.
Relation with the **minimum possible** number of **extra edges** to make the relation both transitive and reflexive.

The **transitive-reflexive closure** of a relation $R$ is the connectivity relation $R^*$.
Let $A_1, A_2, ..., A_n$ be sets. An $n$-ary relation on these sets is a subset of $A_1 \times A_2 \times \cdots \times A_n$. 
## Relational Databases

### STUDENT

<table>
<thead>
<tr>
<th>Student_Name</th>
<th>ID_Number</th>
<th>Office</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knuth</td>
<td>328012098</td>
<td>022</td>
<td>4.00</td>
</tr>
<tr>
<td>Von Neuman</td>
<td>481080220</td>
<td>555</td>
<td>3.78</td>
</tr>
<tr>
<td>Russell</td>
<td>238082388</td>
<td>022</td>
<td>3.85</td>
</tr>
<tr>
<td>Einstein</td>
<td>238001920</td>
<td>022</td>
<td>2.11</td>
</tr>
<tr>
<td>Newton</td>
<td>1727017</td>
<td>333</td>
<td>3.61</td>
</tr>
<tr>
<td>Karp</td>
<td>348882811</td>
<td>022</td>
<td>3.98</td>
</tr>
<tr>
<td>Bernoulli</td>
<td>2921938</td>
<td>022</td>
<td>3.21</td>
</tr>
</tbody>
</table>
AND NOW BACK TO OUR REGULARLY SCHEDULED PROGRAMMING
Selecting strings using labeled graphs as “machines”
Finite State Machines

“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...

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

<table>
<thead>
<tr>
<th>Old State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$s_0$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_0$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_0$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
</tr>
</tbody>
</table>
Finite State Machines

- Each machine designed for strings over some fixed alphabet $\Sigma$.

- Must have a transition defined from each state for every symbol in $\Sigma$.

<table>
<thead>
<tr>
<th>Old State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$s_0$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_0$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_0$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
</tr>
</tbody>
</table>
What language does this machine recognize?

<table>
<thead>
<tr>
<th>Old State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$s_0$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_0$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_0$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
</tr>
</tbody>
</table>
What language does this machine recognize?

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

<table>
<thead>
<tr>
<th>Old State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₀</td>
<td>S₀</td>
<td>S₁</td>
</tr>
<tr>
<td>S₁</td>
<td>S₀</td>
<td>S₂</td>
</tr>
<tr>
<td>S₂</td>
<td>S₀</td>
<td>S₃</td>
</tr>
<tr>
<td>S₃</td>
<td>S₃</td>
<td>S₃</td>
</tr>
</tbody>
</table>
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\}

M₁: Strings with an even number of 2’s

\[ \rightarrow s_0 \quad s_1 \]
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
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\}

\(M_2: \text{Strings where the sum of digits mod } 3 \text{ is } 0\)
Strings over \{0, 1, 2\}

\(M_2\): Strings where the sum of digits mod 3 is 0
FSM as abstraction of Java code

```java
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\}

**M₁:** Strings with an even number of 2’s

```
\begin{align*}
\text{M₁: Strings with an even number of 2’s}
\end{align*}
```

**M₂:** Strings where the sum of digits mod 3 is 0

```
\begin{align*}
\text{M₂: Strings where the sum of digits mod 3 is 0}
\end{align*}
```
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 \(\{0,1,2\}\) with 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