Pre-lecture brain teaser

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.
CS/ECE-374: Lecture 2 - Regular Languages

Lecturer: Nickvash Kani
Chat moderator: Samir Khan
January 28, 2021

University of Illinois at Urbana Champaign
Pre-lecture brain teaser

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 $x$. $n > 0$

Formulate a language that describes the above problem.

$$L = \{ "0010", "0100", "1000", "1111", "0010", \ldots \ldots \ldots\ldots , "1111", "000010", \ldots \ldots \ldots \ldots , "1111", \ldots \}$$
Consider the problem of a \textit{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.

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

- Non-recursively-enumerable
- Recursively-enumerable
- Context-sensitive
- Context-Free
- Regular
Chomsky Hierarchy

Non-recursively-enumerable

Recursively-enumerable

Context-sensitive

Context-Free

Regular •
Regular Languages
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 string of length 1.

$\{ab, ba\}$
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 $L^* = \cup_{n \geq 0} L^n$ 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.
Regular Languages

Have basic operations to build regular languages.

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

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

**Example:**
Define $L_i = \{0^i 1^i \}^*$ One string in each $L_i$ is $0^* 1^*$. Let $L = \bigcup_{i=1}^{\infty} L_i = \{0^n 1^n | n \geq 0\}$ Which is $\varepsilon 0^* 1^* 010101$ and not regular.
Lemma
If $w$ is a string then $L = \{w\}$ is regular.

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

\[
L_a = \{a^3\} \quad \{aba^3\} = L_aL_aL_a
\]

\[
L_b = \{b^3\}
\]
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: \( L = \{a, abaab, aba\} \). \( L = \{w \mid |w| \leq 100\} \). Why?

\[
R = \{0, 1, 2, 3, 4, 5\} \]

\[
0.00000000 \quad 0.12345678
\]
Rapid-fire questions - regular languages

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

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

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

\[
L_0 = \{ \epsilon \} \\
L_{120} = \{ \epsilon, 0 \} \cdot L_0 \ldots \cdot L_0 \quad \text{(21 times)} \\
L_2 = \langle L_{120}, 0 \rangle
\]

\[\text{operation}\]

\[\text{TRUE}\]
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?

$L_0 = \{0^3, \infty 0^3\}$
$L_0 L_0 = \{0^6, \infty 0^6\}$
$L_1 = L_2 \cup L_3 \cup L_3$

$L_2 = (L_0 L_0)^*$ \text{ reg}
$L_2 = (L_0 L_0 L_0)^*$ \text{ reg}
$L_2 = (L_0 L_0 L_0 L_0 L_0)^*$ \text{ reg}
Rapid-fire questions - regular languages

4. \( L_4 = \{ w \in \{0,1\}^* : \text{w has at most 3 1s} \} \). \( L_4 \) is regular. \( \text{TRUE} \)

\( L_0 = \{0^*\} \quad L_e = \{e^3\} \quad L_e1 = L_e \cup L_1 \)
\( L_1 = \{1^*\} \quad L_4 = L_0 L_e L_0 L_e L_0 L_e L_0 L_0^* \)

\( L_{4-3} = L_0^* L_1 L_0^* L_1 L_0^* L_1 L_0^* \leq \text{all strings with 3 1s} \)

\( L_{4-2} = L_0^* L_1 L_0^* L_1 L_0^* L_4-1 = \ldots \leq \text{all strings with 2 1s} \)

\( L_{4-0} = \ldots \leq 0^*1 \)

\( L_0 = L_{4-0} \cup L_{4-1} \cup L_{4-2} \cup L_{4-3} \)
Regular Expressions

$L_1L_2 = \{ xy \mid x \in L_1, y \in L_2 \}$
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

<table>
<thead>
<tr>
<th>Regular Languages</th>
<th>Regular Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$ regular</td>
<td>$\emptyset$ denotes $\emptyset$</td>
</tr>
<tr>
<td>${\varepsilon}$ regular</td>
<td>$\varepsilon$ denotes ${\varepsilon}$</td>
</tr>
<tr>
<td>${a}$ regular for $a \in \Sigma$</td>
<td>$a$ denote ${a}$</td>
</tr>
<tr>
<td>$R_1 \cup R_2$ regular if both are</td>
<td>$r_1 + r_2$ denotes $R_1 \cup R_2$</td>
</tr>
<tr>
<td>$R_1R_2$ regular if both are</td>
<td>$r_1 \cdot r_2$ denotes $R_1R_2$</td>
</tr>
<tr>
<td>$R^*$ is regular if $R$ is</td>
<td>$r^<em>$ denote $R^</em>$</td>
</tr>
</tbody>
</table>

Regular expressions denote regular languages — they explicitly show the operations that were used to form the language
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 $(1 + 0)$ denote same language $\{0, 1\}$
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 \((1 + 0)\) denote same language \( \{0, 1\} \)

• Two regular expressions \( r_1 \) and \( r_2 \) are **equivalent** if \( L(r_1) = L(r_2) \).
• 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 $(1 + 0)$ denote 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: *, concatenate, +.

   **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 $(1 + 0)$ denote 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: $\ast$, concatenate, $\ast$.

**Example:** $r\ast s + t = ((r\ast)s) + t$

- Omit parenthesis by associativity of each of these operations.

**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 $(1 + 0)$ denote 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: $\ast$, concatenate, $\ast$.

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

• Omit parenthesis by associativity of each of these operations.

Example: $rst = (rs)t = r(st)$,

$r + s + t = r + (s + t) = (r + s) + t$.

• Superscript $\ast$. 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 \((1 + 0) \) denote 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: \( * \), concatenate, \( + \).
  
  **Example:** \( r^*s + t = ((r^*)s) + t \)

- Omit parenthesis by associativity of each of these operations.
  
  **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 \( L(r^+) = R^+ \).
Some examples of regular expressions
Interpreting regular expressions

1. $(0 + 1)^*$: All binary strings

$L_0^2$ \hspace{1cm} $(L_0 \cup L_1)^*$

$L_1 = L_0 L_1 L_1^*$
Interpreting regular expressions

1. \((0 + 1)^*\): Could be \(\epsilon\)
2. \((0 + 1)^*001(0 + 1)^*\):
   
   All strings with 001 as substring

0101 001
Interpreting regular expressions

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

\[ L = \{0 \ 001000010010 \ 011111\} \]

\# of 1's divisible by 3
Interpreting regular expressions

1. $(0 + 1)^*$:
2. $(0 + 1)^*001(0 + 1)^*$:
3. $0^* + (0^{*}10^{*}10^{*}10^{*})^*$:
4. $(\epsilon + 1)(01)^*(\epsilon + 0)$:
Creating regular expressions

1. All strings that end in 1011?

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

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

All strings 3 characters or greater

\( \varepsilon + 0^* + 1^* + 0 + 10 + (0 + 1)^3 (0 + 1)^* \)
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?

$1^* \ (\varepsilon + 0) \ 1^* \ (\varepsilon + 0) \ 1^*$

00: $\varepsilon \ 0 \ 0 \ \varepsilon = 00$

0: $\varepsilon \ \varepsilon \ \varepsilon \ 0 \ \varepsilon = 0$

Example: gabtvc
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?

\[ 0^* 1^* \]
Tying everything together

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 the regular expression which describes the above language:

\[
L = \{010^* 111^* 0010^* 0010^* 0010^* 111^* \ldots \}
\]

Case where \( y = 1 \):

\[
P \dagger \dagger 1
\]

Case where \( y = 0 \):

\[
1^* (0+1)^* 0 (C_1 + 05^* 0)
\]

\[
1^* (0+1)^* 0 (C_1 + 05^* 0)
\]
Regular expressions in programming
One last expression....
Bit strings with odd number of 0s and 1s
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...)