Urbana · ZJU · CSP

- Canvas [Linked]
- Gradescope
- Piazza
- Website: ecealgo.com

Lectures
→ 0 ✓
→ 1 ~

Labs
→ 0 (x)
→ 1 (Tomorrow)

HWS
→
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

Instructor: Abhishek Kumar Umrawal
Jan 18, 2024

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

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

\[
L_{\text{AND}_n} = \left\{ 0|0, 1|1, 0\cdot0|0, 0\cdot1|0, 1\cdot0|0, 1\cdot1|1, \ldots \right\}
\]

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.

\[
L_{AND_n} = \left\{ 0|0, 1|1, 0 \cdot 0|0, 0 \cdot 1|0, 1 \cdot 0|0, 1 \cdot 1|1, \ldots, (0.)^n|0, (0.)^{n-1}|1|0, \ldots, (1.)^n|1|1 \ldots \right\}
\]

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

non recursively enumerable (undecidable)

recursively enumerable (decidable)

context sensitive

context free

regular
Chomsky Hierarchy

- Regular
- Context Free
- Context Sensitive
- Recursively Enumerable (Decidable)
- Non Recursively Enumerable (Undecidable)
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.
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.
- $\{\varepsilon\}$ is a regular language.
- $\{a\}$ is a regular language for each $a \in \Sigma$. Interpreting $a$ as 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 * 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?

\[
L = \{aba\} \quad \text{Regular?}
\]

By def. \( La = \{a\} \) is regular!

\( Lb = \{b\} \) is regular!

\[
L = La \cdot Lb \cdot La
\]

Hence \( L \) is regular!
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?
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 $L_{01} = 0^i 1^j$ for all $i, j \geq 0$ is regular:

$L_{01} = \{ \epsilon, 0, 01, 00, 011 \}$

$010 \notin L_{01}$

$L_0 = \{ 0^n \}$

$L_1 = \{ 1^n \}$

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

\[
L_3 = \{ \epsilon, 00, 0000, \ldots, 000, 000000, \ldots, 00000, \ldots \}
\]

\[
L_{i/2} := (L_0)^* \quad L_{i/5} := DIY \\
L_{i/3} := DIY
\]
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?

(DIY)
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 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>∅ regular</td>
<td>∅ denotes ∅</td>
</tr>
<tr>
<td>{ε} regular</td>
<td>ε denotes {ε}</td>
</tr>
<tr>
<td>{a} regular for a ∈ Σ</td>
<td>a denote {a}</td>
</tr>
<tr>
<td>R₁ ∪ R₂ regular if both are</td>
<td>r₁ + r₂ denotes R₁ ∪ R₂</td>
</tr>
<tr>
<td>R₁R₂ regular if both are</td>
<td>r₁ • r₂ denotes R₁R₂</td>
</tr>
<tr>
<td>R* is regular if R is</td>
<td>r* denote R*</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 $(1 + 0)$ denotes same language $\{0, 1\}$

\[
\begin{align*}
\gamma &= 0 + 1 & \gamma_1 &= 1 + 0 \\
L(\gamma) &= \{0, 1\} & L(\gamma_1) &= \{0, 1\}
\end{align*}
\]
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 $L(r_1) = L(r_2)$. 

**Other notation:** $r + s$, $r \cup s$, $r | s$ all denote union. $rs$ is sometimes written as $r \cdot s$. 
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.

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

- Other notation: $r + s$, $r \cup s$, $r \mid s$ all denote union. $rs$ is sometimes written as $r \cdot s$. 

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:** \( rst = (rs)t = r(st), \)

\( r + s + t = r + (s + t) = (r + s) + t. \)
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 $\ast$**. For convenience, define $r^+ = r r^*$. Hence if $L(r) = R$ then $L(r^+) = R^+$.

\[
\gamma^* s := (L(\gamma))^* (L(s))
\]
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:** $r^\ast s + t = ((r^\ast)s) + t$

Omit parenthesis by associativity of each operation.

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

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

**Superscript $\ast$.** For convenience, define $r^+ = rr^\ast$. 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?

\[ \Sigma = \{0, 1\} \]

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

1. All strings that end in 1011?

2. All strings except 11?

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

\[\varepsilon + 0 + 1 + 00 + 10 + 01 + \textcolor{red}{11} \uparrow + (0+1)(0+1)(0+1)^+\]
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?

\[
\begin{align*}
w_1 &= 0100101 \\
000 &\text{ is a subseq of } w_1! \ (\checkmark) \\
w_2 &= 010 \\
000 &\quad \text{---} \quad w_2 \ (\times) \\
\mathcal{E} + 01^* + 001^* + \ldots &\Rightarrow \\
\end{align*}
\]
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?
1. $(0 + 1)^*$:
Interpreting regular expressions

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

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

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

\[
L_{AND_N} = \begin{cases}
0 \cdot | 0, & 1 \cdot | 1, \\
0 \cdot 0 \cdot | 0, & 0 \cdot 1 \cdot | 0, & 1 \cdot 0 \cdot | 0, & 1 \cdot 1 \cdot | 1 \\
\vdots & \vdots & \vdots & \vdots \\
(0 \cdot)^n | 0, & (0 \cdot)^{n-1} | 0, & \ldots & (1 \cdot)^n | 1 \ldots
\end{cases}
\]

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

Consider the problem of a \textit{n}-input \texttt{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 \texttt{AND} of all the elements of \(x\). We know the language used to describe it is:

\[
L_{\text{AND}^N} = \left\{ 0 \cdot |0, \quad 1 \cdot |1, \\
0 \cdot 0 \cdot |0, \quad 0 \cdot 1 \cdot |0, \quad 1 \cdot 0 \cdot |0, \quad 1 \cdot 1 \cdot |1 \\
\vdots \quad \vdots \quad \vdots \\
(0 \cdot)^n |0, \quad (0 \cdot)^{n-1} |1 |0, \quad \ldots \quad (1 \cdot)^n |1 \ldots \right\}
\]

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

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

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

\[(00)+11\]∗[(01)+10]∗[(00)+11]+[(01)+10][(00)+11]\]∗[(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...)