Regular expressions
Sec 2.4

Operations on formal languages
Let \( L_1 = \{ 10 \} \) and \( L_2 = \{ 011, 11 \} \).

- **Union**: \( L_1 \cup L_2 = \{ 10, 011, 11 \} \)
- **Concatenation**: \( L_1 L_2 = \{ 10011, 1011 \} \)
- **Kleene Star**: \( L_1^* = \{ e, 10, 1010, 101010, \ldots \} \)

Other operations: intersection, complement, difference

Closure properties of regular languages

- **Definition**: A regular language is any language that is accepted by a finite automaton
- **Theorem 2.4.1** (pp. 31-3): The class of regular languages is closed under the following operations (that is, performing these operations on regular languages creates other regular languages)
  - Union
  - Complement
  - Intersection
  - Difference
  - Concatenation
  - Kleene star

Regular expressions
A useful shorthand for describing regular languages.

Compare to arithmetic expressions, such as \((x + 3)/2\).
An arithmetic expression is constructed using arithmetic operators, such as addition and division. A regular expression is constructed using operations on languages, in particular, concatenation, union, and Kleene star.

The value of an arithmetic expression is a number.
The value of a regular expression is a language.

Examples
\[ a^* \cup b = \{ b, c, a, aa, aaa, aaaa, \ldots \} \]
\[ a^*b^* = \{ w \in \Sigma^* : w \text{ has exactly one } b \} \]
\[ (a \cup b)^*aa(a \cup b)^* = \{ w \in \Sigma^* : w \text{ contains } aa \} \]
\[ (a \cup b)^*aa(a \cup b)^* \cup (a \cup b)^*bb(a \cup b)^* = \{ w \in \Sigma^* : w \text{ contains } aa \text{ or } bb \} \]
\[ (a \cup c)b^* = \{ ab^n : n \geq 0 \} \cup \{ b^n : n \geq 0 \} \]

As with arithmetic expressions, there is an order of precedence for operators -- unless you change it using parentheses. The order is: star closure first, then concatenation, then union.
Practice
Let \( \Sigma = \{a, b, c\} \)

(a) all strings containing exactly one a

(b) all strings containing no more than three a's

Hints for writing regular expressions
Assume \( \Sigma = \{a, b, c\} \).

Zero or more a's: \( a^* \)

One or more a's: \( aa^* \)

Any string at all: \( (a \cup b \cup c)^* \)

Any nonempty string: \( (a \cup b \cup c)(a \cup b \cup c)^* \)

Any string that does not contain a: \( (b \cup c)^* \)

Any string containing exactly one a: \( (b \cup c)^*a(b \cup c)^* \)

Practice
What languages correspond to the following regular expressions?

\[ a^*b \]

\[ (aaa \cup bba) \]

\[ (ab)^* \]

More practice
Give regular expressions for the following languages on \( \Sigma = \{a, b, c\} \).

-- all strings ending in b

-- all strings containing no more than two a's

-- all strings of even length

More practice
Give regular expressions for the following languages on \( \Sigma = \{0, 1\} \).

-- all strings of one or more 0's followed by a 1

-- all strings of two or more symbols followed by three or more 0's

-- all strings that do not end with 01

More examples
All strings containing no more than two a's:

\[ (b \cup c)^*(e \cup a)(b \cup c)^*(e \cup a)(b \cup c)^* \]

All strings containing no runs of a's of length greater than two:

\[ (b \cup c)^*(e \cup a \cup aa)(b \cup c)^*(b \cup c)(b \cup c)^*(e \cup aaa)(b \cup c)^* \]

All strings in which all runs of a's have lengths that are multiples of three:

\[ (aaa \cup b \cup c)^* \]
Do these strings match the regular expression?

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>(01* ∪ 1)</td>
<td>0101</td>
</tr>
<tr>
<td>(a ∪ ε)b</td>
<td>b</td>
</tr>
<tr>
<td>(ab)<em>a</em></td>
<td>ε</td>
</tr>
<tr>
<td>(a ∪ b)(ab)</td>
<td>bb</td>
</tr>
</tbody>
</table>

Kleene’s theorem

1) For any regular expression \( r \) that represents language \( L(r) \), there is a finite automaton that accepts that same language.

2) For any finite automaton \( M \) that accepts language \( L(M) \), there is a regular expression that represents the same language.

Therefore, the class of languages that can be represented by regular expressions is equivalent to the class of languages accepted by finite automata -- the regular languages.

Proof of 1st half of Kleene’s theorem

Proof by construction: for any regular expression, we show how to construct an equivalent NFA.

Because regular expressions are defined recursively, the proof is by induction.

Base step: Give a NFA that accepts each of the simple or “base” languages, \( ∅ \), \{ε\}, and \{a\} for each \( a \in \Sigma \).

Inductive step: For each of the operations -- union, concatenation and Kleene star -- show how to construct an accepting NFA.

Closure under union:
Closure under concatenation:

Closure under Kleene Star:

Exercise

Use the construction of the first half of Kleene’s theorem to construct a NFA that accepts the language \(L(ab^*aa \cup bba^*ab)\).

Exercise

Construct a NFA that accepts the language corresponding to the regular expression:

\[(b(a\cup b)^*a) \cup a\]

Kleene’s theorem part 2

Any language accepted by a finite automaton can be represented by a regular expression.

The proof is by construction. For any DFA, we show how create an equivalent regular expression. In other words, we describe an algorithm for converting any DFA to a regular expression.

Expression diagram

- Labeled directed graph (similar to a finite state diagram) in which transitions are labeled by regular expressions
- single start state with no incoming transitions
- single accepting state with no outgoing transitions
- Example:

Algorithm for converting a DFA into an equivalent regular expression

Initial step: Change every transition labeled a,b to (a\cup b).
Add a single start state with outgoing \(e\)-transition to the current start state, and a single final state with incoming \(e\)-transitions from every previous final state.

Main step: Until the expression diagram has only two states (an initial state and a final state), repeat the following:
- pick some non-start, non-final state
- remove it from the diagram and re-label the transitions with regular expressions so that the same language is accepted
The key step is removing states and re-labeling transitions with regular expressions. Here are some examples of how to do this.

The simplest possible regular languages are the empty set and languages that consist of a single string that is either the empty string or has length one. For example; if \( \Sigma = \{a, b\} \), the simplest languages are \( \emptyset \), \{e\}, \{a\}, and \{b\}.

A regular language is a language that can be built from these simple languages, by using just the three operations of union, concatenation, and Kleene star.

Exercise

Find a regular expression that corresponds to the language accepted by the following DFA.

Exercise

Find a regular expression that corresponds to the language accepted by the following DFA.

Applications of regular expressions

- Validation
  - checking that an input string is in valid format
  - example 1: checking format of email address on WWW entry form
  - example 2: UNIX regex command
- Search and selection
  - looking for strings that match a certain pattern
  - example: UNIX grep command
- Tokenization
  - converting a sequence of characters (a string) into a sequence of tokens (e.g., keywords, identifiers)
  - used in lexical analysis phase of compiler