Fun:

Assignment 1 is due today!
Corrections due Tuesday!
Assignment 2 soon!
Fun!

Previously:

Equivalent computational models: DFAs and NFAs
Some closure properties of regular languages

Today:

More closure properties
A new formalism for regular languages
Closure properties of regular languages, continued
Closure properties

Certain operations on regular languages are guaranteed to produce regular languages.
The complement of a language

Given a language \( L \subseteq \Sigma^* \), the *complement* of that language (denoted \( \overline{L} \)) is the language of all strings in \( \Sigma^* \) not in \( L \).

Formally,

\[
\overline{L} = \{w \mid w \in \Sigma^* \land w \notin L\}
\]
The complement of a language

Given a language $L \subseteq \Sigma^*$, the *complement* of that language (denoted $\bar{L}$) is the language of all strings in $\Sigma^*$ that aren’t in $L$.

Formally:

$$\bar{L} = \Sigma^* - L$$
Complementing regular languages

A regular language is a language accepted by some DFA.

If $L$ is a regular language, we can show $\overline{L}$ is a regular language by constructing a DFA for it.
Complementing regular languages

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring} \} \]

\[ \bar{L} = \{ w \in \{0, 1\}^* \mid w \text{ does not contain } 11 \text{ as a substring} \} \]
Complementing regular languages

A regular language is a language accepted by some DFA.

If $L$ is a regular language, we can show $\overline{L}$ is a regular language by constructing a DFA for it.

*Therefore, the regular languages are closed under complementation.*
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 $L_1 \cup L_2$?

Yes! We proved it before. But now that we have NFAs, the proof is easier.
The new machine guesses non-deterministically which of the two machines accepts the input.
This construction proves the class of regular languages is closed under the union operation.
Concatenation of strings

*Recall*: If \( w \in \Sigma^* \) and \( x \in \Sigma^* \), the concatenation of \( w \) and \( x \), denoted \( w \circ x \) or just \( wx \), is the string formed by tacking all characters in \( x \) onto the end of \( w \).

E.g., if \( w = \text{quo} \) and \( x = \text{kka} \), the concatenation \( wx = \text{quokka} \)

A quokka, happy just to be mentioned
The concatenation of two languages $L_1$ and $L_2$ over the alphabet $\Sigma$ is the language

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

E.g., consider the languages

- $Noun = \{\text{Puppy}, \text{Rainbow}, \text{Whale}, \ldots\}$
- $Verb = \{\text{Hugs}, \text{Juggles}, \text{Loves}, \ldots\}$
- $Det = \{\text{A, The}\}$

The language $DetNounVerbDetNoun$ is

Concatenation of languages

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 $L_1$ followed by a piece from $L_2$.

Conceptually it’s similar to the Cartesian product of two sets, only with strings.
Concatenation of languages

If $L_1$ and $L_2$ are regular languages, is $L_1L_2$?

We started trying to prove this with DFAs but hit a stumbling block: After every character we read, we might be seeing the start of the string from the second language.

Can we do it with NFAs?
Idea: Given a string $w$, run a finite automaton for $L_1$ on $w$. Whenever it reaches an accepting state, optionally hand the rest of $w$ to the finite automaton for $L_2$.

If the automaton for $L_2$ accepts the rest, $w \in L_1L_2$.

If the automaton for $L_2$ rejects the remainder, either $w \notin L_1L_2$ or the split was incorrect.
The new machine guesses non-deterministically where to split the input in order to have a first part accepted by $N_1$ and a second part accepted by $N_2$. 
This construction proves the class of regular languages is closed under concatenation.
Lots of concatenation

Consider the language $L = \{aa, b\}$

$L L$ is the set of strings formed by concatenating pairs of strings in $L$:

$$\{aaaa, aab, baa, bb\}$$

$LLL$ is the set of strings formed by concatenating triples of strings in $L$:

$$\{aaaaaa, aaaaab, aabaa, aabb, baaaa, baab, bbba, bbb\}$$

$LLLL$ is the set of strings formed by concatenating quadruples of strings in $L$...
Language exponentiation

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

\[ L^0 = \{ \varepsilon \} \]

**Base case:** Any string formed by concatenating zero strings together is just the empty string.

\[ L^{n+1} = LL^n \]

**Recursive case:** Concatenating \( n+1 \) strings together works by concatenating \( n \) strings, then concatenating one more.
Kleene (star) closure

An important operation on languages is the Kleene closure, which is defined as

$$L^* = \{ w \in \Sigma^* | \exists n \in \mathbb{N}_0 . w \in L^n \}$$

A word is in $L^*$ iff it’s in one of the languages $L^0, L^1, L^2, \ldots$

That is, $L^*$ consists of all the possible ways of concatenating zero or more strings in $L$. 
If $L = \{a, bb\}$, then $L^* = \{$

$\varepsilon$,

$a, bb,$

$aa, abb, bba, bbbb,$

$aaa, aabb, abba, abbbbb, bbaa, bbabb, bbbbaa,$

$bbbbbb,$

$\ldots$,

$\}$
If $L$ is a regular language, is $L^*$ necessarily regular?
The new machine has the option of jumping back to the start state to read another piece that $N_1$ accepts.

Why not just make the start state of $N_1$ a final state?
This construction proves the class of regular languages is closed under Kleene star.
Regular languages are closed under common set operations

Regular operations:

- **Union**: $L_1 \cup L_2$
- **Concatenation**: $L_1L_2$
- **Star-closure**: $L_1^*$
Regular languages are closed under common set operations

Regular operations:

- **Union**: \( L_1 \cup L_2 \)
- **Concatenation**: \( L_1L_2 \)
- **Star-closure**: \( L_1^* \)

Additional set operations:

- **Complementation**: \( \overline{L}_1 \)
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 $L_2$.

If $L_1$ and $L_2$ are both regular, is $L_1 \cap L_2$ regular?
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 $L_2$.

If $L_1$ and $L_2$ are both regular, is $L_1 \cap L_2$ regular?
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 $L_2$.

If $L_1$ and $L_2$ are both regular, is $L_1 \cap L_2$ regular?
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 $L_2$.

If $L_1$ and $L_2$ are both regular, is $L_1 \cap L_2$ regular?

De Morgan’s laws!
Regular languages are closed under common set operations

The regular operations:

- **Union**: $L_1 \cup L_2$
- **Concatenation**: $L_1L_2$
- **Star-closure**: $L_1^*$

Additional set operations:

- **Complementation**: $\overline{L_1}$
- **Intersection**: $L_1 \cap L_2$

**Note**: There are more operations regular languages are closed under, too!
Regular expressions
We’ve seen we can show a language is regular by
constructing a DFA for it
constructing an NFA for it (with or without ε transitions)

We can also show a language is regular by
constructing it out of simpler regular languages
using closure properties.
Regular expressions are a concise notation for describing how to assemble a larger language out of smaller pieces.
A bottom-up approach to the regular languages:

Start with a small set of simple languages we know to be regular
Use closure properties to combine these to form more elaborate languages
Operators and operands

Regular expressions are built by combining three kinds of simple operands:

For any symbol $a \in \Sigma$, the regular expression $a$ represents the language $\{a\}$

The symbol $\epsilon$ is a regular expression representing the language $\{\epsilon\}$

The symbol $\emptyset$ is a regular expression for the empty language $\emptyset$

These are combined using symbols for the regular operations union, concatenation, and Kleene star.
Operation: Union

If $R_1$ and $R_2$ are regular expressions, $(R_1 \cup R_2)$ is a regular expression for the union of the languages of $R_1$ and $R_2$: $L((R_1 \cup R_2)) = L(R_1) \cup L(R_2)$. 
Operation: Concatenation

If $R_1$ and $R_2$ are regular expressions, $(R_1 \circ R_2)$ is a regular expression for the concatenation of the languages of $R_1$ and $R_2$. 
Operation: Kleene star

If $R$ is a regular expression, $(R^*)$ is a regular expression for the Kleene closure of the language of $R$. 
**Formal definition of regular expressions**

DE**FINITION**. $R$ is a *regular expression* if $R$ is

| 1 | $a$ for some $a \in \Sigma$                                        |
| 2 | $\varepsilon$                                                     |
| 3 | $\emptyset$                                                      |
| 4 | $(R_1 \cup R_2)$, where $R_1$ and $R_2$ are regular expressions |
| 5 | $(R_1 \circ R_2)$, where $R_1$ and $R_2$ are regular expressions |
| 6 | $(R_1^*)$, where $R_1$ is a regular expression                    |

Every regular expression arises by a finite number of applications of these six rules.
Order of operations

We can omit parentheses (and the concatenation operator) to make regular expressions more compact, but this makes them ambiguous without defining precedence.

0 Parentheses – (R)

1 Kleene star – R*

2 Concatenation – R₁R₂ or R₁ ⋅ R₂

3 Union – R₁ ∪ R₂
Empty strings, empty sets

Do not confuse the regular expressions:

- $\varepsilon$ – the language containing only the empty string
- $\varnothing$ – the language containing no strings

Identities:

- $R \cup \varnothing = R$
- $R \circ \varepsilon = R$
R is a regular expression if R is
1 a for some a ∈ Σ
2 ε
3 ∅
4 (R₁ ∪ R₂), where R₁ and R₂ are regular expressions
5 (R₁ ⊙ R₂), where R₁ and R₂ are regular expressions
6 (R₁^*), where R₁ is a regular expression

Example: To prove ((a(b*))∪a) is a regular expression over Σ = {a, b}, show it can be constructed according to the rules:

1 b is regular by Rule 1
2 (b*) is regular by Rule 6
3 a is regular by Rule 1
4 (a(b*)) is regular by Rule 5
5 ((a(b*))∪a) is regular by Rule 4 applied to expressions (4) and (3)
Examples

\[ L(\text{hi}) = \{\text{hi}\} \]

\[ L(\text{hi} \cup \text{heyy}^*) = \{\text{hi}, \text{hey}, \text{heyy}, \text{heyyy}, \ldots\} \]

\[ L((\text{0}(\text{0} \cup \text{1}))^*) = \text{the set of strings of 0s and 1s, of even length, such that every odd position has a 0} \]
A few more examples…

\(ab^*a\)

\(a^*b^*\)

\((ab)^*\)

Is this the same as \(a^*b^*\)?

\(a^*b^*a^*\)

Is baa in this?

\(L = \{x^{\text{odd}}\} = x(xx)^* \text{ or } (xx)^*x \text{ but not } x^*xx^*\)

All strings of \(a\)s and \(b\)s of exactly length 3

\(L = \{aaa, aab, aba, aabb, baa, bab, bba, bbb\}\)

or \((a\cup b)(a\cup b)(a\cup b)\)

or \((a\cup b)^3\)
Acknowledgments

This lecture incorporates material from:

- Nancy Ide
- Keith Schwarz
- Michael Sipser
- Jeffrey Ullman