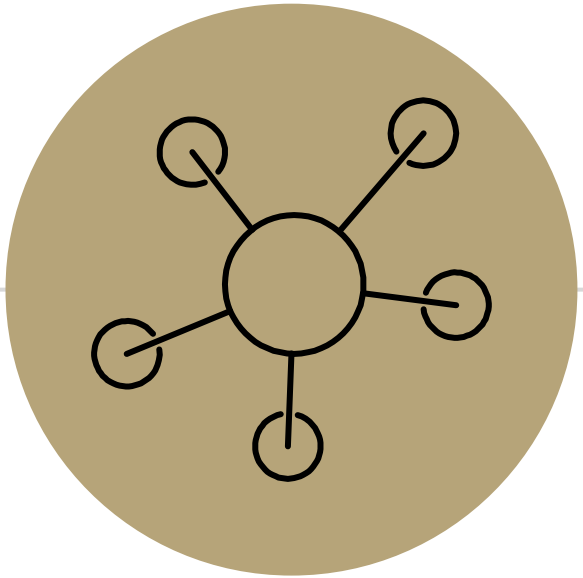
Regular Expressions

CSE 311 Autumn 2025
Lecture 22



Part 3 of the course!

Course Outline

Symbolic Logic (training wheels)

Just make arguments in mechanical ways.

Set Theory/Number Theory (bike in your backyard)

Models of computation (biking in your neighborhood)

Still make and communicate rigorous arguments

But now with objects you haven't used before.

- A first taste of how we can argue rigorously about computers.

First: regular expressions and context free grammars – understand these "simpler computers"

Soon: what these simple computers can do
Then: what simple computers can't do.

Last week: A problem our computers cannot solve.

The next two weeks

What are we learning?

Today/Friday: some theory that's useful for computer scientists

The topics for this weeks are key parts of building compilers, we'll use that as motivation—no details on compilers, but hopefully enough that you'll find it interesting.

They ALSO can be thought of as "underpowered computers."

We'll use them to build up to proving what computers can and can't do.

Next week: Tiny computers! (what can they do/what can't they do?)

Regular Expressions

I have a giant text document. And I want to find all the email addresses inside. What does an email address look like?

[some letters and numbers] @ [more letters] . [com, net, or edu]

We want to ctrl-f for a **pattern of strings** rather than a single string

Languages

A set of strings is called a **language.**

Σ^* is a language

"the set of all binary strings of even length" is a language.

"the set of all palindromes" is a language.

"the set of all English words" is a language.

"the set of all strings matching a given **pattern**" is a language.

Regular Expressions

Basis:

ε is a regular expression. The empty string itself matches the pattern (and nothing else does).

$\emptyset$ is a regular expression. No strings match this pattern.

a is a regular expression, for any $a \in \Sigma$ (i.e. any character). The character itself matching this pattern.

Recursive

If A, B are regular expressions then $(A \cup B)$ is a regular expression
matched by any string that matches A or that matches B [or both]).

If A, B are regular expressions then AB is a regular expression.
matched by any string x such that $x = yz$, y matches A and z matches B.

If A is a regular expression, then A^* is a regular expression.
matched by any string that can be divided into 0 or more strings that match A.

Regular Expressions

$(a \cup bc)$

$\{a, bc\}$

$0(0 \cup 1)1$

$\{001, 011\}$

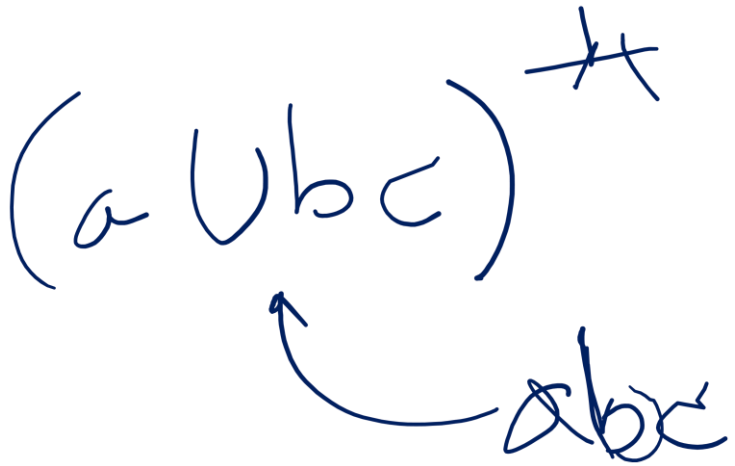
0^*

$\{\varepsilon, 0, 00, 000, 0000, \ldots\}$

$(0 \cup 1)^*$

$\{\varepsilon, 0, 1, 01, 011, 1001, 000, \ldots\}$

Regular Expressions



$(a \cup bc)$

Corresponds to $\{a, bc\}$

$0(0 \cup 1)1$

Corresponds to $\{001, 011\}$

all length three strings that start with a 0 and end in a 1.

0^*

Corresponds to $\{\varepsilon, 0, 00, 000, 0000, \ldots\}$

$(0 \cup 1)^*$

Corresponds to the set of all binary strings.

More Examples

$(0^*1^*)^*$

all binary strings

0^*1^*

$(0 \cup 1)^*(00 \cup 11)^*(0 \cup 1)^*$

$(00 \cup 11)^*$

More Examples

$(0^*1^*)^*$

All binary strings

0^*1^*

All binary strings with any 0's coming before all 1's

$(0 \cup 1)^*(00 \cup 11)^*(0 \cup 1)^*$

This is all binary strings again. Not a "good" representation, but valid.

$(00 \cup 11)^*$

All binary strings where 0s and 1s come in pairs

More Practice

(0∪1)[(0∪1)(0∪1)]*

You can also go the other way

Write a regular expression for “the set of all binary strings of odd length”

(0∪1)(00∪01∪10∪11)*

Write a regular expression for “the set of all binary strings with at most two ones”

0*(1∪ε)0*(1∪ε)0*

Write a regular expression for “strings that don’t contain 00”

(0*)∪(0*10*)∪(0*10*10*)

More Practice

You can also go the other way

Write a regular expression for “the set of all binary strings of odd length”

$(0 \cup 1)(00 \cup 01 \cup 10 \cup 11)^*$

Write a regular expression for “the set of all binary strings with at most two ones”

$0^*(1 \cup \epsilon)0^*(1 \cup \epsilon)0^*$

Write a regular expression for “strings that don’t contain 00”

$(01 \cup 1)^*(0 \cup \epsilon)$ (key idea: all 0s followed by 1 or end of the string)

Practical Advice

Check ε and 1 character strings to make sure they're excluded or included (easy to miss those edge cases).

If you can break into pieces, that usually helps.

"nots" are hard (there's no "not" in standard regular expressions)

But you can negate things, usually by negating at a low-level. E.g. to have binary strings without 00, your building blocks are 1's and 0's followed by a 1

$(01 \cup 1)^*(0 \cup \varepsilon)$ then make adjustments for edge cases (like ending in 0)

Remember $*$ allows for 0 copies! To say "at least one copy" use AA^*.

Regular Expressions In Practice

EXTREMELY useful. Used to define valid "tokens" (like legal variable names or all known keywords when writing compilers/languages)

Used in `grep` to actually search through documents.

```
Pattern p = Pattern.compile("a*b");
Matcher m = p.matcher("aaaaab");
boolean b = m.matches();
```

^	start of string	
$	end of string	
[01]	a 0 or a 1	
[0-9]	any single digit	
\.	period **\,** comma **\-** minus	
.	any single character	
ab	a followed by b	(**AB**)
(a**|**b**)**	a or b	(**A** ∪ **B**)
a**?**	zero or one of a	(**A** ∪ **ε**)
a*****	zero or more of a	**A***
a**+**	one or more of a	**AA***

e.g. **^[\-+]?[0-9]*(\.|\,)?[0-9]+$**

General form of decimal number e.g. 9.12 or -9,8 (Europe)

Regular Expressions In Practice

When you only have ASCII characters (say in a programming language)

| usually takes the place of ∪

? (and perhaps creative rewriting) take the place of ε.

E.g. $(0 \cup \varepsilon)(1 \cup 10)^*$ is `0?(1|10)*`

What **can't** regular expressions do?

Can you write a regular expression for all binary palindromes?

Can you write a regular expression for all binary palindromes of length at most 100?

What **can't** regular expressions do?

Can you write a regular expression for all binary palindromes?

No! There is no such regular expression (we'll prove it in a few weeks).

Can you write a regular expression for all binary palindromes of length at most 100?

Yes! It'll probably take you a while to write it though…

…there are a finite number of strings satisfying this description, just list them all and ∪ them together.

What **can't** regular expressions do?

Can you write a regular expression for all strings of the form $0^k 1^k$?
i.e., same number of 0's and 1's, all 0's coming first

Can you write a regular expression for $\{0^k 1^k : k \leq 100\}$
i.e., same restrictions as above, but also only at most 200 characters total.

What **can't** regular expressions do?

Can you write a regular expression for all strings of the form $0^k 1^k$?
i.e., same number of 0's and 1's, all 0's coming first

No! There is no such regular expression (we'll prove it in a few weeks).

Can you write a regular expression for $\{0^k 1^k : k \leq 100\}$
i.e., same restrictions as above, but also only at most 200 characters total.

Yes! It'll probably take you a while to write it though…
…there are a finite number of strings satisfying this description, just list them all and ∪ them together.

A Vocabulary Note

Not everything can be represented as a regular expression.
E.g. “the set of all palindromes” is not the language of any regular expression.

Some programming languages define features in their “regexes” that can’t be represented by our definition of regular expressions.
Things like “match this pattern, then have exactly that **substring** appear later.

So before you say “ah, you can’t do that with regular expressions, I learned it in 311!” you should make sure you know whether your language is calling a more powerful object “regular expressions”.

But the more “fancy features” beyond regular expressions you use, the slower the checking algorithms run, (and the harder it is to force the expressions to fit into the framework) so this is still very useful theory.

More Practice

All binary strings with a 1 in the third position. (index from 1)

All binary strings with a 1 in the third position from the end (with length at least three).

All binary strings with an even number of 0s or exactly one 1.

More Practice

All binary strings with a 1 in the third position. (index from 1)

$$(0 \cup 1)(0 \cup 1)1(0 \cup 1)^*$$

All binary strings with a 1 in the third position from the end (with length at least three).

$$(0 \cup 1)^*1(0 \cup 1)(0 \cup 1)$$

All binary strings with an even number of 0s or exactly one 1.

$$(1^*01^*01^*)^* \cup (0^*10^*)$$

Sometimes you can write two regular expressions and just ∪ together.