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 strategy: 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, $\emptyset$, $\{\lambda\}$, 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:

\[ M_1 \lambda M_2 \]

Closure under Kleene Star:

\[ \lambda M_1 \lambda \]

Exercise

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

Kleene’s theorem part 2

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

The proof strategy: 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.

Generalized transition graph

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

\[ (a+b) \]

Algorithm for converting a DFA into an equivalent regular expression

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

Main step: Until expression diagram has only two states (initial state and final state), repeat the following:
- pick some non-start, non-final state
- remove it from diagram and re-label transitions with regular expressions so that the same language is accepted

Exercise

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

\[ ((b(a+b)^*a) + a) \]
The key step is removing states and re-labeling transitions with regular expressions. Here are some examples of how to do this.

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.

Alternative definition of regular languages

The simplest possible regular languages are the empty set and languages consisting 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 the three operations of union, concatenation, and Kleene star.

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 sequence of characters (a string) into sequence of tokens (e.g., keywords, identifiers)
  - used in lexical analysis phase of compiler