Lecture 5

Ben Rosenberg

June 14, 2021

Note: In these notes, the symbol $\circ$ is used where $\cdot$ may have been used in lecture. They mean the same thing (concatenation).

Regular Languages and Regular Expressions

Regular operations

Definition: **Regular operations** are applied to sets (languages). There are three, which follow:

(Note: $\Sigma = \{\texttt{a,b,c}\}$ for all examples below.)

Union

Union: The union operation is the usual one as applied to sets.

Example: Consider $L_1 = \{\texttt{a}, \texttt{aa}, \texttt{b}, \texttt{cbca}, \texttt{aba}, \texttt{ab}, \texttt{c}\}$ and $L_2 = \{\lambda, \texttt{a, b, cccba, cc}\}$. Then, the union of L_1 and L_2 is defined as $L_1 \cup L_2 = \{x | x \in L_1 \vee x \in L_2\}$. We say that $b \in L_1 \cup L_2$, because b is in either L_1 or L_2 (the fact that it is in both is not relevant).

Union is commutative: that is, $L_1 \cup L_2 = L_2 \cup L_1$ for all languages L_1 and L_2.

Concatenation

Concatenation: The concatenation $L_1 \circ L_2 = \{w | (\exists x \in L_1)(\exists y \in L_2)(w = xy)\}$. It is the set of all strings that can be obtained by "gluing together" one element of the first language with one element of the second language.

Example: `ab` is an element of $L_1 \circ L_2$, because $a \in L_1$ and $b \in L_2$. Furthermore, $ab \in L_1$ and $\lambda \in L_2$. Either of these "splits" suffices to show that `ab` $\in L_1 \circ L_2$ – the quantifier is existential, meaning that we are able to choose whichever split is most useful to us.

Note that concatenation is *not* commutative. Consider the string `ca`, which is in $L_1 \circ L_2$. This string is not in $L_2 \circ L_1$ as none of the three possible splits of `ca` can be written as the concatenation of a string from L_2 and a string from L_1.

Assuming we have an algorithm to answer $x \in L_1$, or $x \in L_2$, then we have an algorithm to answer whether $x \in L_1 L_2$ or $x \in L_1 \circ L_2$. We simply take x and try all possible "splits". For each split, we take the left side and check whether L_1 contains it, and take the right side, and check whether L_2 contains it. If we find a split for which the left side is in L_1 and the right side is in L_2, then we have shown that $x \in L_1 \circ L_2$.

Aside, before proceeding to the last operation:

Cardinalities The cardinality of $|L_1 \cup L_2| \leq |L_1| + |L_2|$. For the above example, we have $|L_1 \cup L_2| = 7 + 5 - 2 = 10$ because 2 of the elements are duplicates.

The union of L_1 and L_2 can be empty if both L_1 and L_2 are empty.

The cardinality of $|L_1 \circ L_2| \leq |L_1| \cdot |L_2|$, as there may be duplicate strings that can be created from the concatenation. `ab`, for example, belongs to $L_1 \circ L_2$ in two different ways, because `a` is in L_1 and `b` is in L_2, and `ab` is in L_1 and λ is in L_2.

If either L_1 or L_2 is empty, then their concatenation $L_1 \circ L_2$ is empty – you can never find a string that is in a given language if it is empty, meaning that there are no valid concatenated strings.

Onto the third operation.

Kleene star

Notation: For any set L: $L^0 = \{\lambda\}$. Recursively, $L^{n+1} = L^n \circ L$.

For example, $L^1 = L \circ L^0 = L \circ \{\lambda\} = L$. Similarly, $L^2 = L \circ L = LL$.

Definition: The **Kleene star** of a language L, denoted L^*, is equal to:

$$L^* = \bigcup_{x \geq 0} L^k = \boxed{L^0 \cup L^1 \cup L^2 \cup \cdots}$$

This is, in effect, the set of all strings that can be obtained by gluing together zero or more strings of the original language.

Another example: $\texttt{aaa} \in L_1^*$. The split $\texttt{a}|\texttt{aa} \in L_1^2$ works, as do $\texttt{a}|\texttt{a}|\texttt{a} \in L_1^3$ and $\texttt{aa}|\texttt{a} \in L_1^2$.

In fact, there are no strings (from Σ defined above) that are not contained in L_1^* because each of the elements of Σ are contained in L_1^*.

We define Σ^* as the set of all strings over Σ. Since $\Sigma \subseteq L_1$, $\Sigma^* \subseteq L_1^*$.

The Kleene star of any language – even $\emptyset$, the empty language – always contains at least one element: namely, λ. So, $(\quad)^* \neq \emptyset$ in all cases.

There are two cases in which L^* is finite. The first is that in which $L = \emptyset$, which gives $\emptyset^* = \lambda$. The second is $L = \{\lambda\}$, which also gives $L^* = \{\lambda\}$.

If L^* is not finite, then its cardinality is equal to $\aleph_0$. (See Gödel injection.)

Regular expressions

Definition: Given an alphabet $\Sigma = \{\texttt{a},\texttt{b},\texttt{c}\}$, the **class of regular languages** over Σ contains exactly those languages which are obtained by finitely many applications of regular operations to the sets:

$$\emptyset, \{\lambda\}, \{\texttt{a}\}, \{\texttt{b}\}, \{\texttt{c}\}$$

Definition: A **regular set** is any set which can be obtained by starting from the above, and applying any of the regular operations.

Definition: A **regular expression** over an alphabet $\{\Sigma = \{\texttt{a},\texttt{b},\texttt{c}\}\}$ is a string over the alphabet:

$$\{\texttt{a},\ \texttt{b},\ \texttt{c},\ (,\ ),\ \cup,\ \circ,\ *,\ \emptyset,\ \lambda\}$$

This is the *programmer's alphabet* (our "keyboard"), which represents a regular language. Note that these are all *symbols* and do not actually mean anything until they are used to write regular expressions.

Observe that the alphabet "keyboard" used to define regular expressions contains Σ. It also contains the seven symbols that are not contained in Σ and are used to represent notation.

Examples of translation from language to regular expression:

language	regular expression
$\emptyset$	$\emptyset$
$\{\lambda\}$	λ
$\{\mathtt{a}\}$	$\mathtt{a}$
recursively, if:	
$L_1 \mapsto$	$\mapsto e_1$
$L_2 \mapsto$	$\mapsto e_2$
then,	
$L_1 \cup L_2$	$(e_1) \cup (e_2)$
$L_1 \circ L_2$	$(e_1) \circ (e_2)$
L_1^*	$(e_1)^*$

Order of operations:

1. $*$
2. $\circ$
3. $\cup$

Examples ($\iff$ is used to denote going between regex and languages):

$$\mathtt{a} \cup \mathtt{bc}^* = (\mathtt{a}) \cup \Big((\mathtt{b}) \circ ((\mathtt{c})^*)\Big)$$

$$\mathtt{a} + \mathtt{bc}^2 = (\mathtt{a}) \cup \Big((\mathtt{b}) \circ ((\mathtt{c})^2)\Big)$$

$$\emptyset \circ \lambda \iff \emptyset$$

$$\mathtt{a} \iff \{\mathtt{a}\}$$

$$\mathtt{a} \cup \mathtt{b} \iff \{\mathtt{a}, \mathtt{b}\}$$

$$\mathtt{ab} \iff \{\mathtt{ab}\}$$

$$(\mathtt{a} \cup \mathtt{b})(\mathtt{a} \cup \mathtt{b}) = (\mathtt{a} \cup \mathtt{b}) \circ (\mathtt{a} \cup \mathtt{b}) \iff \{\mathtt{aa}, \mathtt{ab}, \mathtt{ba}, \mathtt{bb}\}$$

$$\mathtt{a}^* \iff \{\lambda, \mathtt{a}, \mathtt{aa}, \mathtt{aaa}, \dots\}$$

The assignment in class is always the following:

> Construct a regular expression that defines the set of exactly those strings over Σ that satisfy the following property:
>
> consists of $\mathtt{a}$'s and $\mathtt{b}$'s $\implies (\mathtt{a} \cup \mathtt{b})^*$

Aside: "A contains B" means that B is a subset of or element of A. "A consists of B" means that all of A are defined as being B.

More examples

all possible strings

$\implies (\mathtt{a} \cup \mathtt{b} \cup c)^*$

strings that have length equal to 3 (should be 27 of them)

$\implies (\mathtt{a} \cup \mathtt{b} \cup \mathtt{c}) \circ (\mathtt{a} \cup \mathtt{b} \cup \mathtt{c}) \circ (\mathtt{a} \cup \mathtt{b} \cup \mathtt{c})$

length ≤ 3

$\implies (\mathtt{a} \cup \mathtt{b} \cup \mathtt{c} \cup \lambda) \circ (\mathtt{a} \cup \mathtt{b} \cup \mathtt{c} \cup \lambda) \circ (\mathtt{a} \cup \mathtt{b} \cup \mathtt{c} \cup \lambda)$

consist of even # of $\mathtt{a}$'s

$\implies (\mathtt{aa})^*$

Note that $(\mathtt{aa})^*$ means an even number of $\mathtt{a}$'s, while $\mathtt{aa*}$ is any number of $\mathtt{a}$'s greater than 1.

even length

$\implies \Big((\mathtt{a} \cup \mathtt{b} \cup \mathtt{c}) \circ (\mathtt{a} \cup \mathtt{b} \cup \mathtt{c})\Big)^*$

odd length

$\implies \Big((\mathtt{a} \cup \mathtt{b} \cup \mathtt{c}) \circ (\mathtt{a} \cup \mathtt{b} \cup \mathtt{c})\Big)^* \circ (\mathtt{a} \cup \mathtt{b} \cup c)$

contains odd number of $\mathtt{a}$'s (contains means it could also have $\mathtt{b}$'s and $\mathtt{c}$'s)

$\implies (\mathtt{a}(\mathtt{b} \cup \mathtt{c})^*\mathtt{a} \cup \mathtt{b} \cup \mathtt{c})^*)\mathtt{a}(\mathtt{b} \cup \mathtt{c})^*$

contains even number of $\mathtt{a}$'s (contains means it could also have $\mathtt{b}$'s and $\mathtt{c}$'s)

$\implies (\mathtt{b} \cup \mathtt{c})^*\mathtt{a}(\mathtt{b} \cup \mathtt{c})^*\mathtt{a}(\mathtt{b} \cup \mathtt{c})^*)^*(\mathtt{b} \cup \mathtt{c})^*$

contain substring $\mathtt{abcb}$

$\implies (\mathtt{a} \cup \mathtt{b} \cup \mathtt{c})^*\mathtt{abcb}(\mathtt{a} \cup \mathtt{b} \cup \mathtt{c})^*$

contain as a substring either $\mathtt{abb}$ or $\mathtt{bca}$

$\implies (\mathtt{a} \cup \mathtt{b} \cup \mathtt{c})^*(\mathtt{abb} \cup \mathtt{bca})(\mathtt{a} \cup \mathtt{b} \cup \mathtt{c})^*$

contain as a substring both $\mathtt{abb}$ and $\mathtt{bca}$

$\implies (\mathtt{a}\cup\mathtt{b}\cup\mathtt{c})^*\mathtt{abb}(\mathtt{a}\cup\mathtt{b}\cup\mathtt{c})^*\mathtt{bca}(\mathtt{a}\cup\mathtt{b}\cup\mathtt{c})^* \bigcup(\mathtt{a}\cup\mathtt{b}\cup\mathtt{c})^*\mathtt{bca}(\mathtt{a}\cup\mathtt{b}\cup\mathtt{c})^*\mathtt{abb}(\mathtt{a}\cup \mathtt{b} \cup \mathtt{c})^* \bigcup(\mathtt{a} \cup \mathtt{b} \cup \mathtt{c})^*\mathtt{abbca}(\mathtt{a} \cup \mathtt{b} \cup \mathtt{c})^* \bigcup(\mathtt{a} \cup \mathtt{b} \cup \mathtt{c})^*\mathtt{bcabb}(\mathtt{a} \cup \mathtt{b} \cup \mathtt{c})^*$

evidently, intersection is a challenge.

begin with $\mathtt{bca}$

$\implies \mathtt{bca}(\mathtt{a} \cup \mathtt{b} \cup \mathtt{c})^*$

do not begin with $\mathtt{bca}$

$\implies \ldots ?$

We want, here, to find a means of a "complement". This one is the complement of the previous answer. We could write out the 26 correct 3-letter strings that are not $\mathtt{bca}$, and then add $(\mathtt{a} \cup \mathtt{b} \cup \mathtt{c})^*$ to the end, except that the length does not have to be 3 or greater.

We know that all strings of length 2 or less are good. Given the assumption that all strings of length 2 or less are good, then we can actually do it without enumerating all 27 of them.

We have:

$$(\mathtt{a} \cup \mathtt{b} \cup \mathtt{c} \cup \lambda)^*(\mathtt{a} \cup \mathtt{b} \cup \mathtt{c} \cup \lambda)^*$$
$$\bigcup$$
$$(\mathtt{a} \cup \mathtt{c})(\mathtt{a} \cup \mathtt{b} \cup \mathtt{c})^*$$
$$\bigcup$$
$$\mathtt{b}(\mathtt{a} \cup \mathtt{b})(\mathtt{a} \cup \mathtt{b} \cup \mathtt{c})^*$$
$$\bigcup$$
$$\mathtt{bc}(\mathtt{b} \cup \mathtt{c})(\mathtt{a} \cup \mathtt{b} \cup \mathtt{c})^*$$

Theorem (to be proven later): If there is a regular expression for some language, then there is a regular expression for its complement.

Aside: Good book: “Regular Algebra and Finite Machines”, by John Conway