Strings, Languages, and Regular Expressions

Strings

An *alphabet* set S is simply a nonempty set of things.
We will call these things symbols.
A *finite string of length* n over an alphabet S is a total function from $\{0, ..n\}$ to S.
Examples.

- Suppose $S = \{\text{‘a’}, \text{‘b’}, \text{‘c’}\}$. (A set of three characters.) Then the following function is a finite string of length 3 over S.

  $$(\{0, 1, 2\}, S, \{0 \mapsto \text{‘a’}, 1 \mapsto \text{‘b’}, 2 \mapsto \text{‘b’}, \})$$

  We'll also write this string as $[\text{‘a’}, \text{‘b’}, \text{‘b’}]$.
- Suppose $S = \{\text{‘a’}, \text{‘b’}, \text{‘c’}\}$. Then the following function is a finite string of length 0 over S.

  $$(\emptyset, S, \emptyset)$$

  We will also write this string (and any other string of length 0) as $[\,]$ or as ϵ or as “”.

When S is a set of characters, I'll use double quotes to indicate strings
Example: So $[\text{‘a’}, \text{‘b’}, \text{‘b’}] =$ “abb”.

For each $n \in \mathbb{N}$ the set of strings of length n over S is

$$S^n = \left(\{0, ..n\} \xrightarrow{\text{tot}} S \right)$$

If $s \in S^n$, we say that its *length* is n and write $\|s\| = n$.

Examples

- Suppose $S = \{0, 1\}$ then S^2 is
$$\{[0,0], [0,1], [1,0], [1,1]\}$$
- $\|\text{“aabb”}\| = 4$

The set of finite strings over S is

$$S^* = \bigcup n \in \mathbb{N} \cdot S^n$$

Example Suppose $S = \{\text{‘a’}, \text{‘b’}\}$ then S^* is

$$\{\epsilon, \text{“a”}, \text{“b”}, \text{“aa”}, \text{“ab”}, \text{“ba”}, \text{“bb”}, \text{“aaa”}, \cdots\}$$

Note that while the size of S^* is infinite (even for finite S), the length of each element of S is finite.

We can *concatenate* two strings s and t to get a string $s \hat{} t$. Such that

$$s \hat{} t \in S^{\|s\|+\|t\|} \text{ and thus}$$
$$\|s \hat{} t\| = \|s\| + \|t\| \text{. Furthermore}$$
$$(s \hat{} t)\,(i) = s(i), \text{ if } 0 \leq i < \|s\| \text{, and}$$
$$(s \hat{} t)\,(\|s\| + i) = t(i), \text{ if } 0 \leq i < \|t\|$$

- Note the ϵ is the identity element of catenation $s \hat{} \epsilon = s = \epsilon \hat{} s$, for all s.

Letter conventions for this section of the course

I'll use variables as follows

S an alphabet

$a, b, c \in S$

$s, t, w \in S^*$

$M, N \subseteq S^*$

x, y regular expressions over S

Q a set of states

$p, q, r \in Q$

$R, F \subseteq Q$

T a set of transitions

Languages

A *language* over S is any subset of S^*.

We can extend the operation of concatenation to languages.

If M and N are languages over S.

$$M \hat{} N = \{s \in M, t \in N \cdot s \hat{} t\}$$

For example if

$M = \{$“a”, “aa”, “ab”$\}$ and $N = \{\epsilon,$ “b”$\}$ then

$M \hat{} N = \{$“a”, “aa”, “ab”, “aab”, “abb”$\}$

For each $i \in \mathbb{N}$ we define M^i to be $M \hat{} M \hat{} \cdots \hat{} M$ where there are i Ms.

For example, if

$M = \{$“a”, “aa”, “ab”$\}$ then

$M^1 = \{$“a”, “aa”, “ab”$\}$

$M^2 = \{$“aa”, “aaa”, “aab”,“aaaa”, “aaab”,“aba”, “abaa”, “abab”$\}$

$M^3 = \{$“aaa”, “aaaa”, “aaab”,“aaaaa”, “aaaab”,“aaba”, “aabaa”, “aabab”, “aaaaaa”, “aaaaab”,“aaaba”, “aaabaa”, “aaabab”, “abaa”, “abaaa”, “abaab”, “abaaaa”, “abaaab”,“ababa”, “ababaa”, “ababab”$\}$

By convention, M^0 is $\{\epsilon\}$. So we can define M^i by

$$\begin{aligned} M^0 &= \{\epsilon\} \\ M^{i+1} &= M \hat{} M^i, \text{ for all } i \in \mathbb{N} \end{aligned}$$

$$\begin{aligned} M^0 &= \{\epsilon\} \\ M^{i+1} &= M \hat{} M^i, \text{ for all } i \in \mathbb{N} \end{aligned}$$

We can define the *Kleene closure* M^* of a set of strings as the set of all finite strings that can be generated from its members by catenation. So if $M = \{$“a”,“bb”$\}$ then

$$\begin{aligned} M^* = \{&\epsilon, \text{“a”}, \text{“bb”}, \text{“aa”}, \text{“abb”}, \text{“bba”}, \text{“bbbb”}, \\ &\text{“aaa”}, \text{“aabb”}, \text{“abba”}, \text{“abbbb”}, \\ &\text{“bbaa”}, \text{“bbabb”}, \text{“bbbba”}, \text{“bbbbbb”}, \cdots \} \end{aligned}$$

Formally we can define

$$M^* = \bigcup i \in \mathbb{N} \cdot M^i$$

To put it differently, a finite string s is a member of M^* iff there is a sequence of 0 or more strings

$$s_0, s_1, \cdots, s_n \in M$$

such that

$$s = s_0 \hat{} s_1 \hat{} \cdots \hat{} s_n$$

Regular languages

Given an alphabet S, we can make the following *simple regular languages:*

- $\emptyset$ — the empty language; contains no strings
- $\{\epsilon\}$ — the language that contains only the empty string
- For each $a \in S$, $\{[a]\}$ — the languages of single, length 1 strings.

Example: If $S = 0, 1\}$ we have the following 4 simple regular languages

$$\emptyset, \{\epsilon\}, \{[0]\}, \{[1]\}$$

Given that M and N are languages over S, consider three ways to make languages

- Union: $M \cup N$ — the language that contains all strings either in M or in N
- Concatenation: $M \hat{} N$ — the language of strings s that can be split into two parts $s = t \hat{} w$, where $t \in M$ and $w \in N$.
- Kleene closure: M^*— the language of strings s that can be split into some number (including 0) of parts, each of which is in M.

Examples: If S is $\{0, 1\}$:

- The following are simple regular languages
$$\emptyset, \{\epsilon\}, \{[0]\}, \{[1]\}$$
- We can make the following languages out of the simple regular languages using only union, concatentation, and Kleene closure:
  - $\{[0]\} \cup \{[1]\} \cup \{\epsilon\} = \{\epsilon, [0], [1]\}$
  - $\{[0]\} \mathbin{\hat{}} \{[1]\} = \{[0, 1]\}$
  - $\{[0]\}^* = \{\epsilon, [0], [0, 0], [0, 0, 0], \ldots\}$
  - $\{[0]\}^* \cup \{[1]\}^* = \{\epsilon, [0], [1], [0, 0], [1, 1], \ldots\}$
  - $\{[0]\}^* \mathbin{\hat{}} \{[1]\}^* = \{\ \epsilon$
    $[0], [1],$
    $[0, 0], [0, 1], [1, 1],$
    $[0, 0, 0], [0, 0, 1], [0, 1, 1], [1, 1, 1], \ldots\}$

A *regular language* over S is any language that be made from simple regular languages over S using a finite number of applications of the three operators $\cup$, $\hat{}$, and *.
Recap:

- Simple languages:
  - $\emptyset$ is a regular language
  - $\{\epsilon\}$ is a regular languages
  - For each $a \in S$, $\{[a]\}$ is a regular language.
- If M and N are regular languages, then so are
  - $M \cup N$, $M \mathbin{\hat{}} N$, and M^*.

The restriction to finite applications is important. Let $S = \{0, 1\}$. We can see that

- $\{[0]\}$, $\{[1, 0, 1]\}$, $\{[1, 1, 0, 1, 1]\}$ and so on are all regular languages.
- In general, for each $i \in \mathbb{N}$, $M_i = \{[1]\}^i \mathbin{\hat{}} \{[0]\} \mathbin{\hat{}} \{[1]\}^i$ is a regular language
- $\{[0]\} \cup \{[1, 0, 1]\} \cup \{[1, 1, 0, 1, 1]\}$ is a regular language.
- In general, for each $n \in \mathbb{N}$, $\bigcup_{i \in \{1,..n\}} M_i$ is a regular language.
- But we should not conclude that $\bigcup_{i \in \mathbb{N}} M_i = \{[0]\} \cup \{[1, 0, 1]\} \cup \{[1, 1, 0, 1, 1]\} \cup \cdots$ is regular. (And in fact it is not.)

Next we look at a language for describing regular languages.

Regular expressions

A *regular expression* over S is a string that describes a regular language over S.
The symbols of our regular expressions over S will consist of copies of the symbols of S and 7 special symbols.
The set of regular expressions over S is itself a language over $\underline{S} \cup \{\underline{\epsilon}, \underline{\emptyset}, \underline{|}, \underline{;}, \underline{*}, \underline{(}, \underline{)}\}$ where $\underline{S}$ is a set consisting of the symbols in S underlined.

Syntax:

- For each $a \in S$, $[\underline{a}]$ is a regular expression
- $[\underline{\epsilon}]$ is a regular expression.
- $[\underline{\emptyset}]$ is a regular expression.
- If x and y are regular expressions then so are
  - $[\underline{(}]\hat{}\, x \,\hat{}\, [\underline{|}]\hat{}\, y \,\hat{}\, [\underline{)}]$ (alternation)
  - $[\underline{(}]\hat{}\, x \,\hat{}\, [\underline{;}]\hat{}\, y \,\hat{}\, [\underline{)}]$ (concatenation)
  - $[\underline{(}]\hat{}\, x \,\hat{}\, [\underline{*}]\hat{}\, [\underline{)}]$ (repetition)

Some examples: For $S = \{0, 1\}$, the following are all regular expressions over S.

$$[\underline{0}]$$
$$[\underline{1}]$$
$$[\underline{\epsilon}]$$
$$[\underline{\emptyset}]$$
$$[\underline{(}, \underline{0}, \underline{\mid}, \underline{1}, \underline{)}]$$
$$[\underline{(}, \underline{0}, \underline{;}, \underline{1}, \underline{)}]$$
$$[\underline{(}, \underline{0}, \underline{*}, \underline{)}]$$
$$[\underline{(}, \underline{0}, \underline{;}, \underline{(}, \underline{1}, \underline{*}, \underline{)}, \underline{)}]$$
$$[\underline{(}, \underline{(}, \underline{0}, \underline{;}, \underline{(}, \underline{1}, \underline{*}, \underline{)}, \underline{)}, \underline{\mid}, \underline{(}, \underline{0}, \underline{;}, \underline{1}, \underline{)}, \underline{)}]$$

Underlining convention: An underlined sequence of symbols is a regular expression.

- We'll write the above examples, as follows

$$\underline{0}$$
$$\underline{1}$$
$$\underline{\epsilon}$$
$$\underline{\emptyset}$$
$$\underline{(0 \mid 1)}$$
$$\underline{(0; 1)}$$
$$\underline{(0^*)}$$
$$\underline{(0; 1^*)}$$
$$\underline{((0; (1^*)) \mid (0; 1))}$$

- I've underlined regular expressions to make it clear that that is what they are. For example $\underline{\epsilon}$ is a string of length 1 in the language of regular expressions, whereas ϵ is a string of length 0.

- This convention creates an ambiguity between regular expressions of length 1 and symbols. E.g., the expression $\underline{0}$ could mean the symbol $\underline{0}$ or the string (and regular expression) $[\underline{0}]$.It should be clear from context which is meant.

Semantics

We can define the "meaning" of regular expressions over an alphabet S by defining a function L from regular expressions to languages over S.

- $L(\underline{a}) = \{[a]\}$, for each $a \in S$.
- $L(\underline{\epsilon}) = \{\epsilon\}$
- $L(\underline{\emptyset}) = \emptyset$
- If x and y are regular expressions then
  - $L(\underline{(}x\underline{;}y\underline{)}) = L(x)\hat{}L(y)$
  - $L(\underline{(}x \underline{\mid} y\underline{)}) = L(x) \cup L(y)$
  - $L(\underline{(}x^*\underline{)}) = (L(x))^*$

If $s \in L(x)$ we say that x *describes* s or that x *recognizes* s.

- What language is $L\left(\underline{((0;(1^*)) \mid (0;1))}\right)$?

Two regular expression are *equivalent* if they describe the same language.
If x is a regular expression then $L(x)$ is a regular language.
Every regular language is described by a regular expression.

Parentheses

- Outermost parentheses can be omitted.
  - $\underline{((0; (1^*)) \mid (0; 1))}$ can be written $\underline{(0; (1^*)) \mid (0; 1)}$
- Other parentheses can be omitted with the understanding that * has higher precedence than ; and that ; has higher precedence than $\mid$. Thus:
  - $\underline{0; (1^*)}$ can be abbreviated by $\underline{0; 1^*}$
  - $\underline{(0; 1)^*}$ needs its parentheses
  - $\underline{(0; 1^*) \mid (0; 1)}$ can be abbreviated by $\underline{0; 1^* \mid 0; 1}$
  - $\underline{(0 \mid 1) ; (0^* \mid 1^*)}$ needs its parentheses.
- We can write $\underline{x; y; z}$ to mean $\underline{(x; y); z}$.
- We can write $\underline{x \mid y \mid z}$ to mean $\underline{(x \mid y) \mid z}$.
- Redundant parentheses can be added:
  - $\underline{(0 \mid (1^*))}$ can be written as $\underline{(((0) \mid ((1)^*)))}$.

(The operators $\mid$, ; and * are analogous to $+$, $\times$, and exponentiation both in precedence and in some algebraic properties.)

Examples

Money

Suppose that $S =$ {‘$’,‘0’,‘1’,‘2’,‘3’,‘4’,‘5’,‘6’,‘7’,‘8’,‘9’,‘,’,‘.’,‘-’}
We want to describe strings representing amounts of money such as

“$0.05”

“$-12,345.67”

We can describe a set of strings representing amounts of money as follows. Let x represent the regular expression

$$\underline{(\text{‘0’} \mid \text{‘1’} \mid \text{‘2’} \mid \text{‘3’} \mid \text{‘4’} \mid \text{‘5’} \mid \text{‘6’} \mid \text{‘7’} \mid \text{‘8’} \mid \text{‘9’})}$$

The regular expression

$$y = \underline{\text{‘\$’}}; \underline{(\text{‘-’} \mid \epsilon)}; \underline{(x;x;x \mid x;x;x \mid x)}; \underline{(\text{‘,’};x;x;x)^*}; \underline{\text{‘.’};x;x}$$

is such that “$0.05” $\in L(y)$ and “$-12,345.67” $\in L(y)$, whereas “$12345.67” $\notin L(y)$ and “$12.6” $\notin L(y)$, etc.

More conventions and abbreviations

We leave off the underline when it is clear we are talking about a regular expression

R.E.	Abbreviates	**Describes …**
$x\,y$	$x;y$	catenation
s	$s(0)s(1)\cdots s(\|\|s\|\|-1)$	$\{s\}$
x^+	xx^*	any catenation of 1 or more strings, each described by x.
x^n	$\underbrace{xx\ldots x}_{n \text{ times}}$	any catenation of n, each described by x.
x_m^n	$x^m \mid x^{m+1} \mid \cdots \mid x^n$	any catenation of from m to n strings, each described by x.
$x^?$	$\epsilon \mid x$	a string described by x or ϵ.
.	$a \mid b \mid \cdots$	any string of length one.
$[a-b]$		any string of length one with a character alphabetically between a and b inclusive.

With these abbreviations

$$\text{`\$'};(\text{`-'} \mid \epsilon);(x;x;x \mid x;x \mid x);(\text{`,'};x;x;x)^*;\text{`.'};x;x \quad ,$$

where x is $(\text{`0'} \mid \text{`1'} \mid \text{`2'} \mid \text{`3'} \mid \text{`4'} \mid \text{`5'} \mid \text{`6'} \mid \text{`7'} \mid \text{`8'} \mid \text{`9'})$,
can be written as

$$\text{`\$'}\ \text{`-'}^?\ [\text{`0'}-\text{`9'}]_1^3\ (\text{`,'}\ [\text{`0'}-\text{`9'}]^3)^*\ \text{`.'}\ [\text{`0'}-\text{`9'}]^2$$

The abbreviations are useful in application of regular expressions.

However, any abbreviated regular expression can be rewritten to one that just uses our 6 ways of making a regular language. When dealing with the theory of regular expressions, we can safely ignore the abbreviations.

Examples

Identifiers

Suppose that $S = \{\text{‘\_’},\text{‘0’},\text{‘1’},\ldots,\text{‘9’},\text{‘a’},\text{‘b’},\ldots,\text{‘z’},\text{‘A’},\text{‘B’},\ldots,\text{‘Z’}\}$ then let A be the regular expression

$$\underline{([\text{‘a’} - \text{‘z’}] \mid [\text{‘A’} - \text{‘Z’}] \mid \text{‘\_’})([\text{‘a’} - \text{‘z’}] \mid [\text{‘A’} - \text{‘Z’}] \mid [\text{‘0’} - \text{‘9’}] \mid \text{‘\_’})^*}$$

describes C++ identifiers.

Parity

Let $S = \{0, 1\}$. Define a regular expression that matches only strings with an even number of 1s.

$$\underline{(\text{‘0’}^* \text{ ‘1’ ‘0’}^* \text{ ‘1’})^* \text{ ‘0’}^*}$$

String search

Let $S = \{\text{‘\_’},\text{‘0’},\text{‘1’},\ldots,\text{‘9’},\text{‘a’},\text{‘b’},\ldots,\text{‘z’},\text{‘A’},\text{‘B’},\ldots,\text{‘Z’}\}$. Define an r.e. that matches all strings that contain the substring “engi”.

Now the regular expression is

$$\underline{.^*\text{“engi”}.^*}$$

More string searches

Does a document contain any of the strings “`woman`”, “`women`”, “`man`”, “`men`” ?

$$\underline{.^* (\text{“woman”} \mid \text{“women”} \mid \text{“man”} \mid \text{“men”}) .^*}$$

or, equivalently,

$$\underline{.^*\text{“wo”}^?\text{‘m’} (\text{‘a’} \mid \text{‘e’}) \text{‘n’}.^*}$$

Does a document contain “John” followed by “Smith”

$.^*$ (“John”) $.^*$ (“Smith”) $.^*$

Challenges

- A C comment is a string that follows the following rules. C comments have at least 4 characters. The first two are “/*” and the last two are “*/”. The sequence “*/” does not occur in between the first two and last two characters. Examples include “/**/”, “/*/*/”, and “/*o*o/*/” . Counter examples include “o/**/”, “/**/o”, and “/**/*/”. Suppose x is a regular expression whose language is the set of all strings of length 1 other than “/” and “*”. Devise a regular expression whose language is the set of all C comments.
- Let $s^\smile$ be the string s in reverse and $M^\smile = \{s \in M \cdot s^\smile\}$. Write an algorithm that, given a regular expression x, computes a regular expression y such that $L(y) = L(x)^\smile$.
- Represent an addition such as $2 + 3 = 5$ by “235” and $12 + 35 = 47$ by “134257”. Pad the operands with 0s so that both operands and the result are the same number of digits; e.g., represent $23 + 777 = 800$ by “078270370”. Thus strings in the language are always of lengths that are multiples of 3.Create a regular expression that represents correct additions in binary. Hint: it might be easier to create a regular expression for the reversed language. Examples

  $*\ 11 + 101 = 1000$ is represented by a string

$001\,010\,100\,110$ which is in the language.

* $11 + 11 = 110$ is represented by a string $001\,111\,110$ which is in the language
* $1 + 1 \neq 1$ and so 111 is not in the language.
* 00000 is not in the language, as its length is not a multiple of 3.

Regular languages (again)

A language M is a *regular language* exactly if there exists a regular expression x such that $M = L(x)$.

So far we have seen that some languages are regular. Perhaps not surprisingly, there are also languages that are not regular. In the next section of the course we will see that a language is regular exactly if it can be recognized by a program that has a fixed amount of memory.

Consider the following language

$$\begin{aligned} M &= \bigcup i \in \mathbb{N} \cdot \{\text{“a”}\}^i \mathbin{^\frown} \{\text{“b”}\}^i \\ &= \{\epsilon, \text{“ab”}, \text{“aabb”}, \text{“aaabbb”}, \cdots\} \end{aligned}$$

A program to recognize this language would somehow need to count the number of a’s and b’s. This would take an amount of memory that depends on the size of the input; i.e. it is not fixed. So this language is not regular. The preceding argument is not a formal proof; after learning a bit more, we will be in a position to be able to formally prove that some languages are not regular.