COP4020.02 **Brandon Thompson**
Dr. Muhammed Abid

Notes COP4020: Programming Languages

1 Introduction

Mathematical models called "machines." Collection of successful inputs are called the language of the machines.

1.1 History

- Cantor (1845-1918) - Theory of sets (union, intersections, cardinality, etc) Discovered paradoxes such as the idea that infinity comes in different sizes, some set is bigger than the universal set.
- Hilbert (1906-1978) Methodology for finding proofs. Each true proposition is provided with a rigorous proof in which every line is either an axiom or follows from the axioms and previously proved theorems by a specified small set of rules of inference.
- Godel (1906-1978) Incompleteness theorem. There was no algorithm to provide proofs for all true statements in mathematics. He showed that either there were some true statements in mathematics that had no proofs, or else there were some false statements that did have proofs.
- Church, Kleene, Post, Markov, von Neumann, Turing - Which statements have proofs? Building blocks of mathematical algorithms. Turing proved that there were mathematically definable fundamental questions about the machine itself that the machine could not answer.

1.2 Languages and Machines

The term computer is never used in this course. We study computers by building mathematical models called **machines**, and then studying their limitations by analyzing the types of inputs on which they operate successfully. The collection of successful inputs is called the **language** of the machine.

	Language Defined by	Corresponding Accepting Machine	Nondeterminism= Determinism	Language Closed Under	What Can Be Decided?	Examples of Applications
I.	Regular expression	Finite automaton, transition graph	Yes	Union, product, Kleene star, intersection, complement	Equivalence, emptiness, finiteness, membership	Text editors, sequential circuits, verification
II.	Context-free grammar	Pushdown automaton	No	Union, product, Kleene star	Emptiness, finiteness, membership	Parsing, compilers
III.	Type 0 grammar	Turing machine, Post machine, Pushdown automaton	Yes	Union, product, Kleene star	Not much	Computers

Figure 1.1: Page 434 in textbook.

2 Languages

- Mathematical models of computers
  - Analysis of the input **language**
    - study of their limitations

2.1 Definitions

- alphabet - a finite set of symbols, denoted Σ
- letter - an element of an alphabet
- word - a finite sequence of letters from the alphabet
- Λ (empty string) - a word without letters
- language - a set of words formed from the alphabet

Two words are considered the same if all their letters are the same and in the same order. There is a difference between the word that has no letters (Λ), and the language that has no words (Φ). It is not true that Λ is a word in the language Φ since this language has no words at all.

	English-Words	English-Sentences
alphabet	Σ={a,b,c,d,…}	Γ=words in dictionary + space + punctuation marks
letter	letter	word
word	word	sentence
language	all the words in the dictionary	all English sentences

Figure 2.1: Examples of the definitions.

Languages can be defined as a list of all of the words, or as a set of the rules **(grammar)**. The alphabet is always finite, but the set of words can be infinite. It must be possible in a finite time to determine if a word is in a language or not (algorithm).

$$\Sigma = x \quad L_1 = \{x, xx, xxx, xxxx, \ldots\}$$
$$\text{or}$$
$$L_1 = \{X^n | n = 1, 2, 3, \ldots\}$$

Concatenation is when two words, written down side by side, create a new word:

$$a = xxx \qquad b = xxx \qquad ab = xxxxx$$

When a word is concatenated, we are able to factor out the individual words.

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$
$$L_2 = \{\text{words that do not start with the letter0}\}$$

Length is the number of letters in a word:

$$\begin{aligned} \text{length}(xxxxx) &= 5 \\ \text{length}(1025) &= 4 \\ \text{length}(\Lambda) &= 0 \end{aligned}$$

Reverse will reverse the letters in a word:

$$\begin{aligned} \text{reverse}(xxx) &= xxx \\ \text{reverse}(1570) &= 751 \end{aligned}$$

A palindrome is a word that is the same reading forwards as it is reading backwards:

$$\begin{aligned} \Sigma &= \{a, b\} \\ \text{Palindrome} &:= \{\Lambda \text{ and } x | \text{reverse}(x) = x\} \\ &= \{\Lambda, a, b, aa, bb, aaa, aba, bab, bbb, \ldots\} \end{aligned}$$

2.2 Kleene Closure (Star)

Given an alphabet Σ, the closure of Σ (or Kleene star), denoted as Σ^*, is the language containing all words made up of finite sequences of letters from Σ, including the empty string Λ.

$$\begin{aligned} \Sigma &= \{x\} & \Sigma^* &= \{\Lambda, x, xx, xxx, \ldots\} \\ \Sigma &= \{0, 1\} & \Sigma^* &= \{\Lambda, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\} \\ \Sigma &= \{a, b, c\} & \Sigma^* &= \{\Lambda, a, b, c, aaa, aab, aac, aba, abb, abc, \ldots\} \end{aligned}$$

Let S be a set of words. S^* is the language formed by concatenating words from S, including the empty string (null string) Λ.

$$S = \{a, ab\} \qquad S^* = \{\Lambda, a, aa, ab, aaa, aab, aba, aaaa, \ldots\}$$

$$abaaababa \in S^*$$

$$ab|a|a|ab|ab|a \leftarrow \text{factors}$$

$$\begin{aligned} S^* = \{&\Lambda \text{ plus all sequences of a's and b's except those} \\ &\text{that start with b and those that contain a double b}\} \end{aligned}$$

Factoring is not always unique, if $S = \{xx, xxx\}$ and $xxxxxxx \in S^*$, then there are multiple ways to break up the word between the words in S.

The Kleene closure of an alphabet Σ always produces an infinite language unless Σ is empty. Only for infinite languages, the Kleene closure of two sets can end up being the same language even if the two sets started with were not.

$$S = \{a, b, ab\} \qquad T = \{a, b, bb\} \qquad S^* = T^*$$

2.3 Positive Closure (Plus)

The language with all concatenations that contain at least one word from S and one letter from Σ.
(S^* possibly without Λ.)

If Λ is a member of S, $S^* = S+$.

Equality of sets
$S = T$: $S \subset T$ and $T \subset S$

Subsets
$S \subset T$: for all x in S, x is also in T.

Theorem 2.1. *For any set of words S, we have $S^{**} = S^*$.*

Proof. **Case 1:** $S^{**} \subset S^*$
Every word in S^{**} is made up of factors from S^* (definition of Kleene star). Every word in S^* is made up of factors from S. Therefore every word in S^{**} is made up of factors from S.
Case 2: $S^* \subset S^{**}$
For any set A, we can show that $A \subset A^*$. Let w be a word in A. Then w is certainly in A^*. If we consider S^* as our set A, we can conclude $S^* \subset S^{**}$.
By definition of equality of sets, we have $S^* = S^{**}$. □

3 Recursive Definitions

New method to define languages using three steps:

1. Specify the basic words (base case).
2. Rules for constructing new words from ones already known (recursive case).
3. Declare that no word except those constructed by following rules 1 and 2 are in the language.

3.1 Examples

3.1.1 Factorial Function

1. $0! = 1$
2. $n! = n(n-1)!$
3. No word except those constructed by following rules 1 and 2 are in the language.

3.1.2 Recusive Programs

- $\Sigma+, \Sigma\{x\}$
    1. $x \in \Sigma+$
    2. If w is any word in $\Sigma+$, then xw is in $\Sigma+$.
    3. No word except those constructed by following rules 1 and 2 are in the language.
- $\Sigma^*, \Sigma = \{x\}$
    1. $\Lambda \in \Sigma^*$
    2. If w is any word in Λ^*, then xw is in Σ^*.
    3. No word except those constructed by following rules 1 and 2 are in the language.
- S-ODD
    1. $x \in$ S-ODD
    2. if w is any word in S-ODD then xxw is in S-ODD.
    3. No word except those constructed by following rules 1 and 2 are in the language.

4 Regular Expressions

Alphabets, languages, and "letters" can all be converted to another language using the Kleene closure property.

$$\Sigma = \{x\} \qquad \Sigma^* = \{\Lambda, x, xx, xxx, \ldots\}$$
$$\text{or}$$
$$\{x\}^* = \{\Lambda, x, xx, xxx, \ldots\}$$

$$S = \{xx, xxx\} \qquad S^* = \{\Lambda, xx, xxx, xxxx, \ldots\}$$
$$\text{or}$$
$$\{xx, xxx\}^* = \{\Lambda, xx, xxx, xxxx, \ldots\}$$

$$\text{language}(x^*) = \{\Lambda, x, xx, xxx, \ldots\}$$
$$\text{or}$$
$$x^* = \{\Lambda, x, xx, xxx, \ldots\}$$

Defining language(x^*) as some sequence of x's (or none) gives us the basis for writing regular expressions. Defining language($a\ b^*$) = language(ab^*) is words with an a followed by any number of b's: $\{a, ab, abb, abbb, \ldots\}$. Whereas language($(ab)^*$) defines $\{\Lambda, ab, abab, ababab, \ldots\}$.

The plus sign + is also used for the **or** operation. $x + y$ means to choose either x or y. Consider the language:

$$\Sigma = \{a, b, c\}$$
$$T = \{a, c, ab, cb, abb, cbb, abbb, cbbb, \ldots\}$$
$$T = \text{language}((a + c)b^*)$$

which constructs words with either an a or a c followed by any number of b's.
$(a + b)^*$ is all words formed from the alphabet $\{a, b\}$.
Definition: Given an alphabet Σ, the set of regular expressions is defined by the following rules:

1. For every letter in Σ. The r is a regular expression. Λ is a regular expression.
2. If r_1 and r_2 are in regular expressions, then so are:
   (a) (r_1)
   (b) $r_1 r_2$

(c) $r_1 + r_2$

(d) r_1^*

3. Nothing else is a regular expression.

Remark: r_1^+ is not included because $r_1^+ = r_1 r_1^*$.

Languages defined by regular expressions are referred to as regular languages. This language defining tool is of very limited capacity in that there are many interesting languages that cannot be defined via a regular expressions.

4.1 Equivalence of Regular Expressions

$r_1 = r_2$ if and only if language(r_1)=language(r_2) For example: $(a + b)^* a (a + b)^*$ is all words that have at least one a. b^* is all words without an a. Thus:

$$(a + b)^* = (a + b)^* a (a + b)^* + b^*$$

All strings = (all strings with an a) + (all strings without an a).
In general, concatenation is distributive over the + operation.

$$r_1(r_2 + r_3) = r_1 r_2 + r_1 r_3$$
$$(r_1 + r_2) r_3 = r_1 r_3 + r_2 r_3$$

However the distributive law must be used with caution: $(ab)^* \neq a^* b^*$

4.2 Language associated with a regular expression

By following these rules you can make a regular language from a regular expression:

1. The language associated with a regular expression that is just a single letter is that one letter word alone. The language associated with Λ is $\{\Lambda\}$, a one word language.

2. If L_1 is the language associated with the regular expression r_1 and L_2 is the language associated with the regular expression r_2:

   (a) The product $L_1 L_2$ is the language associated with the regular expression $r_1 r_2$.

   (b) the union $L_1 + L_2$ is the language associated with the regular expression $r_1 + r_2$.

   (c) The Kleene closure of L_1, written L_1^*, is the language associated with the regular expression r_1^*.

For all regular expressions, there is some language associated with it.

4.3 Finite languages are regular

Theorem 4.1. *Let L be a finite language. There is a regular expression that defines it*

Proof. Write each word in L, and write a + between regular expressions. □

The regular expression that is defined by this algorithm is not necessarily unique. This algorithm does not work for infinite languages. Regular expressions must be finite, even if the language defined is infinite.

5 Finite Automata (FA)

A finite automaton is the following three things:

1. A finite set of states, one of which is designated as the initial state, called the **start state**, and some (maybe none) of which are designated the **final states** (or **accepting states**)
2. An **alphabet** Σ of input letters
3. A finite set of **transitions** that tell for each state and for each letter of input alphabet which state to go to next:

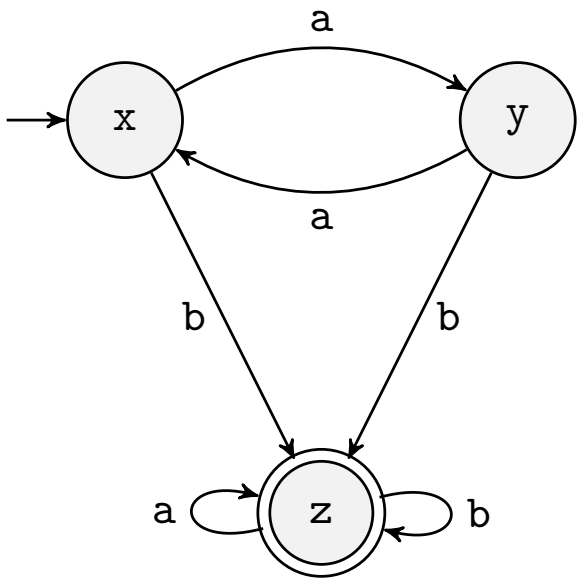
$$\texttt{state} \xrightarrow{\texttt{letter}} \texttt{state}$$

The language is **defined** or **accepted** by a finite automaton is the set of words that end in a final state. If w is in the language defined by a finite automaton, then we also say that the finite automaton **accepts** w.

Finite automaton work by being presented with an input string of letters that it reads letter by letter starting from the leftmost side. Beginning at the start state, the letters determine a sequence of states. The sequence ends when the last input letter has been read. The set of all strings that end in a final state is called the language defined by the finite automaton. The set of strings which do not end in a final state, we call not accepted or rejected by the finite automaton.

5.1 Transition Diagram

	a	b
x	y	z
y	x	x
z	z	z



The regular expression of this FA is $a^*b(a+b)^*$.

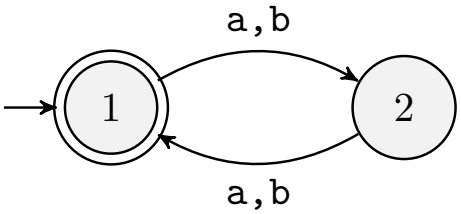
5.2 Studying Automata

1. Starting with a finite automaton (FA), analyze it to determine the language it accepts.
2. Starting from a language, build an FA.

Example of the second method: The language of all words with an even number of letters over the alphabet $\{a, b\}$:

- Two states: 1 - even number, 2 - odd number.
- Start state: 1
- Final state: 1
- Transitions:

	a	b
1	2	2
2	1	1



5.3 From Languages to Finite Automata

There is not necessarily a unique FA that accepts a given language. Every language that can be accepted by an FA can be defined by a regular expression and, conversely, every language that can be defined by a regular expression can be accepted by some FA. However, there are languages that are neither definable by regular expression nor accepted by an FA.

5.4 From Finite Automata to Languages

Characterize states by the purposes they serve.

There are two "direct" paths to get from state 1 to state 4, either the path $1 \to 2 \to 4$

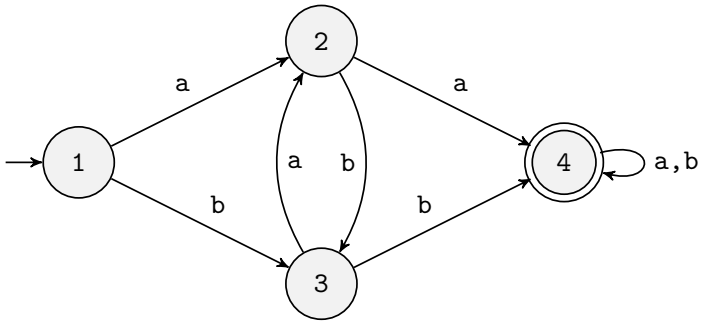


Figure 5.2: Finite Automata accepting the language $(a+b)^*(aa+bb)(a+b)^*$

or from $1 \to 3 \to 4$. The loop between states 2 and 3 signify that there could be any number of letters at the start, this is the first $(a+b)^*$. Only when the pattern aa or bb appears we can continue to state 4, $(aa+bb)$, once in state 4, any number of letters can be accepted this is the second $(a+b)^*$.

6 Transition Graphs

Transition graphs (TG) are similar to Finite Automata (FA), the distinction being that transition graphs can have blocks of inputs on one transition, and do not need to have garbage collection for unwanted inputs.

6.1 Definition and example

A Transition Graph is the collection of three things:

1. A finite set of states, at least one of which is designated as the **start state**, and some (maybe none) of which are designated the **final states** or **accepting states**
2. An **alphabet** Σ of input letters from which input strings are formed.
3. A finite set of **transitions**(edge labels) that show how to go from some states to some others, based on reading specified substrings on input letters (possibly even the null string Λ).

A **successful path** is a series of edges beginning at some start state and ending at a final state. The concatenation of all the substrings that label the edges in the path is a word **accepted** by this machine. The set of words accepted is the **language** of the transition graph.

It is important to note that every FA is a TG but not every TG is an FA.

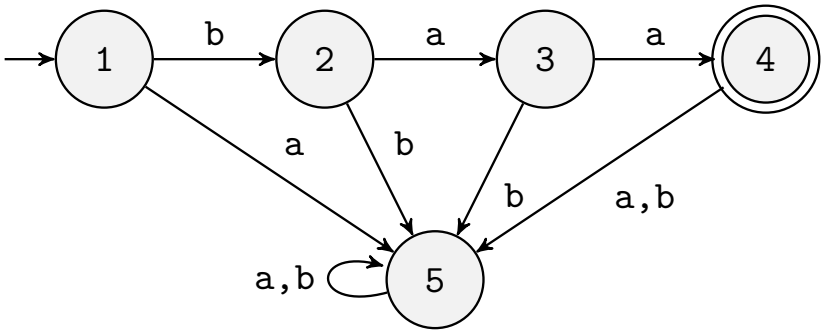


Figure 6.1: Finite Automata accepting the language $\{baa\}$.

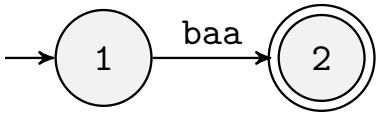


Figure 6.2: Transition Graph accepting the language $\{baa\}$.

On an FA it is not possible for any input to crash because there is always an outgoing edge for every possible letter from each state. As long as there are unread letters, progress is possible. TG's can crash when there is not an edge for the word to follow.

A TG in which some start state is also a final state will always accept the null string Λ.

6.2 Generalized Transition Graphs

A generalized transition graph (GTG) is the collection of three things:

1. A finite set of states, at least one of which is designated as the **start state**, and some (maybe none) of which are designated the **final states** or **accepting states**
2. An **alphabet** Σ of input letters from which input strings are formed.
3. A finite set of edges connecting some pairs of states, each labeled with a regular expression.

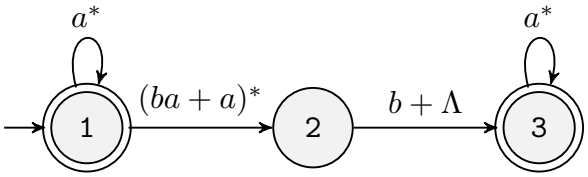


Figure 6.3: Generalized Transition Graph of all words without two b's in a row.

6.3 Nondeterministic Finite Automata

If there are multiple transitions that could be chosen simultaneously, the machine is considered **Nondeterministic**. Human choices become a factor in selecting the path; the machine does not make all its own determinations.

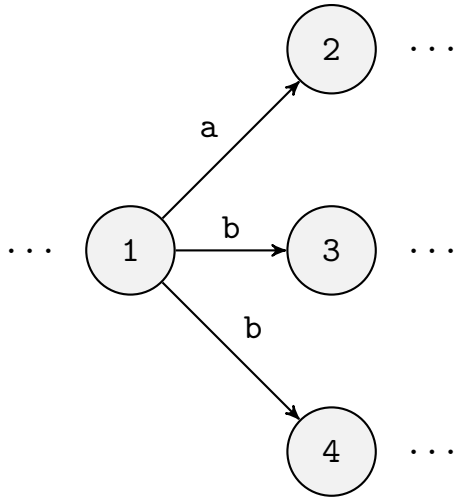


Figure 6.4: Section of Nondeterministic finite automata.

7 Kleene's Theorem

Theorem 7.1. *Let A, B, B be sets such that $A \subset B, B \subset C, C \subset A$.*
Then $A = B = C$.

Proof. Let x be an element of B. $x \in C$ (because $B \subset C$). Also $x \in A$ (because $C \subset A$). Thus $B \subset C$, and we can conclude $A = B$. By analogy, $B = C$ and thus all are equal. □

Regular expressions, finite automate, and transitions graphs each define a set of languages.

Lemma 7.1. *Every language that can be defined by a finite automaton can also be defined by a transition graph. $FA \subset TG$*

Proof. By definition (6.1), every finite automaton is itself already a transition graph. Therefore, any language that has been defined by an FA has already been defined by a TG. □

Lemma 7.2. *Every language that can be defined by a transition graph can also be defined by a regular expression. $TG \subset RE$*

Proof. By **constructive algorithm**, which works:

1. Add (if necessary) a unique start state without incoming edges and a unique final state without outgoing edges (connect with Λ). For each state repeat steps two and three.
2. Use bypass and state elimination operation to reduce states.
3. Combine edges that have the same starting and ending state.
4. Combine edges from start and end state. The label on the only remaining edge is the regular expression result. If there is none, the regular expression is Φ.

□

Example 7.1. *This is a small example proving Lemma 7.2.*
The final regular expression is: $(aa + bb)(a + b)^*(aa + bb)$.

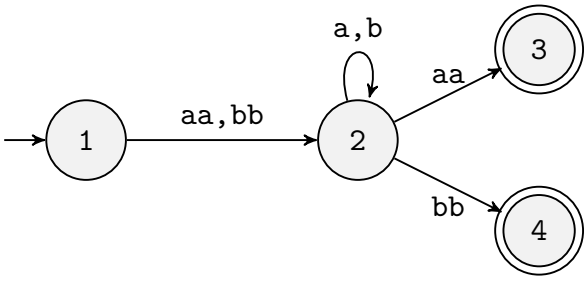


Figure 7.1: Starting transition graph.

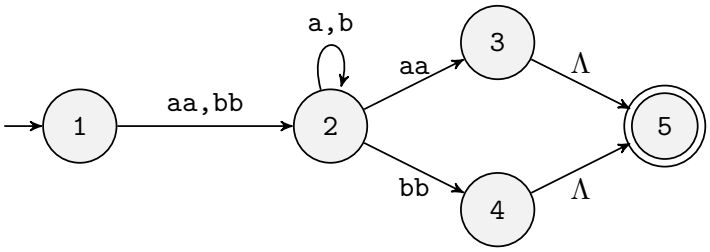


Figure 7.2: Add final state 5 and remove acceptance from states 3 and 4.

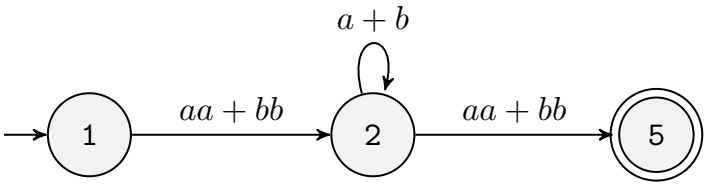


Figure 7.3: Bypass states 3 and 4. Convert edges to regular expressions.

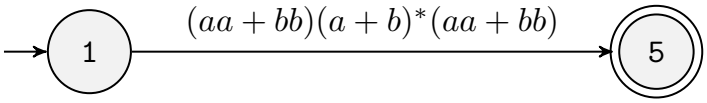


Figure 7.4: Bypass state 2. Regular expression complete.

Lemma 7.3. *Every language that can be defined by a regular expression can also be defined by a finite automaton.* $RE \subset FA$

Proof. 1. There is an FA that accepts only the empty word Λ and an FA that accepts only a single letter.

2. If there is an FA that accepts the language defined by r_1, and there is an FA that accepts the language defined by r_2 then there is an FA that accepts the language $r_1 + r_2$.

3. If there is an FA that accepts the language defined by r_1, and there is an FA that accepts the language defined by r_2 then there is an FA that accepts the language $r_1 r_2$.

4. If there is an FA that accepts the language defined by r then there is an FA that accepts the language defined by r^*.

Thus for every regular expression, we can construct an FA. □

Kleene's Theorem states that any language that can be defined by a regular expression, finite automaton, or transition graph can be defined by all three methods.