Consider the problem of a $n$-input AND function. The input ($x$) is a string $n$-digits long with $\Sigma = \{0, 1\}$ and has an output ($y$) which is the logical AND of all the elements of $x$.

Formulate a language that describes the above problem.
ECE-374-B: Lecture 1 - Regular Languages

Lecturer: Nickvash Kani
August 24, 2023

University of Illinois at Urbana-Champaign
Pre-lecture brain teaser

Consider the problem of a \( n \)-input \textit{AND} function. The input \( x \) is a string \( n \)-digits long with \( \Sigma = \{0, 1\} \) and has an output \( y \) which is the logical \textit{AND} of all the elements of \( x \).

Formulate a \textbf{language} that describes the above problem.

\[
\begin{align*}
0 \cdot 0 &= 0, \\
0 \cdot 1 &= 0, \\
1 \cdot 0 &= 1, \\
1 \cdot 1 &= 1
\end{align*}
\]
Consider the problem of a \( n \)-input AND function. The input \((x)\) is a string \( n \)-digits long with \( \Sigma = \{0, 1\} \) and has an output \((y)\) which is the logical AND of all the elements of \( x \).

Formulate a **language** that describes the above problem.

\[
L_{\text{AND}_n} = \left\{ 0|0, 1|1, 0 \cdot 0|0, 0 \cdot 1|0, 1 \cdot 0|0, 1 \cdot 1|1, \ldots \right\}
\]

This is an example of a regular language which we’ll be discussing today.
Consider the problem of a $n$-input **AND** function. The input $(x)$ is a string $n$-digits long with $\Sigma = \{0, 1\}$ and has an output $(y)$ which is the logical **AND** of all the elements of $x$.

Formulate a **language** that describes the above problem.

$$L_{AND_N} = \left\{ 0|0, \quad 1|1, \quad 0 \cdot 0|0, \quad 0 \cdot 1|0, \quad 1 \cdot 0|0, \quad 1 \cdot 1|1 \right\}$$

This is an example of a regular language which we’ll be discussing today.
Chomsky Hierarchy

- Regular
- Context free
- Context sensitive
- Recursively enumerable (decidable)
- Non recursively enumerable (undecidable)

\[ L_{w_0} + L_{w_1} + L_{w_2} + \ldots + L_{w_n} = L \]

\[ |\{ w_0, w_1, w_2 \}| = \infty \]
Chomsky Hierarchy

non recursively enumerable (undecidable)

recursively enumerable (decidable)

context sensitive

context free

regular
Regular Languages
Theorem (Kleene’s Theorem)

A language is regular if and only if it can be obtained from finite languages by applying the three operations:

- Union
- Concatenation
- Repetition

a finite number of times.
Regular Languages

A class of simple but useful languages. The set of regular languages over some alphabet \( \Sigma \) is defined inductively.

**Base Case**

- \( \emptyset \) is a regular language.
- \( \{\epsilon\} \) is a regular language.
- \( \{a\} \) is a regular language for each \( a \in \Sigma \). Interpreting \( a \) as a string of length 1.
Inductive step:

We can build up languages using a few basic operations:

- If $L_1, L_2$ are regular then $L_1 \cup L_2$ is regular.
- If $L_1, L_2$ are regular then $L_1L_2$ is regular.
- If $L$ is regular, then $L^* = \bigcup_{n \geq 0} L^n$ is regular. The $\cdot^*$ operator name is Kleene star.
- If $L$ is regular, then so is $\overline{L} = \Sigma^* \setminus L$.

Regular languages are closed under operations of union, concatenation and Kleene star.
Some simple regular languages

Lemma
If $w$ is a string then $L = \{w\}$ is regular.

Example: $\{aba\}$ or $\{abbabbab\}$. Why?

Base: $\emptyset$

$\emptyset \rightarrow \emptyset$

$\emptyset \rightarrow \emptyset$

$L_{aba} = L_a \cdot L_b \cdot L_a$

$L_{abbabab} = L_a \cdot L_b \cdot L_b \cdot L_a \cdot L_b \cdot L_b \cdot L_a \cdot L_b$
Some simple regular languages

**Lemma**
If \( w \) is a string then \( L = \{ w \} \) is regular.

**Example:** \( \{aba\} \) or \( \{abbabbab\} \). Why?

**Lemma**
Every finite language \( L \) is regular.

Examples: \( L_3 = \{a, abaab, aba\} \). \( L = \{w \mid |w| \leq 100\} \). Why?

\[ L_a \cup L_aabaab \cup Laba = L_3 \]
Regular Languages

Have basic operations to build regular languages.

**Important:** Any language generated by a finite sequence of such operations is regular.

**Lemma**

Let $L_1, L_2, \ldots$ be regular languages over alphabet $\Sigma$. Then the language $\bigcup_{i=1}^{\infty} L_i$ is not necessarily regular.
Regular Languages

Have basic operations to build regular languages.

Important: Any language generated by a finite sequence of such operations is regular.

Lemma
Let $L_1, L_2, \ldots$, be regular languages over alphabet $\Sigma$. Then the language $\bigcup_{i=1}^{\infty} L_i$ is not necessarily regular.

Note: Kleene star (repetition) is a single operation!
Example: The language $L_{01} = 0^i1^j$ for all $i, j \geq 0$ is regular:

$L_{01} = L_0^* \cdot L_1^*$
1. \( L_1 = \{ 0^i \mid i = 0, 1, \ldots, \infty \} \). The language \( L_1 \) is regular. T/F?

\[
L_1 = \{0^i \mid i = 0, 1, \ldots, \infty \} = \bigcup_{n > 0} L_0^* = \{0^0, 0, 0^2, 0^3, \ldots\}
\]

\[
L_1 = \{0^i \mid i = 0, 1, \ldots, \infty \} = \{\varepsilon, 0, 00, \ldots\}
\]
1. \( L_1 = \{0^i \mid i = 0, 1, \ldots, \infty\} \). The language \( L_1 \) is regular. T/F?

2. \( L_2 = \{0^{17i} \mid i = 0, 1, \ldots, \infty\} \). The language \( L_2 \) is regular. T/F?

\[
L_2 = (L_0 \cdot L_0 \cdot \ldots \cdot L_0)^* \tag{22 times}
\]
Rapid-fire questions - regular languages

1. $L_1 = \{0^i \mid i = 0, 1, \ldots, \infty\}$. The language $L_1$ is regular. T/F?

2. $L_2 = \{0^{17i} \mid i = 0, 1, \ldots, \infty\}$. The language $L_2$ is regular. T/F?

3. $L_3 = \{0^i \mid i \text{ is divisible by 2, 3, or 5}\}$. $L_3$ is regular. T/F?

\[
L_{20}s = (L_0 \cdot L_0)^* \\
L_{30}s = (L_0 \cdot L_0 \cdot L_0)^* \\
L_{50}s = (L_0 \cdot L_0 \cdot L_0 \cdot L_0 \cdot L_0)^* \\
L_3 = L_{20}s \cup L_{30}s \cup L_{50}s
\]
Rapid-fire questions - regular languages

1. $L_1 = \{0^i \mid i = 0, 1, \ldots, \infty \}$. The language $L_1$ is regular. T/F?

2. $L_2 = \{0^{17i} \mid i = 0, 1, \ldots, \infty \}$. The language $L_2$ is regular. T/F?

3. $L_3 = \{0^i \mid i \text{ is divisible by 2, 3, or 5} \}$. $L_3$ is regular. T/F?

4. $L_4 = \{w \in \{0, 1\}^* \mid w \text{ has at most 2 1s} \}$. $L_4$ is regular? T/F?
Regular Expressions
Regular Expressions

A way to denote regular languages

- simple patterns to describe related strings
- useful in
  - text search (editors, Unix/grep, emacs)
  - compilers: lexical analysis
  - compact way to represent interesting/useful languages
  - dates back to 50’s: Stephen Kleene who has a star names after him \(^1\).
Inductive Definition

A *regular expression* $r$ over an alphabet $\Sigma$ is one of the following:

**Base cases:**

- $\emptyset$ denotes the language $\emptyset$
- $\epsilon$ denotes the language $\{\epsilon\}$
- $a$ denote the language $\{a\}$.

**Inductive cases:** If $r_1$ and $r_2$ are regular expressions denoting languages $R_1$ and $R_2$ respectively then,

- $(r_1 + r_2)$ denotes the language $R_1 \cup R_2$
- $(r_1 \cdot r_2) = r_1 \cdot r_2 = (r_1 r_2)$ denotes the language $R_1 R_2$
- $(r_1)^*$ denotes the language $R_1^*$
### Regular Languages vs Regular Expressions

**Regular Languages**

- $\emptyset$ regular
- $\{\epsilon\}$ regular
- $\{a\}$ regular for $a \in \Sigma$
- $R_1 \cup R_2$ regular if both are
- $R_1R_2$ regular if both are
- $R^*$ is regular if $R$ is

**Regular Expressions**

- $\emptyset$ denotes $\emptyset$
- $\epsilon$ denotes $\{\epsilon\}$
- $a$ denote $\{a\}$
- $r_1 + r_2$ denotes $R_1 \cup R_2$
- $r_1 \cdot r_2$ denotes $R_1R_2$
- $r^*$ denote $R^*$

Regular expressions denote regular languages — they explicitly show the operations that were used to form the language

$$R_i = L_a \cup L_b = \{ \epsilon, 1, 63 \} = a \cdot b$$
Notation and Parenthesis

- For a regular expression $r$, $L(r)$ is the language denoted by $r$. Multiple regular expressions can denote the same language!

Example: $(0 + 1)$ and $(1 + 0)$ denotes same language $\{0, 1\}$

\[
0 + 0 + 0 + 0 + 0 + \cdots + 0 = \{0, 1\}
\]
Notation and Parenthesis

• For a regular expression \( r \), \( L(r) \) is the language denoted by \( r \). Multiple regular expressions can denote the same language!
  
  **Example:** \((0 + 1)\) and \((1 + 0)\) denotes same language \( \{0, 1\} \)

• Two regular expressions \( r_1 \) and \( r_2 \) are **equivalent** if \( L(r_1) = L(r_2) \).
Notation and Parenthesis

• For a regular expression $r$, $L(r)$ is the language denoted by $r$. Multiple regular expressions can denote the same language!

  **Example:** $(0 + 1)$ and $(1 + 0)$ denotes same language $\{0, 1\}$

• Two regular expressions $r_1$ and $r_2$ are **equivalent** if $L(r_1) = L(r_2)$.

• Omit parenthesis by adopting precedence order: $\cdot$, $\star$, +.

  **Example:** $r^*s + t = ((r^*)s) + t$
Notation and Parenthesis

• For a regular expression $r$, $L(r)$ is the language denoted by $r$. Multiple regular expressions can denote the same language!

  **Example:** $(0 + 1)$ and $(1 + 0)$ denotes same language $\{0, 1\}$

• Two regular expressions $r_1$ and $r_2$ are equivalent if $L(r_1) = L(r_2)$.

• Omit parenthesis by adopting precedence order: $\ast, \cdot, +$.

  **Example:** $r^*s + t = ((r^*)s) + t$

• Omit parenthesis by associativity of each operation.

  **Example:** $rst = (rs)t = r(st)$,
  
  $r + s + t = r + (s + t) = (r + s) + t$. 
Notation and Parenthesis

• For a regular expression \( r \), \( L(r) \) is the language denoted by \( r \). Multiple regular expressions can denote the same language!

  **Example:** \((0 + 1)\) and \((1 + 0)\) denotes same language \(\{0, 1\}\)

• Two regular expressions \( r_1 \) and \( r_2 \) are **equivalent** if \( L(r_1) = L(r_2) \).

• Omit parenthesis by adopting precedence order: \(*, \cdot, +\).

  **Example:** \( r^*s + t = ((r^*)s) + t \)

• Omit parenthesis by associativity of each operation.

  **Example:** \( rst = (rs)t = r(st) \),
  \( r + s + t = r + (s + t) = (r + s) + t \).

• **Superscript \( +\).** For convenience, define \( r^+ = rr^* \). Hence if \( L(r) = R \) then \( L(r^+) = R^+ \).
Notation and Parenthesis

- For a regular expression $r$, $L(r)$ is the language denoted by $r$. Multiple regular expressions can denote the same language!
  
  **Example:** $(0 + 1)$ and $(1 + 0)$ denotes same language $\{0, 1\}$

- Two regular expressions $r_1$ and $r_2$ are equivalent if $L(r_1) = L(r_2)$.

- Omit parenthesis by adopting precedence order: $\ast, \cdot, +$.
  
  **Example:** $r^*s + t = ((r^*)s) + t$

- Omit parenthesis by associativity of each operation.
  
  **Example:** $rst = (rs)t = r(st)$, 
  
  $r + s + t = r + (s + t) = (r + s) + t$.

- Superscript $+$. For convenience, define $r^+ = rr^*$. Hence if $L(r) = R$ then $L(r^+) = R^+$.

- Other notation: $r + s$, $r \cup s$, $r|s$ all denote union. $rs$ is sometimes written as $r \cdot s$. 
Some examples of regular expressions
Creating regular expressions

1. All strings that end in 1011?

\((0+1)^*1011\)
Creating regular expressions

1. All strings that end in 1011?
2. All strings except 11?

\[(0+1)^* - 11\]

\[(0+1)^* 00 + (0+1)^* 01 + (0+1)^* 10\]

\[\sum \{0+1\} + 00 + 01 + 10 + (0+1)^* \]

\[+ (0+1)(0+1)(0+1)^* \]

\[+ (0+1)(0+1) (0+1)(0+1)^* \]
1. All strings that end in 1011?
2. All strings except 11?
3. All strings that do not contain 000 as a subsequence?

\[
\begin{align*}
\epsilon + 1^*01^* + 1^*001^*01^* \\
\epsilon^* (0 + \epsilon)^* \langle 1 \rangle^* (0 + \epsilon)^* \langle 1 \rangle^* \langle 0 \rangle^* \langle 1 \rangle^* (0 + \epsilon)^* \langle 1 \rangle^* 
\end{align*}
\]
Creating regular expressions

1. All strings that end in 1011?
2. All strings except 11?
3. All strings that do not contain 000 as a subsequence?
4. All strings that do not contain the substring 10?
Interpreting regular expressions

1. \((0 + 1)^*\): All binary strings
Interpreting regular expressions

1. \((0 + 1)^*:\)
2. \((0 + 1)^*001(0 + 1)^*:\)

All strings that have \textit{001} as a substring.
Interpreting regular expressions

1. \((0 + 1)^*:\)
2. \((0 + 1)^*001(0 + 1)^*:\)
3. \(0^* + (0^*10^*10^*10^*)^*:\)

All string whose #1's is divisible by 3
Interpreting regular expressions

1. \((0 + 1)^*:\)
2. \((0 + 1)^*001(0 + 1)^*:\)
3. \(0^* + (0^*10^*10^*10^*)^*:\)
4. \((\epsilon + 1)(01)^*(\epsilon + 0):\)

Alternating
Consider the problem of an \( n \)-input \textit{AND} function. The input \((x)\) is a string \( n \)-digits long with an input alphabet \( \Sigma_i = \{0,1\} \) and has an output \((y)\) which is the logical \textit{AND} of all the elements of \( x \). We know the language used to describe it is:

\[
L_{\text{AND}_N} = \left\{ 0 \cdot |0, 1 \cdot |1, 0 \cdot 0 \cdot |0, 0 \cdot 1 \cdot |0, 1 \cdot 0 \cdot |0, 1 \cdot 1 \cdot |1 \right\}
\]

Formulate the regular expression which describes the above language:
Tying everything together

Consider the problem of a \( n \)-input **AND** function. The input \((x)\) is a string \(n\)-digits long with an input alphabet \(\Sigma_i = \{0, 1\}\) and has an output \((y)\) which is the logical **AND** of all the elements of \(x\). We know the language used to describe it is:

\[
L_{\text{AND}_N} = \left\{ \begin{array}{ll}
0 \cdot |0, & 1 \cdot |1, \\
0 \cdot 0 \cdot |0, & 0 \cdot 1 \cdot |0, & 1 \cdot 0 \cdot |0, & 1 \cdot 1 \cdot |1 \\
\vdots & \vdots & \vdots & \vdots \\
(0 \cdot)^n |0, & (0 \cdot)^{n-1} |0, & \ldots & (1 \cdot)^n |1 \ldots
\end{array} \right\}
\]

Formulate the regular expression which describes the above language: \(\Sigma = \{0, 1, '.', '|'\}\)

\[
r_{\text{AND}_N} = \left(\left(0 \cdot + 1 \cdot\right)^* 0 \cdot \left(0 \cdot + 1 \cdot\right)^* |0\right) + \left(1 \cdot\right)^* |1
\]

all output 0 instances

all output 1 instances
Regular expressions in programming
One last expression....
Bit strings with odd number of 0s and 1s

The regular expression is

\[00^+11\]

\[\downarrow\]

\[00^+11^+\ (01^+10)(00^+11^+)\]

\[\uparrow\]

\[\ast\ (01^+10)\]

(Solved using techniques to be presented in the following lectures...)
Bit strings with odd number of 0s and 1s

The regular expression is

\[(00 + 11)^* (01 + 10) \left( 00 + 11 + (01 + 10)(00 + 11)^* (01 + 10) \right)^* \]
Bit strings with odd number of 0s and 1s

The regular expression is

\[(00 + 11)^*(01 + 10)\]

\[\left(00 + 11 + (01 + 10)(00 + 11)^*(01 + 10)\right)^*\]

(Solved using techniques to be presented in the following lectures...)