# **Section 11.1: Regular Languages**

What is a language? A set of strings

What is an alphabet? A finite set of symbols (discluding lambda)

Given a language and a string, the simplest computer will say:

Yes, this string is in the language

No, this string is not in the language

Could you use this for real? Yes, for example, recognizing integers.

The simple computer will recognize regular languages; there’s an infinite number.

Remember, natural numbers include 0 and the rest of the positive numbers.

Examples of Regular Languages: {1, 2, 3, 4}, the set of all words in the English language, . To find languages (in the form of the set), just think of strings of length 0, then 1, then 2.

**Not All Languages are Regular**

Examples of NON-Regular Languages:

* The set of all palindromes over the English alphabet (same letters forwards and backwards, ex. daacaad) If you take the set of every single of these, it is NOT regular.
* The number of a’s and b’s are related, so there are always twice as many b’s as there are a’s, whereas is the opposite (the number of a’s and b’s are not related).
* The set of all integers in which the number of odd digits is the same as the number of even digits (ex. 111222, 23, 232352).

**What makes a language regular (over an alphabet A)?**

First 3 rules (base cases):

1. ∅ is a regular language over A
2. {^} is a regular language over A (the SET containing lambda, NOT lambda by itself)
3. {a} is a regular language over A for all aA (for any symbol of your alphabet, the single set that JUST contains that symbol IS a regular language)

Ex. Suppose F = {r, o, w, a, n}. List some regular languages over F.

Firstly, ask yourself, is this a finite set of symbols (the definition of an alphabet)? You can tell that it is thanks to the separation the commas provide. Based off of the three rules so far, the regular languages for alphabet F include ∅, {^}, {r}, {o}, {w}, {a}, {n}.

If the question were, what is the set of some regular languages over F?, the answer would be {∅, {^}, {r}, {o}, {w}, {a}, {n}}.

**Regular Languages: Full List of Rules over Alphabet A**

1. Base cases:
   1. ∅ is a regular language over A
   2. {^} is a regular language over A
   3. {a} is a regular language over A for all a ∈ A
2. Inductive step: If L and M are regular languages over A, then
   1. L ∪ M is a regular language over A

For example, suppose we have αβ = {a, b, c}, L = {a}, and M = {b}. The union of these is {a, b}. They do happen to be symbols since they’re part of an alphabet, but here, those symbols are strings since this is a language.

* 1. ML is a regular language over A

For example, using the same alphabet from the last, the concatenation of {b} and {a} (the product of two languages is the set of taking one string from the first language and another from the second) is {ba}, creating a single string in the language for this.

* 1. L\* is a regular language over A

This tells you that you can have languages that are infinitely long. This means that {a}\* = {^, a, aa, aaa, …} (the set) is a regular language.

Ex. Suppose αβ A = {a, b}. Based on the base case rules, what languages are regular over A? ∅, {^}, {a}, {b}

Ex. Take the same alphabet from the previous example Based on the inductive step rules, what languages are regular over A? M\* = {^, b, bb, bbb, …}, LM = {ab}, LM\* = {^, ab, abab, ababab, …}, and so on.

Ex. How big is the set of regular languages over A (same alphabet)? (Note: You’ve got a set of languages now, or a set of sets.) There are an infinite number of regular languages; this is because every time you make something with the inductive step rules, you can continue to concatenate, union, and star them, and they go on for practically forever.

**Question:** Is {^, b, bb, bn} a regular language over the αβ {a, b, c}?

Yes! This is because the base case rules give us the regular language {b}. You can say that {b} is L (since L is a regular language over A). The set is representative of L\*, in which there is taking of 0, 1, 2, etc. b’s.

**Question:** Let A be an αβ, A = {a, b, c}. Is {a, bc} a language? Is {a, bc} regular over A? Prove your answers.

Yes. {a, bc} is a language because it is a set of strings. Strings are a finite sequence of zero or more symbols, and both elements of the set are strings.

Yes. To prove this, build your reasoning up by parts.

1. {a} is a regular language over αβ A due to the base case rule
2. {b} is a regular language over αβ A due to the base case rule
3. {c} is a regular language over αβ A due to the base case rule
4. {bc} is a regular language over αβ A due to the concatenation rule of lines 2, 3
5. {a, bc}is a regular language over αβ A due to the union rule of lines 1, 4

**Ways to Specify Regular Languages**

A regular language is a language, so it’s a set. One way to specify it is using set notation, and another slightly easier way is to use a regular expression.

* All regular languages can be represented by a regular expression
* All regular expressions represent regular language

Some examples

The regular expressions are in order (for the third one, a and b are first just like in the language). Just like in math, the plus signs are associative. The only operators used are +, •, and \*.

| **Language** | **Equivalent Regular Expression** |
| --- | --- |
| {a} | a |
| {a, b} | a + b |
| {a, b, c} | a + b + c == (a + b) + c |
| {a} ∪ {b, c} | a + (b + c) == a + b + c |
| {a, b}{b, c} == {ab, ac, bb, bc} | (a + b) • (b + c) == (a + b) (b + c)  == ab + ac + bb + bc  *Eventually you won’t write the dot, but remember that it’s there when it’s not.* |
| {rowan, rules} | rowan + rules  (r • o • w • a • n) + (r • u • l • e • s)  r • o • w • a • n + r • u • l • e • s |
| {ab}\* == {^, ab, abab, ababab, …} | (ab)\*  *Remember this is different as ab\*. Think of the asterisk as an exponent.* |
| {a}{b}\* == {a, ab, abb, abbb, …} | ab\* |

Notation

L(Q) means “the language of the regular expression Q”. The first example reads, “the language of the regular expression (a + b + c) is {a, b, c}.

L(a + b + c) = {a, b, c}

L(rowan + rocks) = {rowan, rocks}

L((a + b) • (b + c)) = {a, b}{b, c}

L(∅) = ∅ (a set *or* a regular expression) = {} (a set)

NOTE: rowan + rocks ≠ {rowan, rocks} because a set and a regular language are two different things; you just have a way to convert between them. The language OF a regular expression equals the set.

**Regular Expressions: Full List of Rules over Alphabet A**

1. Base cases:
   1. ∅ is a regular expression over A
   2. ^ is a regular expression over A
   3. a is a regular expression over A for all a ∈ A
2. Inductive step: If R and S are regular expressions over A, then
   1. (R) is a regular expression over A
   2. R + S is a regular expression over A
   3. R • S is a regular expression over A
   4. R\* is a regular expression over A

**Precedence Rules (highest to lowest):** (), \*, •, +

**Problem:** Add parentheses and dots to the regular expression a + ba\* to show the order of evaluation. (a + (b • (a\*)))

Firstly, add the • operator in between the b and a\* since they’re being concatenated together. Now, it is time to add parentheses. First, add them around the a\* since in the precedence rules, \* comes first. Then, do it around the b • a (• is next), and then eventually around the entire regular expression.

**How Regular Expressions Represent Regular Languages**

If you have a regular expression E, you’ll *associate* a regular language with it. You call that “the language of E”, or notation-wise, L(E).

**Procedure: Finding the Regular Language Associated with a Regular Expression**

Let A be an alphabet, and let R and S be any regular expression over A. Then,

L(∅ ) = ∅

L(^) = {^}

L(a) = {a} (for each a ∈ A)

L(R + S) = L(R) ∪ L(S)

L(R • S) = L(R)L(S)

L(R)\* = (L(R))\*

**Problem from the book: Page 747, #1:** Find a language (i.e. a set of strings) to describe each of the following regular expressions: a + bc, ab\* + c, a\*bc\* + ac.

{a} ∪ {bc} == {a, bc} Remember the + is an equivalent to union in this.

{a}{b}\* ∪ {c} == {a, ab, abb, abbb, …} ∪ {c} Remember the \* takes the most precedence. Also remember the ab indicates concatenation, so they must be next to each other. There are two types of strings in this language. First is the c, and the second is the a followed by 0 or more b’s.

{a}\*{b}{c}\* ∪ {ac} == {b, ab, bc, abc, aab, bcc, …} ∪ {ac} The \* takes precedence so put them as individual sets concatenated together. Use the same reasonings as the previous two.

**Problem from the book: Page 747, #2:** Find a regular expression to describe each of the following languages: {aa, ab, ac}, {a, aaa, aaaaa, a2*n* + 1 | n ∈ N}, {^, a, b, c, aa, bb, cc, a*n*, b*n*, c*n* | n ∈ N}, {a2*k* | k ∈ N} ∪ {b2*k* + 1 | k ∈ N}

aa + ab + ac == a(a + b + c) Remember the language for this one are just sets of strings unioned together, essentially, thus creating that first option. The second is where you factor out that common factor.

a(aa)\* You definitely have 1 a in each string of the language. Then, you have 0 or more copies of aa after that. This is because (aa)\* is basically aa, aaaa, etc., and since the number of a’s is even here (due to the double a’s), you can add another a to make it odd.

a\* + b\* + c\* You could try plugging in sample values of *n* into each one continued and separating each by element: {^, a, aa, aaa, …} ∪ {^, b, bb, bbb, …} ∪ {^, c, cc, ccc, …}. This can be rewritten as {a}\* ∪ {b}\* ∪ {c}\*. Now you can see clearly where the answer derives from.

(aa)\* + b(bb)\* The last one is similar to the second language. 2*k* makes sure that the number of a’s or b’s are even numbers, so if you write out both sets, you can see it as {^, aa, aaaa, …} ∪ {^, b, bbb, …}, which can be represented as the answer given. In this set, you can use a q and r instead of k since the two sets are separate.

**Problem from the book: Page 747, #4:** Find a regular expression for each of the following languages over the ɑβ {a, b}: Strings whose length is a multiple of 3, strings with an odd number of a’s

(aaa + aab + aba + abb + baa + bab + bba + bbb)\* You can come up with this by thinking of every single combination of elements of the alphabet into length 3 strings; adding the star at the outside of the parentheses shows that there can be 0 or more of each. For example, aaaabbbba is an example of a string whose length is a multiple of 3.

((a + b)(a + b)(a + b))\* You have to pick an element of a or b, then concatenate that with a or b, then concatenate that with a or b, then have 0 or more of that string. The inner piece shows that there can be any combination of the three letters and have 0 or more copies of that, basically.

b\*ab\*(b\*ab\*ab\*)\* How this works is: Start out with any number of b’s. After that, put an a. After that, put as many b’s as you want. You have to have at least 1 a in the string to make it odd. The first section (the one outside the parentheses) represents all the strings with 1 a in it.

The second piece says to take as many copies of that piece inside as you want; start with however many b’s, stuff an a in, as many b’s, stuff an a in, and as many b’s. The two a’s on the inside show that it needs to be even in that area + the 1 outside is an odd number of a’s, guaranteed. This essentially demonstrates 2*k* + 1 (odd definition).

# **More Practice with Regular Expressions**

Find a language (i.e. a set of strings) to describe each of the following regular expressions:

bab + c, ab\*c\*, (aa)\*ab.

{bab}∪{c} == {bab, c} Note that the + means the ∪, and there are • in between the b, a, and b. This would normally look like L(b • a • b + c).

{a}{b}\*{c}\* Find the answer by noticing the concatenation of two elements. This language is essentially where all strings that start with an a that goes next with 0 or more b’s, then 0 or more c’s. This could be portrayed as {a, ab, ac, abb, abc, acc, …}

{aa}\*{a}{b} This means you take 0 or more copies of aa, then it always ends with an ab. It’s easier to think of this as a direct translation. {ab, aaab, aaaaab}.

Find a regular expression to describe each of the following languages: {b, aba, abba},

{rowan}∪{profs}{are, tops}, {^, a, aa, aaa, aaaa, aaaaa, …}{^, b, bb, bbb, bbbb}.

b + aba + abba Since the three strings are in the set together, they must be unioned, individual strings. The language for this is equivalent to {b}∪{aba}∪{ab}∪{abba}.

(rowan + profs)(are + tops) == rowanare + rowantops + profsare + profstops The language is equivalent to {rowan, profs}{are, tops} since you can union rowan and profs. The direct translation includes {rowanare, rowantops, profsare, proftops}.

a\*(^ + b + bb + bbb + bbbb) The ellipses of that first set is the set of all strings that have 0 or more a’s in it. The second doesn’t have the ellipses, so the b doesn’t need an ellipses, but instead it’s 0 to 4 b’s.

**Know your definitions (reasoning below)!**

|  | **Alphabet?** | **Language?** | **String?** | **If lang, regular?** | **If reg, give reg expression** |
| --- | --- | --- | --- | --- | --- |
| **{a, b, c, d}** | yes | yes | no | yes | a + b + c + d |
| **{aa, bb, cc, dd}** | no | yes | no | yes | aa + bb + cc + dd |
| **{^, a, b, c, d}** | no | yes | no | yes | ^ + a + b + c + d |
| **{∅, a, b, c, d}** | no | no | no | n/a | n/a |
| **{∅, ^, a, b, c, d}** | no | no | no | n/a | n/a |
| **{abcd}** | no | yes | no | yes | abcd |
| **abcd** | no | no | yes | n/a | n/a |

**If lang, regular? (If it could be a language, is it regular?)** The criteria is as follows: If you can make a regular expression for it, then it’s regular.

Write a set for each of these. Then, is it regular? If so, give the regular expression: All strings of length 2 over the alphabet {a, b}, all strings over the alphabet {a, b} of length > 0 that contain just a’s, all strings over the alphabet {a, b} that contain exactly two b’s

aa + ab + ba + bb All of these have length 2 and are part of the set that results from the descriptions. It’s pretty simple to write this as a regular expression.

aa\* == a\*a The direct translation of this is {a, aa, aaa, …}. Think of this because the a\* indicates that there are 0 or more copies of a. You need to ensure that there is at least 1 a due to the length constraint, so concatenate that with a.

a\*ba\*ba\* The placement of the b’s isn’t specified, so this is a bit tricky. Let’s see the direct translation first. You would want {a}\*b{a}\*b{a}\*, which gives the combinations like bb and baab. The reason why this translation works is because you can have 0 or more a’s before and after the first b AND 0 or more a’s before and after the second b.

**Problem from the book: #9:** Find regular expressions for each of the following languages over the alphabet {a, b}: No string contains the substring aaa

If you have a string over the alphabet, what’s the beginning of it? You can start with b, no problem, OR one a, then b, OR two a, then b. You can repeat these terms as many times as you’d like. This process gives the regular expression (b + ab + aab)\*.

This means you can have 0 or more copies of the unioned elements inside, which is equal to {b, ab, aab}\*, and remember the star means all possible concatenations of the strings in the language, which includes {^, b, ab, aab, bab, baab, abb, abaab, aabbb, aabab, …}

This is NOT the full answer, because you might want to have an a, or a couple of a’s at the end, or end with itself. Thus, the answer is (b + ab + aab)\*(^ + a + aa).

# **Extra Examples of Regular Expressions**

| **Regular Expression (1)** | **Language (1)** |
| --- | --- |
| a + b + cb\* | {a}∪{b}∪{c}{b}\*  {a}∪{b}∪{cb*n* | n ∈ N}  {a, b, c, cb, cbbb, …} |
| a\* + c\* + d\*  *Notice that in the last one, you don’t mix up the letters since they’re unioned and NOT concatenated.* | {a}\* ∪{c}\* ∪{d}\*  {^, a, aa, aaa, …}∪{^, b, bb, bbb, …}∪{^, c, cc, ccc, …}  {^, a, c,d, aa, cc, dd, …} |
| (a + c + d)\*  *Now you union individual strings, and then take the star, so now you can come up with every string by taking 0 or more copies of the things inside. Now, you CAN mix.* | {a, c, d}\*  {^, a, c, d, aa, ac, ad, ca, cc, cd, da, dc, dd, …}  *Showing enough of the pattern.* |
| (acd)\* == (a • c • d)\* | {acd}\*  {^, acd, acdacd, acdacdacd, …} |
| (a\* + c\* + d\*)\* == (acd)\*  *This means taking 0 or more copies of the things inside. If you think about it, this is the same as the row 2 spots above; their direct translations are the same.* | {a, c, d}\* |

| **Similar reg. exp.** | a\*b\* | a\* + b\* | (a\*b\*)\* | (a\* + b\*)\* |
| --- | --- | --- | --- | --- |
| **Explanation (2)** | These are concatenated together. | These are unioned (∪) together. | These are taking 0 or more copies of the things inside, which is a concatenation, NOT a union like the right column.  Same as (a + b)\* b/c the direct translations are the same. | These are taking 0 or more copies of the things inside. This is equivalent to {a, b}\*, (a + b)\* since the stars inside don’t matter, the 0 or more copies is already done by the outside star. |
| **Direct Translations,**  **Shorthand Translations** | {^, a, b, aa, ab, bb, aaa, aab, abb, bbb, …}  {(ab)*n* | n ∈ N}  *Note, b can’t come before the a, EVER.* | {^, a, b,aa, bb, …}  {a*n* | n ∈ N}∪{b*n* | n ∈ N}  *Note, in the shorthand, the n’s are separate.* | {^, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bbb, …}  *Note, b CAN come before the a now.* |  |
| **Similar reg. exp.** | a\* + b\* + c\* | (a + b + c)\* | (a\*b\*c\*)\* | (a\* + b\* + c\*)\* |
| **Explanation (3)** | These are the unions, so there are 0 or more a’s unioned w/ 0 or more b’s unioned w/ 0 or more c’s. | The same as the 2 right columns. | You can have as many a’s, b’s, c’s you want in any order. If you want a string cc, you can either take two c’s inside the parens, 1 copy of that outside, and so on. | The same as the 2 left columns. |
| **Direct Translations,**  **Shorthand Translations** | {^, a, b, c, aa, bb, cc, …}  {a*n* | n ∈ N}∪{b*n* | n ∈ N}∪{c*n* | n ∈ N}  *Note, the a’s, b’s, c’s, separate.* | The same as the 2 right columns. | {^, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, …} | The same as the 2 left columns. |

L is the set of all strings over the alphabet {a, b, c, d} that contain substring baba. What is its regular expression?

(a + b + c + d)\*baba(a + b + c + d)\* You can have any letters before the combination baba, so you could have any of the symbols, then any number of letters after. This regular expression means you can have 0 or more copies of the elements inside the parentheses. Remember the \* means you have L0, L1, and so on forever, and the union of a, b, c, and d means that the numbers of each element can be concatenated with the numbers of others.

M is the set of all strings over the alphabet {a, b, c, d} that do not contain the substring ba. What is its regular expression?

(c + d + a + b\*c + b\*d)\*b\* == (a + b\*c + b\*d)\*b\* You know it’s infinite, so there must be a star somewhere. You can have as many a’s, c’s, and d’s as you want, and as many b’s as you want as long as you have either a c or d after it, NOT an a. Consider all possibilities and realize that you may want to have more b’s after in a string, so you should put that outside.

N is the set of strings over the alphabet {a, b, c, d} that contains only ONE copy of the substring ba. What is its regular expression?

(a + b\*c + b\*d)\*b\*ba(a + b\*c + b\*d)\*b\* Because you want only 1, combine the ideas of the previous 2 problems into this one!

Remember, {a, b, c}\* is 0 or more copies of the things inside, and {a, b, c}+ is 1 or more copies, thus meaning no lambda if there is no lambda in that set.

You CANNOT talk about a+ in regular expressions.