In this lecture, we introduce Regular Expressions as an equivalent way, yet more elegant, to describe regular languages.

Basic Regular Expressions

A Regular Expression (RE in short) is a string of symbols that describes a regular language.

1. Let \( \Sigma \) be an alphabet. For each symbol \( \sigma \in \Sigma \), the symbol \( \sigma \) is an RE representing the set \( \{ \sigma \} \).
2. The symbol \( \epsilon \) is an RE representing the set containing the empty string \( \{ \epsilon \} \).
3. The symbol \( \phi \) is an RE representing the empty set \( \emptyset \).

Motivation

If one wants to describe a regular language, \( L(D) \), she can use the DFA \( D \) or an NFA \( N \), such that \( L(D) = L_N \).

This is not always very convenient. Consider for example the regular expression \( \{ \} \) describing the language of binary strings containing a single 1.
Inductive Construction

Let $R_1$ and $R_2$ be two regular expressions representing languages $L_1$ and $L_2$, resp.

- The string $(R_1 \cup R_2)$ is a regular expression representing the set $L_1 \cup L_2$.
- The string $(R_1R_2)$ is a regular expression representing the set $L_1 \circ L_2$.
- The string $(R_1)^*$ is a regular expression representing the set $L_1^*$.

Inductive Construction - Remark

Note that in the inductive part of the definition larger RE-s are defined by smaller ones. This ensures that the definition is not circular.

This inductive definition also dictates the way we will prove theorems:

Stage 1: Prove $T \text{e}$ correct for all base cases.

Stage 2: Assume $T \text{e}$ is correct for $R_1$ and $R_2$ and prove its correctness for $(R_1 \cup R_2)(R_1R_2)(R_1)^*$. 

Some Useful Notation

Let $R$ be a regular expression:

- The string $R^+$ represents $RR^*$, and it also holds that $R^+ \cup \{\varepsilon\} = R^*$.
- The string $R^*$ represents $RR\ldots R$ \text{k times}.
- The string $\sum \sigma_1,\sigma_2,\ldots,\sigma_k$.
- The Language represented by $R$ is denoted by $L(R)$.

Precedence Rules

- The star ($*$) operation has the highest precedence.
- The concatenation ($\circ$) operation is second on the preference order.
- The union ($\cup$) operation is the least preferred.
- Parentheses can be omitted using these rules.
Examples

• $0^*10^*$ – \( \{ w \mid w \text{ contains a single } 1 \} \).
• $\Sigma^*1\Sigma^*$ – \( \{ w \mid w \text{ has at least a single } 1 \} \).
• $\Sigma^*(\text{str})\Sigma^*$ – \( \{ w \mid w \text{ contains str as a substring} \} \).
• $1^*(01^+)^*$ – \( \{ w \mid \text{every } 0 \text{ in } w \text{ is followed by at least a single } 1 \} \).
• $(\Sigma\Sigma)^*$ – \( \{ w \mid w \text{ is of even length} \} \).

Equivalence With Finite Automata

Regular expressions and finite automata are equivalent in their descriptive power. This fact is expressed in the following Theorem:

**Theorem**

A set is regular if and only if it can be described by a regular expression.

The proof is by two Lemmata (Lemmata):

Lemma ->

If a language $L$ can be described by regular expression then $L$ is regular.
Proofs Using Inductive Definition

This inductive definition of Regular expressions dictates the way we will prove theorems. The proof for the Theorem follows the following stages:

**Stage 1:** Prove correctness for all base cases.

**Stage 2:** Assume correctness for $R_1$ and $R_2$, and show its correctness for $(R_1 \cup R_2)$, $(R_1 \cdot R_2)$ and $(R_1)^*$.

Induction Basis

1. For any $\sigma \in \Sigma$, the expression $\sigma$ describes the set $\{\sigma\}$, recognized by:

   - $q_0 \rightarrow \sigma \rightarrow q_4$

2. The set represented by the expression $\varepsilon$ is recognized by:

   - $q_4$

3. The set represented by the expression $\phi$ is recognized by:

   - $q_4$

The Induction Step

Now, we assume that $R_1$ and $R_2$ represent two regular sets and claim that $R_1 \cup R_2$, $R_1 \cdot R_2$ and $(R_1)^*$ represent the corresponding regular sets.

The proof for this claim is straightforward using the constructions given in the proof for the closure of the three regular operations.

Examples

Show that the following regular expressions represent regular languages:

1. $(ab \cup a)^*$.
2. $(a \cup b)^* aba$.

To be demonstrated on the Blackboard.
Lemma <-

If a language $L$ is regular then $L$ can be described by regular expression.

Proof Stages

The proof follows the following stages:
2. Show how to convert any DFA to an equivalent GNFA.
3. Show an algorithm to convert any GNFA to an equivalent GNFA with 2 states.
4. Convert a 2-state GNFA to an equivalent RE.

Properties of a Generalized NFA

1. A GNFA is a finite automaton in which each transition is labeled with a regular expression over the alphabet $\Sigma$.
2. A single initial state with all possible outgoing transitions and no incoming trans.
3. A single final state without outgoing trans.
4. A single transition between every two states, including self loops.

Example of a Generalized NFA
A computation of a GNFA is similar to a computation of an NFA, except:
In each step, a GNFA consumes a block of symbols that matches the RE on the transition used by the NFA.

Example of a GNFA Computation

Consider $\text{abbbaaabbbbb}$ or $\text{bb}$ or $\text{abba}$

21

Example of a GNFA Computation

22

Converting a DFA to a GNFA (Cont)

4. Replace any transition with multiple labels by a single transition labeled with the union of all labels.
5. Add any missing transition, including self transitions; label the added transition by $\phi$.

Converting a DFA to a GNFA (Cont)

Conversion is done by a very simple process:
1. Add a new start state with an $\epsilon$-transition from the new start state to the old start state.
2. Add a new accepting state with $\epsilon$-transition from every old accepting state to the new accepting state.

23

24
Ripping a state from a GNFA

The final element needed for the proof is a procedure in which for any GFN $G$, any state of $G$, not including $q_{start}$ and $q_{accept}$, can be ripped off $G$, while preserving $L(G)$. This is demonstrated in the next slide by considering a general state, denoted by $q_{rip}$, and an arbitrary pair of states, $q_i$ and $q_j$, as demonstrated in the next slide:

Removing a state from a GNFA

Before Ripping

After Ripping

Note: This should be done for every pair of outgoing and incoming outgoing $q_{rip}$.

Ellaboration

Consider the RE $(R_1)(R_2)^*R_3$, representing all strings that enable transition from $q_i$ via $q_{rip}$ to $q_j$.

What we want to do is to augment the Regular expression of transition $(q_i, q_j)$, namely $R_4$, so These strings can pass through $(q_i, q_j)$. This is done by setting it to $R_4 \cup (R_1)(R_2)^*(R_3)_4$.

Ellaboration

Note that this change does not affect all pairs in which either $(q_i, q_{rip})$ or $(q_{rip}, q_j)$ participate. Thus, before $q_{rip}$ is removed all these pairs should be processed in the same way, as demonstrated on the next slide:
**Elaboration**

Assume the following situation:
In order to rip $q_{\text{rip}}$, all pairs of incoming and outgoing transitions should be considered in the way showed on the previous slide namely consider $(t_1, t_4), (t_1, t_5), (t_2, t_4), (t_2, t_5), (t_3, t_4), (t_3, t_5)$ one after the other. After that $q_{\text{rip}}$ can be ripped while preserving $L(G)$.

**A (half?) Formal Proof of Lemma**

The first step is to formally define a GNFA. Each transition should be labeled with an RE. Define the transition function as follows:

$$\delta : (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \rightarrow RE_{\Sigma}$$

where $RE_{\Sigma}$ denotes all regular expressions over $\Sigma$.

**Note:** The definition of $\delta$ is different than for NFA.

**Changes in $\delta$ Definition**

**Note:** The definition of $\delta$ as:

$$\delta : (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \rightarrow RE_{\Sigma}$$

is different than the original definitions (For DFA and NFA).

In this definition we rely on the fact that every 2 states (except $q_{\text{start}}$ and $q_{\text{accept}}$) are connected in both directions.

**GNFA – A Formal Definition**

A *Generalized Finite Automaton* is a 5-tuple $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ where:

1. $Q$ is a finite set called the **states**.
2. $\Sigma$ is a finite set called the **alphabet**.
3. $\delta : (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \rightarrow RE_{\Sigma}$ is the **transition function**.
4. $q_{\text{start}} \in Q$ is the **start state**, and
5. $q_{\text{accept}} \subseteq Q$ is the **accept state**.
GNFA – Defining a Computation

A GNFA accepts a string \( w \in \Sigma^* \) if \( w = w_1w_2 \cdots w_k \) and there exists a sequence of states \( q_{start}q_1q_2 \cdots q_{accept} \), satisfying:

For each \( i, \ 1 \leq i \leq k, w_i \in L(R_i) \), where
\( R_i = \delta(q_{i-1}, q_i) \), or in other words, \( R_i \) is the expression on the arrow from \( q_i \) to \( q_{i+1} \).

Procedure **CONVERT**

Procedure **CONVERT** takes as input a GNFA \( G \) with \( k \) states.

If \( k = 2 \) then these 2 states must be \( q_{start} \) and \( q_{accept} \), and the algorithm returns \( \delta(q_{start}, q_{accept}) \).

If \( k > 2 \), the algorithm converts \( G \) to an equivalent \( G' \) with \( k - 1 \) states by use of the ripping procedure described before.

Recap

In this lecture we:

1. Motivated and defined regular expressions as a more concise and elegant method to represent regular Languages.
2. Proved that FA-s (Deterministic as well as Nondeterministic) and RE-s is identical by:
   2.1 Defined GNFA – s.
   2.2 Showed how to convert a DFA to a GNFA.
   2.3 Showed an algorithm to converted a GNFA with \( K \) states to an equivalent GNFA with \( K - 1 \) states.