Finite Automata
Part Three
Recap from Last Time
A language $L$ is called a *regular language* if there exists a DFA $D$ such that $\mathcal{L}(D) = L$. 

**regular language**
NFAs

- An **NFA** is a
  - **N**ondeterministic
  - **F**inite
  - **A**utomaton
- Can have missing transitions or multiple transitions defined on the same input symbol.
- Accepts if *any possible series of choices* leads to an accepting state.
\(\varepsilon\text{-Transitions}\)

- NFAs have a special type of transition called the \(\varepsilon\text{-transition}\).

- An NFA may follow any number of \(\varepsilon\)-transitions at any time without consuming any input.
New Stuff!
DFAs and NFAs
NFAs and DFAs

• Any language that can be accepted by a DFA can be accepted by an NFA.

• Why?
  • Just use the same set of transitions as before.

• **Question:** Can any language accepted by an NFA also be accepted by a DFA?

• Surprisingly, the answer is **yes!**
Finite Automata

- NFAs and DFAs are *finite* automata; there can only be finitely many states in an NFA or DFA.
- An NFA can be in any combination of its states, but there are only finitely many possible combinations.
- **Idea:** Build a DFA where each state of the DFA corresponds to a set of states in the NFA.
Simulating an NFA with a DFA
The Subset Construction

- This construction for transforming an NFA into a DFA is called the *subset construction* (or sometimes the *powerset construction*).
  - States of the new DFA correspond to *sets of states* of the NFA.
  - The initial state is the start state, plus all states reachable from the start state via $\varepsilon$-transitions.
  - Transition on state $S$ on character $a$ is found by following all possible transitions on $a$ for each state in $S$, then taking the set of states reachable from there by $\varepsilon$-transitions.
  - Accepting states are any set of states where some state in the set is an accepting state.
- *Read Sipser for a formal account.*
The Subset Construction

- In converting an NFA to a DFA, the DFA's states correspond to sets of NFA states.

- **Useful fact:** \(|\emptyset(S)| = 2^{|S|}\) for any finite set \(S\).

- In the worst-case, the construction can result in a DFA that is exponentially larger than the original NFA.

- Interesting challenge: Find a language for which this worst-case behavior occurs (there are infinitely many of them!)
A language $L$ is called a **regular language** if there exists a DFA $D$ such that $\mathcal{L}(D) = L$. 

An Important Result

**Theorem:** A language \( L \) is regular iff there is some NFA \( N \) such that \( \mathcal{L}(N) = L \).

**Proof Sketch:** If \( L \) is regular, there exists some DFA for it, which we can easily convert into an NFA. If \( L \) is accepted by some NFA, we can use the subset construction to convert it into a DFA that accepts the same language, so \( L \) is regular. ■
Why This Matters

• We now have two perspectives on regular languages:
  • Regular languages are languages accepted by DFAs.
  • Regular languages are languages accepted by NFAs.
• We can now reason about the regular languages in two different ways.
Time-Out for Announcements!
Midterm Denouement

• The TAs will be grading the midterm exam over the weekend. We'll have it returned on Monday.

• We'll release solution sets, common mistakes, the rationale behind each question, and exam statistics along with the exam.
Problem Set Four

• As a reminder, Problem Set Four is due this upcoming Monday at 12:50PM.

• Please feel free to ask questions!
  • Email the staff list!
  • Ask on Piazza!
  • Stop by office hours!
WiCS Board

• “Apply for WiCS Board by midnight tonight! You **do not** have to be a declared CS major and all years are welcome! We have a wide variety of events planned for next year and you can be a part of helping WiCS be bigger than it has ever been! Please reach out if you have questions!”

• **Click here** to apply.
Back to CS103!
Properties of Regular Languages
The Union of Two Languages

- If $L_1$ and $L_2$ are languages over the alphabet $\Sigma$, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.
- If $L_1$ and $L_2$ are regular languages, is $L_1 \cup L_2$?
The Intersection of Two Languages

- If $L_1$ and $L_2$ are languages over $\Sigma$, then $L_1 \cap L_2$ is the language of strings in both $L_1$ and $L_2$.
- Question: If $L_1$ and $L_2$ are regular, is $L_1 \cap L_2$ regular as well?

Hey, it's De Morgan's laws!
Concatenation
String Concatenation

- If $w \in \Sigma^*$ and $x \in \Sigma^*$, the *concatenation* of $w$ and $x$, denoted $wx$, is the string formed by tacking all the characters of $x$ onto the end of $w$.

- Example: if $w = \text{quo}$ and $x = \text{kka}$, the concatenation $wx = \text{quokka}$.

- Analogous to the $+$ operator for strings in many programming languages.
The **concatenation** of two languages $L_1$ and $L_2$ over the alphabet $\Sigma$ is the language

$$L_1L_2 = \{ wx \in \Sigma^* \mid w \in L_1 \land x \in L_2 \}$$
Concatenation Example

- Let $\Sigma = \{ a, b, ..., z, A, B, ..., Z \}$ and consider these languages over $\Sigma$:
  - $Noun = \{ \text{Puppy, Rainbow, Whale, ... } \}$
  - $Verb = \{ \text{Hugs, Juggles, Loves, ... } \}$
  - $The = \{ \text{The} \}$
  - The language $TheNounVerbTheNoun$ is
    \[
    \{ \text{ThePuppyHugsTheWhale, TheWhaleLovesTheRainbow, TheRainbowJugglesTheRainbow, ... } \}
    \]
Concatenation

• The *concatenation* of two languages $L_1$ and $L_2$ over the alphabet $\Sigma$ is the language

$$L_1L_2 = \{ wx \in \Sigma^* \mid w \in L_1 \land x \in L_2 \}$$

• Two views of $L_1L_2$:
  • The set of all strings that can be made by concatenating a string in $L_1$ with a string in $L_2$.
  • The set of strings that can be split into two pieces: a piece from $L_1$ and a piece from $L_2$.
• Conceptually similar to the Cartesian product of two sets, only with strings.
Concatenating Regular Languages

- If $L_1$ and $L_2$ are regular languages, is $L_1 L_2$?
- Intuition – can we split a string $w$ into two strings $xy$ such that $x \in L_1$ and $y \in L_2$?

Machine for $L_1$

Machine for $L_2$

book

keeper
Concatenating Regular Languages

• If $L_1$ and $L_2$ are regular languages, is $L_1L_2$?
• Intuition – can we split a string $w$ into two strings $xy$ such that $x \in L_1$ and $y \in L_2$?
• **Idea**: Run the automaton for $L_1$ on $w$, and whenever $L_1$ reaches an accepting state, optionally hand the rest off $w$ to $L_2$.
  • If $L_2$ accepts the remainder, then $L_1$ accepted the first part and the string is in $L_1L_2$.
  • If $L_2$ rejects the remainder, then the split was incorrect.
Concatenating Regular Languages

Machine for $L_1$

Machine for $L_2$

Machine for $L_1L_2$
Lots and Lots of Concatenation

- Consider the language $L = \{ \text{aa, b} \}$
- $LL$ is the set of strings formed by concatenating pairs of strings in $L$.
  \[
  \{ \text{aaaa, aab, baa, bb} \}
  \]
- $LLL$ is the set of strings formed by concatenating triples of strings in $L$.
  \[
  \{ \text{aaaaaa, aaaaab, aabaa, aabb, baaaa, baab, bbaa, bbb} \}
  \]
- $LLLL$ is the set of strings formed by concatenating quadruples of strings in $L$.
  \[
  \{ \text{aaaaaaaaaa, aaaaaaab, aaaaabaa, aaaaabb, aabaaaaa, aabaab, aabbaa, aabbb, baaaaaa, baaaaab, baabaa, baabb, bbaaaa, bbaaab, bbbbaa, bbbb} \}
  \]
Language Exponentiation

- We can define what it means to “exponentiate” a language as follows:

- $L^0 = \{\varepsilon\}$
  - The set containing just the empty string.
  - Idea: Any string formed by concatenating zero strings together is the empty string.

- $L^{n+1} = LL^n$
  - Idea: Concatenating $(n+1)$ strings together works by concatenating $n$ strings, then concatenating one more.

- **Question:** Why define $L^0 = \{\varepsilon\}?$
The Kleene Closure

• An important operation on languages is the **Kleene Closure**, which is defined as

\[ L^* = \bigcup_{i=0}^{\infty} L^i \]

• Mathematically:

\[ w \in L^* \quad \text{iff} \quad \exists n \in \mathbb{N}. \, w \in L^n \]

• Intuitively, all possible ways of concatenating any number of copies of strings in \( L \) together.
The Kleene Closure

If $L = \{a, bb\}$, then $L^* = \{\epsilon, a, bb, aa, aabb, abba, bba, bbb, bbba, bbbba, bbbbbbb, \ldots\}$
Reasoning about Infinity

- If $L$ is regular, is $L^*$ necessarily regular?

**A Bad Line of Reasoning:**

- $L^0 = \{ \varepsilon \}$ is regular.
- $L^1 = L$ is regular.
- $L^2 = LL$ is regular
- $L^3 = L(LL)$ is regular
- ...

- Regular languages are closed under union.
- So the union of all these languages is regular.
Reasoning about Infinity
Reasoning about Infinity

\[ x \neq 2x \]
Reasoning about Infinity

0.999 < 1
Reasoning about Infinity

\[0.99999\bar{9} \not< 1\]
Reasoning about Infinity

1 is finite
Reasoning about Infinity

\[ \infty \text{ is finite} \]

\[ ^{\wedge} \text{ not} \]
Reasoning About the Infinite

- If a series of finite objects all have some property, the “limit” of that process does not necessarily have that property.
- In general, it is not safe to conclude that some property that always holds in the finite case must hold in the infinite case.
  - (This is why calculus is interesting).
**Idea:** Can we directly convert an NFA for language $L$ to an NFA for language $L^*$?
The Kleene Star

Question: Why add the new state out front? Why not just make the old start state accepting?
Summary

- NFAs are a powerful type of automaton that allows for nondeterministic choices.
- NFAs can also have $\epsilon$-transitions that move from state to state without consuming any input.
- The subset construction shows that NFAs are not more powerful than DFAs, because any NFA can be converted into a DFA that accepts the same language.
- The union, intersection, complement, concatenation, and Kleene closure of regular languages are all regular languages.