Sets & Functions

- Sets
  - Membership:
    - $a \in \{a, b, c\}$
    - $a \in \{b, c\}$
    - $a \in \{b, \{a, b, c\}, d\}$
    - $\{a, b, c\} \in \{b, \{a, b, c\}, d\}$
Sets & Functions

- Sets
  - Membership:
    - $a \in \{a, b, c\}$
    - $a \notin \{b, c\}$
    - $a \notin \{b, \{a, b, c\}, d\}$
    - $\{a, b, c\} \in \{b, \{a, b, c\}, d\}$
FR-2: Sets & Functions

- Sets
  - Subset:
    - \{a\} \subseteq \{a, b, c\}
    - \{a\} \subseteq \{b, c, \{a\}\}
    - \{a, b\} \subseteq \{a, b, c, d\}
    - \{a, b\} \subseteq \{a, b\}
    - \{\} \subseteq \{a, b, c, d\}
• **Sets**
  
  • **Subset:**
    
    - \( \{a\} \subseteq \{a, b, c\} \)
    - \( \{a\} \nsubseteq \{b, c, \{a\}\} \)
    - \( \{a, b\} \subseteq \{a, b, c, d\} \)
    - \( \{a, b\} \subseteq \{a, b\} \)
    - \( \emptyset \subseteq \{a, b\} \)
    - \( \emptyset \subseteq \{a, b, c, d\} \)
Sets & Functions

- Sets
  - Cross Product:
    - \( A \times B = \{(a, b) : a \in A, b \in B\} \)
    - \( \{a, b\} \times \{a, b\} = \)
    - \( \{a, b\} \times \{\{a, b\}\} = \)
Sets & Functions

- **Sets**
  
  - **Cross Product:**
    
    - $A \times B = \{(a, b) : a \in A, b \in B\}$
    - $\{a, b\} \times \{a, b\} = \{(a, a), (a, b), (b, a), (b, b)\}$
    - $\{a, b\} \times \{\{a, b\}\} = \{(a, \{a, b\}), (b, \{a, b\})\}$
Sets

Power Set:

- $2^A = \{S : S \subseteq A\}$
- $2\{a,b\} =$
- $2\{a\} =$
- $2^2\{a\} =$
Sets & Functions

- Sets
  - Power Set:
    - $2^A = \{ S : S \subseteq A \}$
    - $2^{\{a,b\}} = \{\{\}, \{a\}, \{b\}, \{a, b\}\}$
    - $2^{\{a\}} = \{\{\}, \{a\}\}$
    - $2^{2^{\{a\}}} = \{\{\}, \{\{\}\}, \{\{a\}\}, \{\}, \{a\}\}$
\[ \Pi \text{ is a partition of } S \text{ if:} \]

- \( \Pi \subseteq 2^S \)
- \( \emptyset \notin \Pi \)
- \( \forall (X, Y \in \Pi), X \neq Y \implies X \cap Y = \{\} \)
- \( \bigcup \Pi = S \)

\{\{a, c\}, \{b, d, e\}, \{f\}\} \text{ is a partition of } \{a,b,c,d,e,f\}

\{\{a, b, c, d, e, f\}\} \text{ is a partition of } \{a,b,c,d,e,f\}

\{\{a, b, c\}, \{d, e, f\}\} \text{ is a partition of } \{a,b,c,d,e,f\}
In other words, a partition of a set $S$ is just a division of the elements of $S$ into 1 or more groups.

- All the partitions of the set \{a, b, c\}?
In other words, a partition of a set $S$ is just a division of the elements of $S$ into 1 or more groups.

- All the partitions of the set $\{a, b, c\}$?
  - $\{\{a, b, c\}\}, \{\{a, b\}, \{c\}\}, \{\{a, c\}, \{b\}\}, \{\{a\}, \{b, c\}\}, \{\{a\}, \{b\}, \{c\}\}$
Relation

- A relation $R$ is a set of ordered pairs
- That’s *all* that a relation is
- Relation Graphs
FR-12: Sets & Functions

- Properties of Relations
  - Reflexive
  - Symmetric
  - Transitive
  - Antisymmetric

- Equivalence Relation: Reflexive, Symmetric, Transitive

- Partial Order: Reflexive, Antisymmetric, Transitive

- Total Order: Partial order, for each $a, a' \in A$, either $(a, a') \in R$ or $(a', a) \in R$
FR-13: Sets & Functions

- What does a graph of an Equivalence relation look like?
- What does a graph of a Total Order look like?
- What does a graph of a Partial Order look like?
A set $A \subseteq B$ is closed under a relation $R \subseteq ((B \times B) \times B)$ if:

- $a_1, a_2 \in A \land ((a_1, a_2), c) \in R \implies c \in A$

That is, if $a_1$ and $a_2$ are both in $A$, and $((a_1, a_2), c)$ is in the relation, then $c$ is also in $A$.

- $\mathbb{N}$ is closed under addition
- $\mathbb{N}$ is not closed under subtraction or division
FR-15: Closure

• Relations are also sets (of ordered pairs)
• We can talk about a relation \( R \) being closed over another relation \( R' \)
  • Each element of \( R' \) is an ordered triple of ordered pairs!
Relations are also sets (of ordered pairs)

We can talk about a relation $R$ being closed over another relation $R'$

- Each element of $R'$ is an ordered triple of ordered pairs!

Example:

- $R \subseteq A \times A$
- $R' = \{(((a, b), (b, c)), (a, c)) : a, b, c \in A\}$
- If $R$ is closed under $R'$, then ...
FR-17: Closure

- Relations are also sets (of ordered pairs)
- We can talk about a relation $R$ being closed over another relation $R'$
  - Each element of $R'$ is an ordered triple of ordered pairs!
- Example:
  - $R \subseteq A \times A$
  - $R' = \{(((a, b), (b, c)), (a, c)) : a, b, c \in A\}$
  - If $R$ is closed under $R'$, then $R$ is transitive!
FR-18: Closure

• Reflexive closure of a relation $R \subseteq A \times A$ is the smallest possible superset of $R$ which is reflexive
  • Add self-loop to every node in relation
  • Add $(a,a)$ to $R$ for every $a \in A$

• Transitive Closure of a relation $R \subseteq A \times A$ is the smallest possible superset of $R$ which is transitive
  • Add direct link for every path of length 2.
  • $\forall (a, b, c \in A)$ if $(a, b) \in R \land (b, c) \in R$ add $(a, c)$ to $R$.

(examples on board)
FR-19: Sets & Functions

- Functions
  - Relation $R$ over $A \times B$
  - For each $a \in A$:
    - Exactly one element $(x, y) \in R$ with $x = a$
For a function $f$ over $(A \times A)$, what does the graph look like?

For a function $f$ over $(A \times B)$, what does the graph look like?
Functions

- one-to-one: $f(a) \neq f(a')$ when $a \neq a'$ (nothing is mapped to twice)
- onto: for each $b \in B$, $\exists a$ such that $f(a) = b$ (everything is mapped to)
- bijection: Both one-to-one and onto
For a function $f$ over $(A \times B)$

- What does the graph look like for a one-to-one function?
- What does the graph look like for an onto function?
- What does the graph look like for a bijection?
FR-23: Sets & Functions

- Infinite sets
  - Countable, Countably infinite
    - Bijection with the Natural Numbers
  - Uncountable, uncountable infinite
    - Infinite
    - No bijection with the Natural Numbers
• We can show that a set is countable infinite by giving a bijection between that set and the natural numbers

• Same thing as imposing an ordering on an infinite set
A set is countable infinite (or just countable) if it is equinumerous with $\mathbb{N}$.

• Even elements of $\mathbb{N}$?
A set is countable infinite (or just countable) if it is equinumerous with $\mathbb{N}$.

- Even elements of $\mathbb{N}$?
- $f(x) = 2x$
Countable Sets

A set is countable infinite (or just countable) if it is equinumerous with \( \mathbb{N} \).

- Integers \((\mathbb{Z})\)?
A set is countable infinite (or just countable) if it is equinumerous with \( \mathbb{N} \).

- Integers \((\mathbb{Z})\)?
- \( f(x) = \left\lceil \frac{x}{2} \right\rceil \cdot (-1)^x \)

-4 -3 -2 -1 0 1 2 3 4

...
A set is countable infinite (or just countable) if it is equinumerous with \( \mathbb{N} \).

- Union of 3 (disjoint) countable sets A, B, C?
A set is countable infinite (or just countable) if it is equinumerous with \( \mathbb{N} \).

Union of 3 (disjoint) countable sets \( A, B, C \)?

\[
f(x) = \begin{cases} 
  a \frac{x}{3} & \text{if } x \mod 3 = 0 \\
  b \frac{x-1}{3} & \text{if } x \mod 3 = 1 \\
  c \frac{x-2}{3} & \text{if } x \mod 3 = 2 
\end{cases}
\]
FR-31: Countable Sets

- A set is countable infinite (or just countable) if it is equinumerous with $\mathbb{N}$.

- $\mathbb{N} \times \mathbb{N}$?

  (0,0)  (0,1)  (0,2)  (0,3)  (0,4)  ...
  (1,0)  (1,1)  (1,2)  (1,3)  (1,4)  ...
  (2,0)  (2,1)  (2,2)  (2,3)  (2,4)  ...
  (3,0)  (3,1)  (3,2)  (3,3)  (3,4)  ...
  (4,0)  (4,1)  (4,2)  (4,3)  (4,4)  ...
  \vdots  \vdots  \vdots  \vdots  \vdots  \vdots  \

FR-32: **Countable Sets**

- A set is *countable infinite* (or just *countable*) if it is equinumerous with \( \mathbb{N} \).

<table>
<thead>
<tr>
<th>0,0</th>
<th>0,1</th>
<th>0,2</th>
<th>0,3</th>
<th>0,4</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,0</td>
<td>1,1</td>
<td>1,2</td>
<td>1,3</td>
<td>1,4</td>
<td>...</td>
</tr>
<tr>
<td>2,0</td>
<td>2,1</td>
<td>2,2</td>
<td>2,3</td>
<td>2,4</td>
<td>...</td>
</tr>
<tr>
<td>3,0</td>
<td>3,1</td>
<td>3,2</td>
<td>3,3</td>
<td>3,4</td>
<td>...</td>
</tr>
<tr>
<td>4,0</td>
<td>4,1</td>
<td>4,2</td>
<td>4,3</td>
<td>4,4</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- \( f((x, y)) = \frac{(x+y)(x+y+1)}{2} + x \)
A set is countable infinite (or just countable) if it is equinumerous with \( \mathbb{N} \).

- Real numbers between 0 and 1 (exclusive)?
Proof by contradiction

• Assume that $\mathbb{R}$ between 0 and 1 (exclusive) is countable
  • (that is, assume that there is some bijection from $\mathbb{N}$ to $\mathbb{R}$ between 0 and 1)

• Show that this leads to a contradiction
  • Find some element of $\mathbb{R}$ between 0 and 1 that is not mapped to by any element in $\mathbb{N}$
• Assume that there is some bijection from $\mathbb{N}$ to $\mathbb{R}$ between 0 and 1

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.3412315569...</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.0123506541...</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.1143216751...</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.2839143215...</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.2311459412...</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.8381441234...</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.7415296413...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Assume that there is some bijection from $\mathbb{N}$ to $\mathbb{R}$ between 0 and 1

<table>
<thead>
<tr>
<th>$n$</th>
<th>$r_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.3412315569...</td>
</tr>
<tr>
<td>1</td>
<td>0.0123506541...</td>
</tr>
<tr>
<td>2</td>
<td>0.1143216751...</td>
</tr>
<tr>
<td>3</td>
<td>0.2839143215...</td>
</tr>
<tr>
<td>4</td>
<td>0.2311459412...</td>
</tr>
<tr>
<td>5</td>
<td>0.8381441234...</td>
</tr>
<tr>
<td>6</td>
<td>0.7415296413...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Consider: 0.425055...
FR-37: **Formal Languages**

- **Alphabet** $\Sigma$: Set of symbols
  - \{0, 1\}, \{a, b, c\}, etc
- **String** $w$: Sequence of symbols
  - *cat*, *dog*, *firehouse* etc
- **Language** $L$: Set of strings
  - \{*cat, dog, firehouse*\}, \{a, aa, aaa, ...\}, etc
- **Language class**: Set of Languages
  - Regular languages, $\text{P}$, $\text{NP}$, etc.
• Language Hierarchy.
Regular Expressions

- Regular expressions are a way to describe formal languages
- Regular expressions are defined recursively
  - Base case – simple regular expressions
  - Recursive case – how to build more complex regular expressions from simple regular expressions
\textbf{FR-40: Regular Expressions}

- $\epsilon$ is a regular expression, representing $\{\epsilon\}$
- $\emptyset$ is a regular expression, representing $\{\}$
- $\forall a \in \Sigma$, $a$ is a regular expression representing $\{a\}$
- If $r_1$ and $r_2$ are regular expressions, then $(r_1r_2)$ is a regular expression
  - $L[(r_1r_2)] = L[r_1] \circ L[r_2]$
- If $r_1$ and $r_2$ are regular expressions, then $(r_1 + r_2)$ is a regular expression
  - $L[(r_1 + r_2)] = L[r_1] \cup L[r_2]$
- If $r$ is regular expressions, then $(r^*)$ is a regular expression
  - $L[(r^*)] = (L[r])^*$
FR-41: r.e. Precedence

From highest to Lowest:

Kleene Closure *
Concatenation
Alternation +

$ab^*c+e = (a(b^*)c) + e$

(We will still need parentheses for some regular expressions: $(a+b)(a+b)$)
FR-42: Regular Expressions

- Intuitive Reading of Regular Expressions
  - Concatenation == “is followed by”
  - + == “or”
  - * == “zero or more occurrences”
- (a+b)(a+b)(a+b)
- (a+b)*
- aab(aa)*
A language $L$ is regular if there exists a regular expression which generates it.

Give a regular expression for:

- All strings over $\{a, b\}$ that have an odd # of $a$’s.
A language $L$ is regular if there exists a regular expression which generates it.

Give a regular expression for:

- All strings over $\{a, b\}$ that have an odd # of $a$’s
  $b^* a (b^* ab^* a)^* b^*$

- All strings over $\{a, b\}$ that contain exactly two occurrences of $bb$ ($bbb$ counts as 2 occurrences!)
A language $L$ is regular if there exists a regular expression which generates it.

Give a regular expression for:

- All strings over $\{a, b\}$ that have an odd # of $a$’s
  $b^*a(b^*ab^*a)^*b^*$

- All strings over $\{a, b\}$ that contain exactly two occurrences of $bb$ ($bbb$ counts as 2 occurrences!)
  $a^*(baa^*)^*bb(aa^*b)^*aa^*bb(aa^*b)^*a^* + a^*(baa^*)^*bbb(aa^*b)^*a^*$
All strings over \{0, 1\} that begin (or end) with 11

All strings over \{0, 1\} that begin (or end) with 11, but not both
FR-47: Regular Languages

- All strings over \{0, 1\} that begin (or end) with 11
  - 11 \((0+1)^*\) 11 + 11

- All strings over \{0, 1\} that begin (or end) with 11, but not both
  - 11\((0+1)^*0\) + 11\((0+1)^*01\) + 0\((0+1)^*11\) + 10\((0+1^*)11\)
FR-48: Regular Languages

- Shortest string not described by following regular expressions?
  - $a^*b^*a^*b^*$
  - $a^*(ab)^*(ba)^*b^*a^*$
  - $a^*b^*(ab)^*b^*a^*$
FR-49: Regular Languages

- Shortest string not described by following regular expressions?
  - $a^*b^*a^*b^*$
    - baba
  - $a^*(ab)^*(ba)^*b^*a^*$
    - baab
  - $a^*b^*(ab)^*b^*a^*$
    - baab
FR-50: **Regular Languages**

- English descriptions of following regular expressions:
  - 
  - 
  - 
  - 
  - 

```plaintext
(aa+aaa)*
b(a+b)*b + a(a+b)*a + a + b
a*(baa*)*bb(aa*b)*a*
```
A language $L$ is regular if there exists a DFA which accepts it
- DFA for all strings with exactly 2 occurrences of $bb$
FR-52: DFA Definition

- A DFA is a 5-tuple $M = (K, \Sigma, \delta, s, F)$
  - $K$ Set of states
  - $\Sigma$ Alphabet
  - $\delta : (K \times \Sigma) \mapsto K$ is a Transition function
  - $s \in K$ Initial state
  - $F \subseteq K$ Final states
A language $L$ is regular if there exists a DFA which accepts it.

DFA for all strings with exactly 2 occurrences of $bb$. 

The DFA is shown in the diagram with states labeled 0 to 7 and transitions indicated by arrows labeled with symbols 'a' and 'b'. The start state is 0, and the accepting state is 6.
A language $L$ is regular if there exists a DFA which accepts it.

- DFA for all strings over {0,1} that start and end with 111.
A language $L$ is regular if there exists a DFA which accepts it.

DFA for all strings over \{0,1\} that start and end with 111.
A language $L$ is regular if there exists a DFA which accepts it

- DFA for all strings over $\{0,1\}$ that start with 110, end with 011
A language $L$ is regular if there exists a DFA which accepts it.

- DFA for all strings over \{0,1\} that start with 110, end with 011.
Regular Languages

- Give a DFA for all strings over \( \{0,1\} \) that begin or end with 11
- Give a DFA for all strings over \( \{0,1\} \) that begin or end with 11 (but not both)
FR-59: Regular Languages

- Give a DFA for all strings over \(\{0, 1\}\) that contain 101010
- Give a DFA for all strings over \(\{0, 1\}\) that contain 101 or 010
- Give a DFA for all strings over \(\{0, 1\}\) that contain 010 and 101
• Way to describe the computation of a DFA

• **Configuration**: What state the DFA is currently in, and what string is left to process

  • \( \in K \times \Sigma^* \)
  
  • \((q_2, abba)\) Machine is in state \(q_2\), has \(abba\) left to process
  
  • \((q_8, bba)\) Machine is in state \(q_8\), has \(bba\) left to process
  
  • \((q_4, \epsilon)\) Machine is in state \(q_4\) at the end of the computation (accept iff \(q_4 \in F')\)
Way to describe the computation of a DFA

**Configuration**: What state the DFA is currently in, and what string is left to process

- $\in K \times \Sigma^*$

**Binary relation $\vdash_M$**: What machine $M$ yields in one step

- $\vdash_M \subseteq (K \times \Sigma^*) \times (K \times \Sigma^*)$
- $\vdash_M = \{(q_1, aw), (q_2, w) \mid q_1, q_2 \in K_M, w \in \Sigma^*_M, a \in \Sigma_M, ((q_1, a), q_2) \in \delta_M\}$
Given the following machine $M$:

- $((q_0, abba), (q_2, bba)) \in \vdash_M$
- can also be written $(q_0, abba) \vdash_M (q_2, bba)$
FR-63: DFA Configuration & $\vdash_M$

$(q_0, 11101) \vdash_M (q_1, 1101)$

$(q_1, \epsilon) \vdash_M (q_1, \epsilon)$
FR-64: DFA Configuration & $\vdash_M$

\[
(q_0, 10111) \vdash_M (q_1, 0111) \\
\vdash_M (q_0, 111) \\
\vdash_M (q_1, 11) \\
\vdash_M (q_2, 1) \\
\vdash_M (q_3, \epsilon)
\]
FR-65: DFA Configuration & $\vdash^*_M$

- $\vdash^*_M$ is the reflexive, transitive closure of $\vdash_M$
- Smallest superset of $\vdash_M$ that is both reflexive and transitive
- “yields in 0 or more steps”

- Machine $M$ accepts string $w$ if:
  $$(s_M, w) \vdash^*_M (f, \varepsilon) \text{ for some } f \in F_M$$
• Language accepted by a machine $M = L[M]$
  • $\{w : (s_M, w) \vdash^*_M (f, \epsilon) \text{ for some } f \in F_M\}$

• DFA Languages, $L_{DFA}$
  • Set of all languages that can be defined by a DFA
  • $L_{DFA} = \{L : \exists M, L[M] = L\}$

• To think about: How does $L_{DFA} = L_{REG}$
• Difference between a DFA and an NFA
  • DFA has exactly only transition for each state/symbol pair
    • Transition function: \( \delta : (K \times \Sigma) \mapsto K \)
  • NFA has 0, 1 or more transitions for each state/symbol pair
    • Transition relation: \( \Delta \subseteq ((K \times \Sigma) \times K) \)
A NFA is a 5-tuple $M = (K, \Sigma, \Delta, s, F)$
- $K$ Set of states
- $\Sigma$ Alphabet
- $\Delta : (K \times \Sigma) \times K$ is a Transition relation
- $s \in K$ Initial state
- $F \subseteq K$ Final states
Create an NFA for:

- All strings over \{a, b\} that start with a and end with b

(also create a DFA, and regular expression)
Fun with NFA

Create an NFA for:

- All strings over \{a, b\} that contain 010 or 101
A language $L$ is regular if there exists an NFA which accepts it.

NFA for all strings over $\{a, b\}$ that contain $abba$. 
A language $L$ is regular if there exists an NFA which accepts it.

- NFA for all strings over \{a, b\} that contain abba.
A language $L$ is regular if there exists an NFA which accepts it

- NFA for all strings over $\{a, b\}$ that do not contain $abba$
A language $L$ is regular if there exists an NFA which accepts it

- NFA for all strings over $\{a, b\}$ that do not contain $abba$
Give a regular expression for all strings over \{a,b\} that have an even number of a’s, and a number of b’s divisible by 3.
FR-76: Pumping Lemma

- Not all languages are Regular
- $L = \text{all strings over } \{a, b, c\} \text{ that contain more } a's \text{ than } b's \text{ and } c's \text{ combined}$
To show that a language $L$ is not regular, using the pumping lemma:

- Let $n$ be the constant of the pumping lemma
- Create a string $w \in L$, such that $|w| > n$
- For each way of breaking $w = xyz$ such that $|xy| \leq n$, $|y| > 0$:
  - Show that there is some $i$ such that $xy^iz \notin L$
- By the pumping lemma, $L$ is not regular
FR-78: Pumping Lemma

- Prove $L = \{a, b, c\}$ that contain more $a$’s than $b$’s and $c$’s combined is not regular.
- Let $n$ be the constant of the pumping lemma.
- Consider $w = b^n a^{n+1} \in L$.
- If we break $w = xyz$ such that $|xy| \leq n$, then $y$ must be all $b$’s. Let $|y| = k$.
- Consider $w' = xy^2x = b^{n+k}a^n$. $w' \not\in L$ for any $k > 0$, thus by the pumping lemma, $L$ is not regular.
A language is context-free if a CFG generates it
All strings over \( \{a, b, c\} \) with same # of \( a \)'s as \( b \)'s
A language is context-free if a CFG generates it
- All strings over \( \{a, b, c\} \) with same # of a’s as b’s

\[
\begin{align*}
S & \rightarrow aSb \\
S & \rightarrow bSa \\
S & \rightarrow SS \\
S & \rightarrow cS \\
S & \rightarrow Sc \\
S & \rightarrow \epsilon
\end{align*}
\]
A language is context-free if a CFG generates it

All strings over \( \{a, b, c\} \) with more \( a \)'s than \( b \)'s
A language is context-free if a CFG generates it.

All strings over \( \{a, b, c\} \) with more \( a \)'s than \( b \)'s

\[
\begin{align*}
S & \rightarrow cS | Sc \\
S & \rightarrow aSb | bSa \\
S & \rightarrow aA | Aa \\
S & \rightarrow SA \\
A & \rightarrow aAb \\
A & \rightarrow bAa \\
A & \rightarrow AA \\
A & \rightarrow cA | Ac \\
A & \rightarrow aA | Aa \\
A & \rightarrow \epsilon 
\end{align*}
\]
A language is context-free if a PDA accepts it
- All strings over \( \{a, b, c\} \) that contain more \( a \)'s than \( b \)'s and \( c \)'s combined
A language is context-free if a PDA accepts it.

- All strings over \{a, b, c\} that contain more a’s than b’s and c’s combined.

\[
\begin{align*}
(a, \varepsilon, \varepsilon) &\rightarrow (b, \varepsilon, X) \\
(a, \varepsilon, A) &\rightarrow (b, A, \varepsilon) \\
(a, X, \varepsilon) &\rightarrow (c, \varepsilon, X) \\
(a, X, \varepsilon) &\rightarrow (c, A, \varepsilon)
\end{align*}
\]
FR-85: Recursive Languages

- A language $L$ is recursive if an always-halting Turing Machine accepts it
  - In other words, a Turing Machine decides $L$
- Create a Turing Machine for all strings over $\{a, b, c\}$ with an equal number of a’s, b’s and c’s.
Computing functions with TMs

- Give a TM that computes negation, for a 2’s complement binary number
- (flip bits, add one, discard overflow)
• Computing functions with TMs
  • Give a TM that computes negation, for a 2’s complement binary number
Computing functions with TMs
- Give a TM that computes negation, for a 2’s complement binary number
- (flip bits, add one, discard overflow)
A language $L$ is recursively enumerable if there is some Turing Machine $M$ that halts and accepts everything in $L$, and runs forever on everything not in $L$.

Give a TM that semi-decides $L = a^n b^n$

- Note that this language is also context-free – context-free languages are a subset of the r.e. languages.
FR-90: r.e. Languages

• Enumeration Machines
  • Create a Turing Machine that enumerate the language:
    \[ L = \text{all strings of the form } wcw, w \in (a + b)^* \]
FR-91: **Counter Machines**

- Finite automata with a counter (never negative)
- Add one, subtract 1, check for zero
- Create a 1-counter machine for all strings over \{a,b\} that contain the same number of a’s as b’s
FR-92: Unrestricted Grammars

\[ G = (V, \Sigma, R, S) \]

- \( V = \) Set of symbols, both terminals & non-terminals
- \( \Sigma \subset V \) set of terminals (alphabet for the language being described)
- \( R \subset (V^*(V - \Sigma)V^* \times V^*) \) Set of rules
- \( S \in (V - \Sigma) \) Start symbol
Unrestricted Grammars

- $R \subset (V^*(V - \Sigma)V^* \times V^*)$ Set of rules
- In an Unrestricted Grammar, the left-hand side of a rule contains a string of terminals and non-terminals (at least one of which must be a non-terminal)
- Rules are applied just like CFGs:
  - Find a substring that matches the LHS of some rule
  - Replace with the RHS of the rule
To generate a string with an Unrestricted Grammar:

- Start with the initial symbol
- While the string contains at least one non-terminal:
  - Find a substring that matches the LHS of some rule
  - Replace that substring with the RHS of the rule
Example: Grammar for \( L = \{a^n b^n c^n : n > 0\} \)

- First, generate \((ABC)^*\)
- Next, non-deterministically rearrange string
- Finally, convert to terminals \((A \rightarrow a, B \rightarrow b,\) etc.), ensuring that string was reordered to form \(a^*b^*c^*\)
Example: Grammar for $L = \{a^n b^n c^n : n > 0\}$

- $S \rightarrow ABCS$
- $S \rightarrow TC$
- $CA \rightarrow AC$
- $BA \rightarrow AB$
- $CB \rightarrow BC$
- $CTC \rightarrow T_C c$
- $TC \rightarrow TB$
- $BTB \rightarrow TB b$
- $TB \rightarrow TA$
- $ATA \rightarrow TA a$
- $TA \rightarrow \epsilon$
Unrestricted Grammars

\begin{align*}
S & \Rightarrow \text{ABCS} & \Rightarrow \text{AAT}_A bbcc \\
& \Rightarrow \text{ABC} \text{ABCS} & \Rightarrow \text{AT}_A abbcc \\
& \Rightarrow \text{ABACBCS} & \Rightarrow T_A aabbcc \\
& \Rightarrow \text{AABCBCS} & \Rightarrow \text{aabbcc} \\
& \Rightarrow \text{AABBCCS} & \\
& \Rightarrow \text{AABBCCS} & \\
& \Rightarrow \text{AABBCCCT}_C & \\
& \Rightarrow \text{AABBCT}_C cc & \\
& \Rightarrow \text{AABBT}_C cc & \\
& \Rightarrow \text{AABBT}_B cc & \\
& \Rightarrow \text{AABT}_B bcc & \\
& \Rightarrow \text{AAT}_B bcc &
\end{align*}
**FR-98: Unrestricted Grammars**

\[
\begin{align*}
S & \Rightarrow ABCS \quad \Rightarrow AAABBBCCT_C \\
& \Rightarrow ABCABC \quad \Rightarrow AAABBBCCT_{Cc} \\
& \Rightarrow ABCABCABC \quad \Rightarrow AAABBBCCT_{Cc} \\
& \Rightarrow ABABAC \quad \Rightarrow AAABBBCCT_{Cccc} \\
& \Rightarrow AABABC \quad \Rightarrow AAABBBBT_Ccc \\
& \Rightarrow AABABCABC \quad \Rightarrow AAABBBBT_{Bccc} \\
& \Rightarrow AABABCABCABC \quad \Rightarrow AAABBBBT_{Bccc} \\
& \Rightarrow AABABCABCABC \quad \Rightarrow AAABBT_{Bccc} \\
& \Rightarrow AABABCABCABC \quad \Rightarrow AAABBT_{Bccc} \\
& \Rightarrow AABABCABCABC \quad \Rightarrow AAABBT_{Bccc} \\
& \Rightarrow AABABCABCABC \quad \Rightarrow AAABBT_{Bccc} \\
& \Rightarrow AABABCABCABC \quad \Rightarrow AAABBT_{Bccc} \\
& \Rightarrow AABABCABCABC \quad \Rightarrow AAABBT_{Bccc} \\
& \Rightarrow AABABCABCABC \quad \Rightarrow AAABBT_{Bccc} \\
& \Rightarrow AABABCABCABC \quad \Rightarrow AAABBT_{Bccc} \\
& \Rightarrow AABABCABCABC \quad \Rightarrow AAABBT_{Bccc} \\
& \Rightarrow AABABCABCABC \quad \Rightarrow AAABBT_{Bccc} \\
& \Rightarrow AABABCABCABC \quad \Rightarrow AAABBT_{Bccc} \\
& \Rightarrow AABABCABCABC \quad \Rightarrow AAABBT_{Bccc} \\
\end{align*}
\]