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.

ECE-374-B: Lecture 1 - Regular Languages

Lecturer: Nickvash Kani

August 29, 2024

University of Illinois at Urbana-Champaign

Pre-lecture brain teaser

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

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

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.

$$L_{AND_N} = \left\{ \begin{array}{cccc} 0|0, & 1|1, & & \\ 0 \cdot 0|0, & 0 \cdot 1|0, & 1 \cdot 0|0, & 1 \cdot 1|1 \\ \vdots & \vdots & \vdots & \vdots \\ (0 \cdot)^n|0, & (0 \cdot)^{n-1}1|0, & \ldots & (1 \cdot)^n|1 \ldots \end{array} \right\} \tag{1}$$

Pre-lecture brain teaser

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

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

$$L_{AND_N} = \left\{ \begin{array}{cccc} 0|0, & 1|1, & & \\ 0 \cdot 0|0, & 0 \cdot 1|0, & 1 \cdot 0|0, & 1 \cdot 1|1 \\ \vdots & \vdots & \vdots & \vdots \\ (0\cdot)^n|0, & (0\cdot)^{n-1}1|0, & \dots & (1\cdot)^n|1 \dots \end{array} \right\} \tag{1}$$

This is an example of a regular language which we'll be discussing today.

Terminology Review

- A character(a, b, c, x) is a unit of information represented by a symbol: (letters, digits, whitespace)
- A alphabet(Σ) is a set of characters
- A string(w) is a sequence of characters
- A language(A, B, C, L) is a set of strings

Terminology Review

- A character(a, b, c, x) is a unit of information represented by a symbol: (letters, digits, whitespace)
- A alphabet(Σ) is a set of characters
- A string(w) is a sequence of characters
- A language(A, B, C, L) is a set of strings

Defining a language

How do we define a language? **Through grammars!**

What is a grammar?

Grammar (CFG) Definition

Definition
A CFG is a quadruple $G = (V, T, P, S)$

- V is a finite set of non-terminal (variable) symbols

$$G = \Big(\quad \text{Variables}, \quad \text{Terminals}, \quad \text{Productions}, \quad \text{Start var} \quad \Big)$$

Grammar (CFG) Definition

Definition
A CFG is a quadruple $G = (V, T, P, S)$

- V is a finite set of non-terminal (variable) symbols
- T is a finite set of terminal symbols (alphabet)

$$G = \left(\text{Variables}, \quad \text{Terminals}, \quad \text{Productions}, \quad \text{Start var} \right)$$

Grammar (CFG) Definition

Definition
A CFG is a quadruple $G = (V, T, P, S)$

- V is a finite set of non-terminal (variable) symbols
- T is a finite set of terminal symbols (alphabet)
- P is a finite set of productions, each of the form
  $A \to \alpha$
  where $A \in V$ and α is a string in $(V \cup T)^*$.
  Formally, $P \subset V \times (V \cup T)^*$.

$$G = \Big(\text{ Variables, } \quad \text{Terminals, } \quad \text{Productions, } \quad \text{Start var } \Big)$$

Grammar (CFG) Definition

Definition
A CFG is a quadruple $G = (V, T, P, S)$

- V is a finite set of non-terminal (variable) symbols
- T is a finite set of terminal symbols (alphabet)
- P is a finite set of productions, each of the form
  $A \to \alpha$
  where $A \in V$ and α is a string in $(V \cup T)^*$.
  Formally, $P \subset V \times (V \cup T)^*$.
- $S \in V$ is a start symbol

$$G = \Big(\quad \text{Variables}, \quad \text{Terminals}, \quad \text{Productions}, \quad \text{Start var} \quad \Big)$$

Example

L = all strings with 000 as a substring

- $V = \{S, A, B\}$
- $T = \{0, 1\}$
- $P = \left\{ \begin{array}{l} S \rightarrow 0S|1S|A \\ A \rightarrow 000B \\ B \rightarrow 0B|1B|\epsilon \end{array} \right\}$

  ($A \rightarrow B|C$ is abbreviation for $A \rightarrow B, A \rightarrow C$)

Example

L = all strings with 000 as a substring

- $V = \{S, A, B\}$
- $T = \{0, 1\}$
- $P = \left\{ \begin{array}{l} S \to 0S|1S|A \\ A \to 000B \\ B \to 0B|1B|\epsilon \end{array} \right\}$

  ($A \to B|C$ is abbreviation for $A \to B, A \to C$)

What strings can S generate like this?

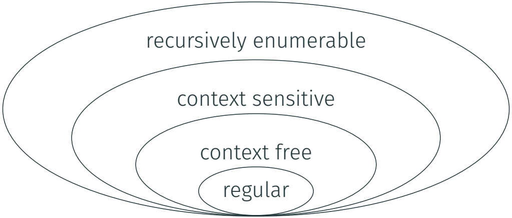
Example

- $V = \{S, A, B\}$
- $T = \{0, 1\}$
- $P = \left\{ \begin{array}{l} S \to 0S|1S|A \\ A \to 000B \\ B \to 0B|1B|\epsilon \end{array} \right\}$

  ($A \to B|C$ is abbreviation for $A \to B, A \to C$)

$$S \rightsquigarrow 1S \rightsquigarrow 10S \rightsquigarrow 10A \rightsquigarrow 10000B \rightsquigarrow 10000\varepsilon \rightsquigarrow 10000$$

Chomsky Hierarchy



Grammar	Languages	Production Rules	Automation	Examples
Type-0	Recursively enumerable	$\gamma \rightarrow \alpha$ (no constraints)	Turing machine	$L = \{\langle M, w \rangle \| M$ is a TM which halts on $w\}$
Type-1	Context-sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$	Linear bounded Non-deterministic Turing machine	$L = \{a^n b^n c^n \| n > 0\}$
Type-2	Context-free	$A \rightarrow \alpha$	Non-deterministic Push-down automata	$L = \{a^n b^n \| n > 0\}$
Type-3	Regular	$A \rightarrow aB$	Finite State Machine	$L = \{a^n \| n > 0\}$

Meaning of symbols: • a = terminal • A, B = variables • α, β, γ = string of $\{a \cup A\}^*$ • α, β = maybe empty —– γ = never empty

- Table borrowed from wikipedia: `https://en.wikipedia.org/wiki/Chomsky_hierarchy`

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 Σ 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.

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 $L_1 L_2$ is regular.
- If L is regular, then $L^* = \cup_{n \geq 0} L^n$ is regular.
  The $\cdot^*$ operator name is <u>Kleene star</u>.
- 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?

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?

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 Σ. Then the language $\cup_{i=1}^{\infty} L_i$ 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 Σ. Then the language $\cup_{i=1}^{\infty} L_i$ is not necessarily regular.

Note:Kleene star (repetition) is a **single** operation!

Regular Languages - Example

Example: The language $L_{01} = 0^i 1^j |$ for all $i, j \geq 0$ is regular:

Rapid-fire questions - regular languages

1. $L_1 = \left\{ 0^i \;\middle|\; i = 0, 1, \ldots, \infty \right\}$. The language L_1 is regular. T/F?

Rapid-fire questions - regular languages

1. $L_1 = \left\{ 0^i \;\middle|\; i = 0, 1, \ldots, \infty \right\}$. The language L_1 is regular. T/F?
2. $L_2 = \left\{ 0^{17i} \;\middle|\; i = 0, 1, \ldots, \infty \right\}$. The language L_2 is regular. T/F?

Rapid-fire questions - regular languages

1. $L_1 = \left\{ 0^i \;\middle|\; i = 0, 1, \ldots, \infty \right\}$. The language L_1 is regular. T/F?
2. $L_2 = \left\{ 0^{17i} \;\middle|\; i = 0, 1, \ldots, \infty \right\}$. The language L_2 is regular. T/F?
3. $L_3 = \left\{ 0^i \;\middle|\; i \text{ is divisible by } 2, 3, \text{or } 5 \right\}$. L_3 is regular. T/F?

Rapid-fire questions - regular languages

1. $L_1 = \left\{ 0^i \;\middle|\; i = 0, 1, \ldots, \infty \right\}$. The language L_1 is regular. T/F?
2. $L_2 = \left\{ 0^{17i} \;\middle|\; i = 0, 1, \ldots, \infty \right\}$. The language L_2 is regular. T/F?
3. $L_3 = \left\{ 0^i \;\middle|\; i \text{ is divisible by } 2, 3, \text{or } 5 \right\}$. 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 $\mathbf{r}$ over an alphabet Σ is one of the following:

Base cases:

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

Inductive cases: If $\mathbf{r}_1$ and $\mathbf{r}_2$ are regular expressions denoting languages R_1 and R_2 respectively then,

- $(\mathbf{r}_1 + \mathbf{r}_2)$ denotes the language $R_1 \cup R_2$
- $(\mathbf{r}_1 \bullet \mathbf{r}_2) = r_1 \bullet r_2 = (\mathbf{r}_1\mathbf{r}_2)$ denotes the language $R_1 R_2$
- $(\mathbf{r}_1)^*$ denotes the language R_1^*

Regular Languages vs Regular Expressions

Regular Languages	Regular Expressions
$\emptyset$ regular	$\emptyset$ denotes $\emptyset$
$\{\epsilon\}$ regular	ϵ denotes $\{\epsilon\}$
$\{a\}$ regular for $a \in \Sigma$	$\mathbf{a}$ denote $\{a\}$
$R_1 \cup R_2$ regular if both are	$\mathbf{r_1} + \mathbf{r_2}$ denotes $R_1 \cup R_2$
$R_1 R_2$ regular if both are	$\mathbf{r_1} \bullet \mathbf{r_2}$ denotes $R_1 R_2$
R^* is regular if R is	$\mathbf{r}^*$ denote R^*

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

Notation and Parenthesis

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

Notation and Parenthesis

- For a regular expression $\mathbf{r}$, $L(\mathbf{r})$ is the language denoted by $\mathbf{r}$. Multiple regular expressions can denote the same language!
  Example: $(0+1)$ and $(1+0)$ denotes same language $\{0,1\}$
- Two regular expressions $\mathbf{r}_1$ and $\mathbf{r}_2$ are equivalent if $L(\mathbf{r}_1) = L(\mathbf{r}_2)$.

Notation and Parenthesis

- For a regular expression $\mathbf{r}$, $L(\mathbf{r})$ is the language denoted by $\mathbf{r}$. Multiple regular expressions can denote the same language!
  Example: $(0+1)$ and $(1+0)$ denotes same language $\{0,1\}$
- Two regular expressions $\mathbf{r}_1$ and $\mathbf{r}_2$ are equivalent if $L(\mathbf{r}_1) = L(\mathbf{r}_2)$.
- Omit parenthesis by adopting precedence order: $*$, $\cdot$, $+$.
  Example: $r^*s + t = ((r^*)s) + t$

Notation and Parenthesis

- For a regular expression $\mathbf{r}$, $L(\mathbf{r})$ is the language denoted by $\mathbf{r}$. Multiple regular expressions can denote the same language!
  Example: $(0+1)$ and $(1+0)$ denotes same language $\{0,1\}$
- Two regular expressions $\mathbf{r}_1$ and $\mathbf{r}_2$ are equivalent if $L(\mathbf{r}_1) = L(\mathbf{r}_2)$.
- 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 $\mathbf{r}$, $L(\mathbf{r})$ is the language denoted by $\mathbf{r}$. Multiple regular expressions can denote the same language!
  Example: $(0 + 1)$ and $(1 + 0)$ denotes same language $\{0, 1\}$
- Two regular expressions $\mathbf{r}_1$ and $\mathbf{r}_2$ are equivalent if $L(\mathbf{r}_1) = L(\mathbf{r}_2)$.
- 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 $\mathbf{r}^+ = \mathbf{r}\mathbf{r}^*$. Hence if $L(\mathbf{r}) = R$ then $L(\mathbf{r}^+) = R^+$.

Notation and Parenthesis

- For a regular expression $\mathbf{r}$, $L(\mathbf{r})$ is the language denoted by $\mathbf{r}$. Multiple regular expressions can denote the same language!
  Example: $(0+1)$ and $(1+0)$ denotes same language $\{0,1\}$
- Two regular expressions $\mathbf{r}_1$ and $\mathbf{r}_2$ are equivalent if $L(\mathbf{r}_1) = L(\mathbf{r}_2)$.
- 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 $\mathbf{r}^+ = \mathbf{r}\mathbf{r}^*$. Hence if $L(\mathbf{r}) = R$ then $L(\mathbf{r}^+) = R^+$.
- Other notation: $r + s$, $r \cup s$, $r|s$ all denote union. rs is sometimes written as $r \bullet s$.

Some examples of regular expressions

Creating regular expressions

1. All strings that end in 1011?

Creating regular expressions

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

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?

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. $(0+1)^*$:

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. $(\epsilon+1)(01)^*(\epsilon+0)$:

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 knwo the language used to describe it is:

$$L_{AND_N} = \left\{ \begin{matrix} 0 \cdot |0, & 1 \cdot |1, & & \\ 0 \cdot 0 \cdot |0, & 0 \cdot 1 \cdot |0, & 1 \cdot 0 \cdot |0, & 1 \cdot 1 \cdot |1 \\ \vdots & \vdots & \vdots & \vdots \\ (0 \cdot)^n |0, & (0 \cdot)^{n-1} 1 |0, & \dots & (1 \cdot)^n |1 \dots \end{matrix} \right\}$$

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 knwo the language used to describe it is:

$$L_{AND_N} = \left\{ \begin{array}{cccc} 0 \cdot |0, & 1 \cdot |1, & & \\ 0 \cdot 0 \cdot |0, & 0 \cdot 1 \cdot |0, & 1 \cdot 0 \cdot |0, & 1 \cdot 1 \cdot |1 \\ \vdots & \vdots & \vdots & \vdots \\ (0 \cdot)^n|0, & (0 \cdot)^{n-1}1|0, & \ldots & (1 \cdot)^n|1 \ldots \end{array} \right\}$$

Formulate the regular expression which describes the above language:

$$\Sigma = \{0, 1, `\cdot', `|'\} \; r_{AND_N} = \underbrace{(\text{“}0\cdot\text{”} + \text{“}1\cdot\text{”})^* \, \text{“}0\cdot\text{”} \, (\text{“}0\cdot\text{”} + \text{“}1\cdot\text{”})^* \, \text{“}|0\text{”}}_{\text{all output 0 instances}} + \overbrace{(\text{“}1\cdot\text{”})^* \, \text{“}|1\text{”}}^{\text{all output 1 instances}}$$

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

$$(00+11)^*(01+10)\Big(00+11+(01+10)(00+11)^*(01+10)\Big)^*$$

Bit strings with odd number of 0s and 1s

The regular expression is

$$(00 + 11)^{*}(01 + 10)\Big(00 + 11 + (01 + 10)(00 + 11)^{*}(01 + 10)\Big)^{*}$$

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