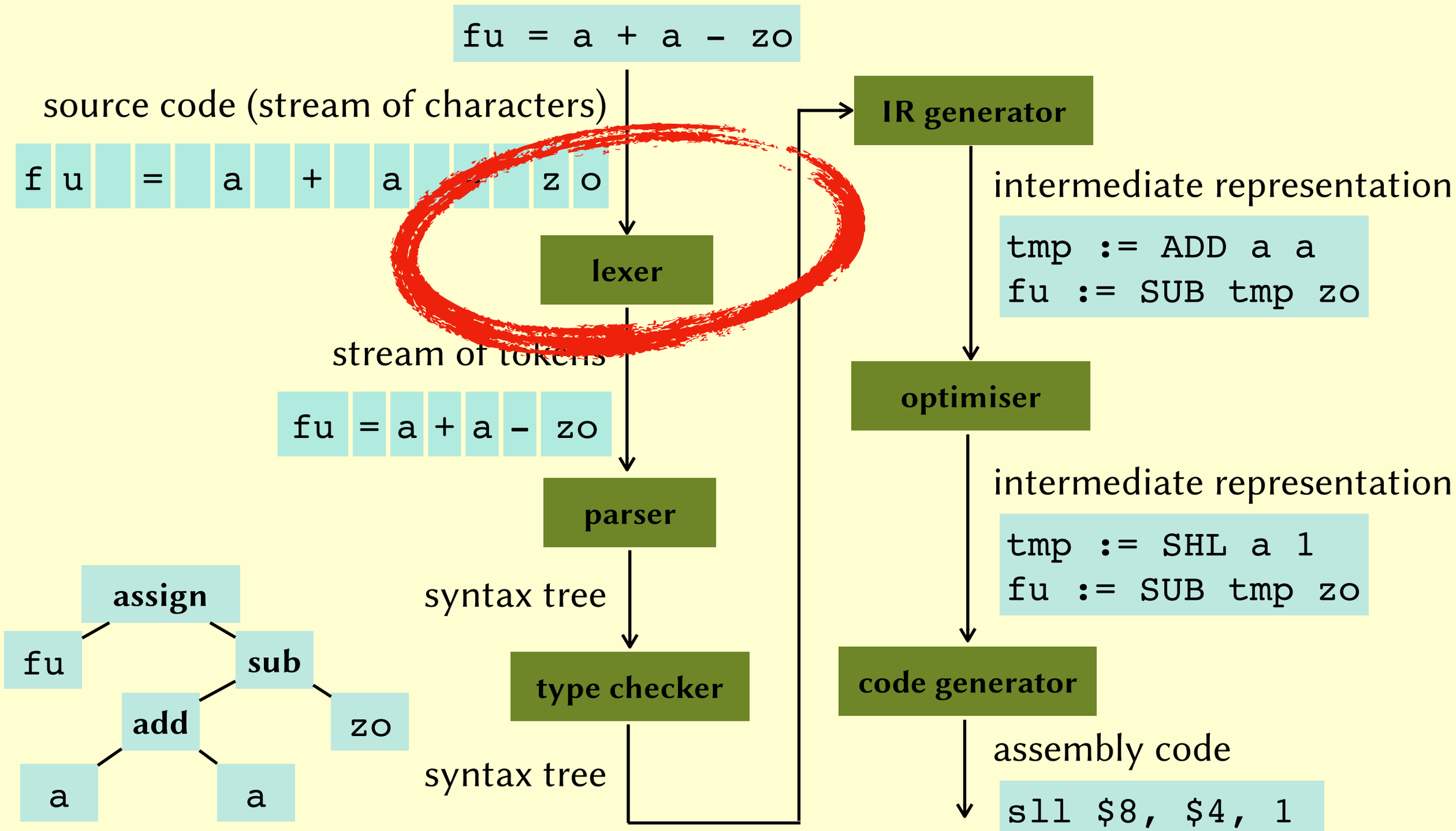
Lecture 2: Lexing and regular expressions

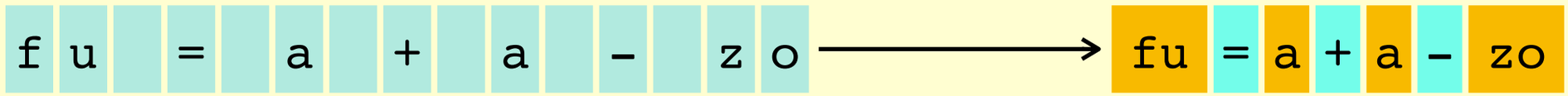
John Wickerson

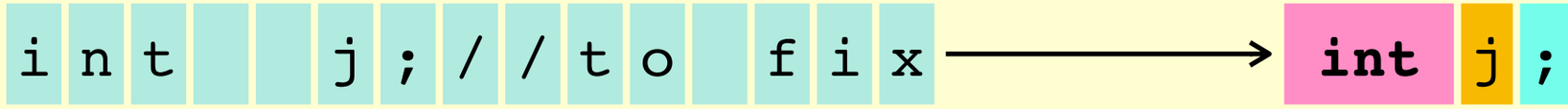
Compilers

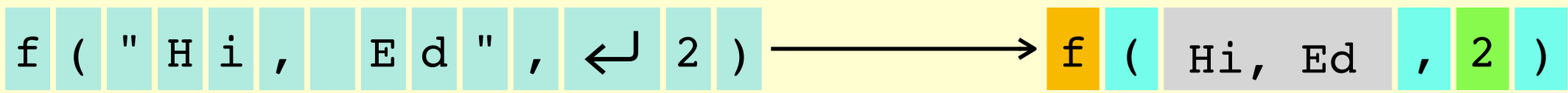
Anatomy of a compiler

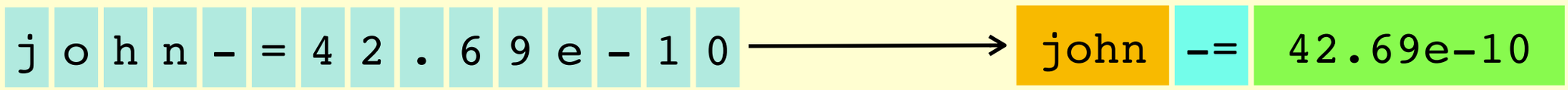


Lexing examples









Defining tokens

- The form of each type of token is defined using a **regular expression**.

- Regular expressions are built on a branch of mathematics called **Kleene algebra**.

Stephen C. Kleene
1909-1994

Words and languages

- We start by fixing an **alphabet**. English uses { `a`, `b`, `c`, ..., `z` }.

- A **word** is a finite sequence of characters.
  Example: `aardvark`.

- The **empty word** has zero characters. We write it as ε.

- If *w* and *v* are words, let us write *wv* for their **concatenation**.
  Example: if *w*=`cat` and *v*=`fish`, then *wv* = `catfish`.

- A **language** is a set of words.
  Example: { `aardvark`, `abattoir`, `abyss`, ... }.

Regular expressions

- **0** is a regex. **(zero)**
- **1** is a regex. **(unit)**
- If *c* is a single character, then *c* is a regex. **(singleton)**
- If *r* and *s* are regexes, then *r*+*s* is a regex. **(alternation)**
- If *r* and *s* are regexes, then *rs* is a regex. **(concatenation)**
- If *r* is a regex, then r^* is a regex. **(iteration)**
- **Examples.** $\texttt{aa}^*\texttt{b}$, $(\texttt{ab})^*\texttt{a}$, $(\texttt{bb})^*(\texttt{aa})^*$, $\texttt{a}(\mathbf{1}+\texttt{b})\texttt{a}$

Language accepted by a regex

- The *meaning* of a regular expression is a **language**.
  We shall write L(*r*) for the language accepted by regex *r*.

Language accepted by a regex

- The *meaning* of a regular expression is a **language**.
  We shall write L(*r*) for the language accepted by regex *r*.

- L(`aa*b`)

Language accepted by a regex

- The *meaning* of a regular expression is a **language**.
  We shall write L(*r*) for the language accepted by regex *r*.
- L(`aa*b`) = { `ab`, `aab`, `aaab`, ... }

Language accepted by a regex

- The *meaning* of a regular expression is a **language**.
  We shall write L(*r*) for the language accepted by regex *r*.
- L(`aa*b`) = { `ab`, `aab`, `aaab`, ... }
- L((`ab`)*`a`)

Language accepted by a regex

- The *meaning* of a regular expression is a **language**.
  We shall write L(*r*) for the language accepted by regex *r*.
- L(`aa*b`) = { `ab`, `aab`, `aaab`, … }
- L((`ab`)*`a`) = { `a`, `aba`, `ababa`, `abababa`, … }

Language accepted by a regex

- The *meaning* of a regular expression is a **language**.
  We shall write L(*r*) for the language accepted by regex *r*.

- L(`aa*b`) = { `ab`, `aab`, `aaab`, … }

- L((`ab`)*`a`) = { `a`, `aba`, `ababa`, `abababa`, … }

- L((`bb`)*(`aa`)*)

Language accepted by a regex

- The *meaning* of a regular expression is a **language**.
  We shall write L(*r*) for the language accepted by regex *r*.

- L(`aa*b`) = { `ab`, `aab`, `aaab`, … }

- L((`ab`)*`a`) = { `a`, `aba`, `ababa`, `abababa`, … }

- L((`bb`)*(`aa`)*) = { ε, `bb`, `aa`, `bbaa`, `bbbbaa`, … }

Language accepted by a regex

- The *meaning* of a regular expression is a **language**.
  We shall write L(*r*) for the language accepted by regex *r*.

- L(`aa*b`) = { `ab`, `aab`, `aaab`, … }

- L((`ab`)*`a`) = { `a`, `aba`, `ababa`, `abababa`, … }

- L((`bb`)*(`aa`)*) = { ε, `bb`, `aa`, `bbaa`, `bbbbaa`, … }

- L(`a`(**1**+`b`)`a`)

Language accepted by a regex

- The *meaning* of a regular expression is a **language**.
  We shall write L(*r*) for the language accepted by regex *r*.

- L(`aa*b`) = { `ab`, `aab`, `aaab`, … }

- L((`ab`)*`a`) = { `a`, `aba`, `ababa`, `abababa`, … }

- L((`bb`)*(`aa`)*) = { ε, `bb`, `aa`, `bbaa`, `bbbbaa`, … }

- L(`a`(**1**+`b`)`a`) = { `aa`, `aba` }

Language accepted by a regex

- L(**0**)

Language accepted by a regex

- $L(\mathbf{0}) = \emptyset$

Language accepted by a regex

- $L(\mathbf{0}) = \emptyset$
- $L(\mathbf{1})$

Language accepted by a regex

- $L(\mathbf{0}) = \emptyset$
- $L(\mathbf{1}) = \{ \varepsilon \}$

Language accepted by a regex

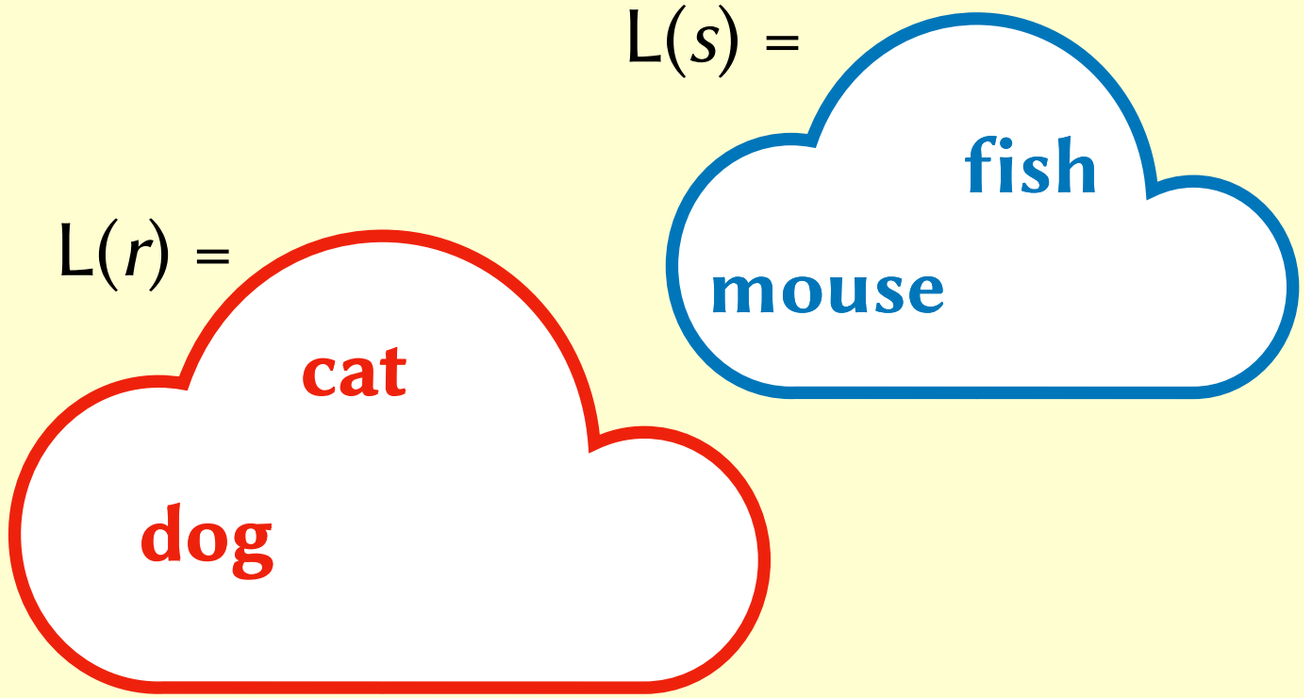
- $L(\mathbf{0}) = \emptyset$
- $L(\mathbf{1}) = \{ \varepsilon \}$
- $L(c)$

Language accepted by a regex

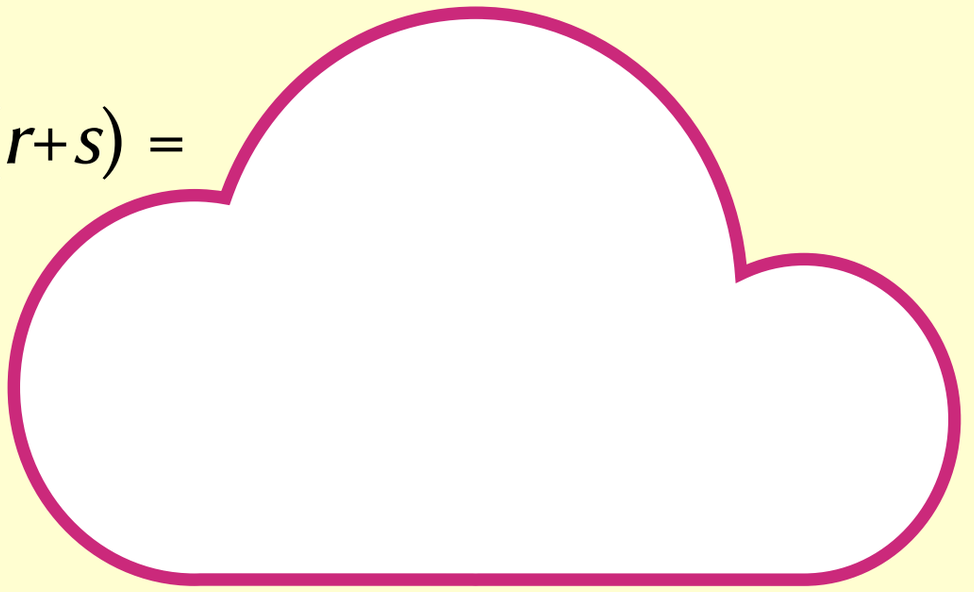
- $L(\mathbf{0}) = \emptyset$
- $L(\mathbf{1}) = \{ \varepsilon \}$
- $L(c) = \{ c \}$

Language accepted by a regex

- L(**0**) = ∅
- L(**1**) = { ε }
- L(*c*) = { *c* }
- L(*r*+*s*)

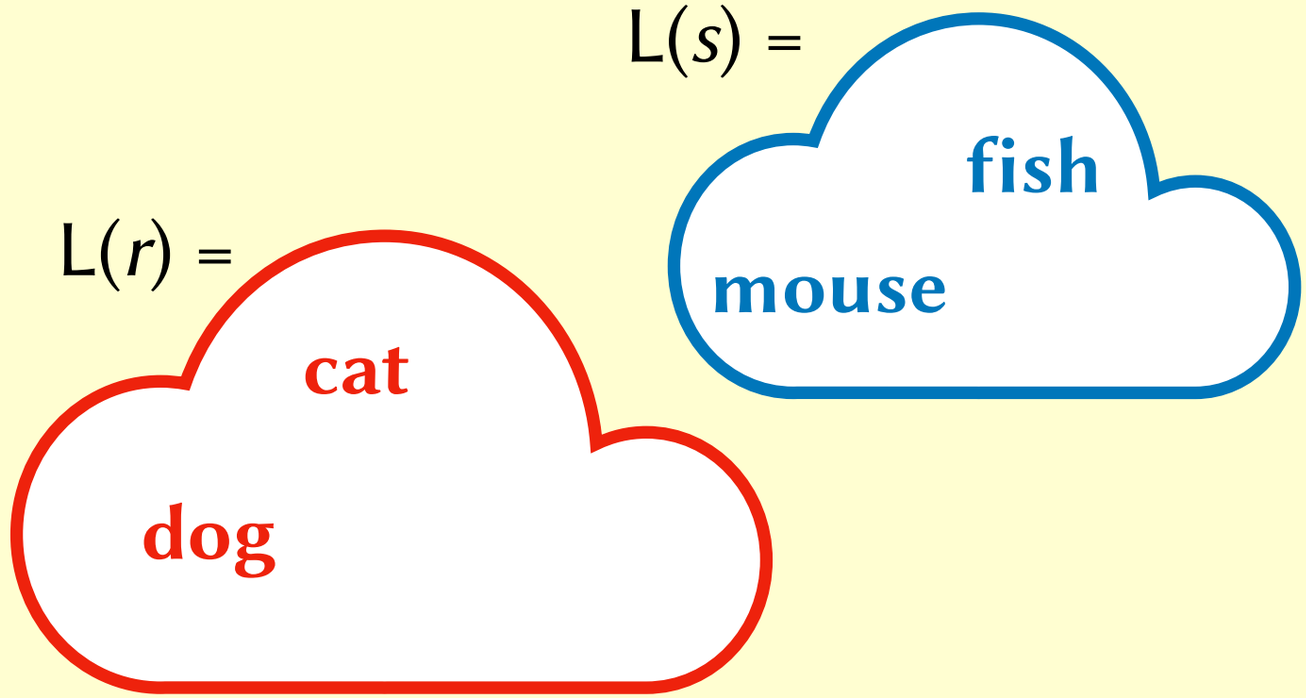


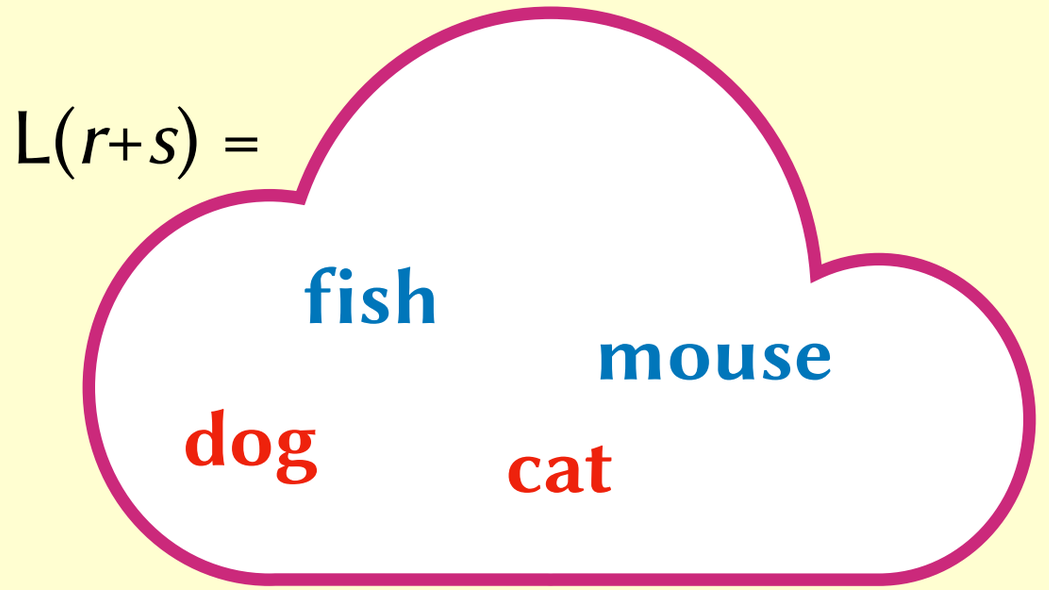
L(*r*+*s*) =



Language accepted by a regex

- L(**0**) = ∅
- L(**1**) = { ε }
- L(*c*) = { *c* }
- L(*r*+*s*)



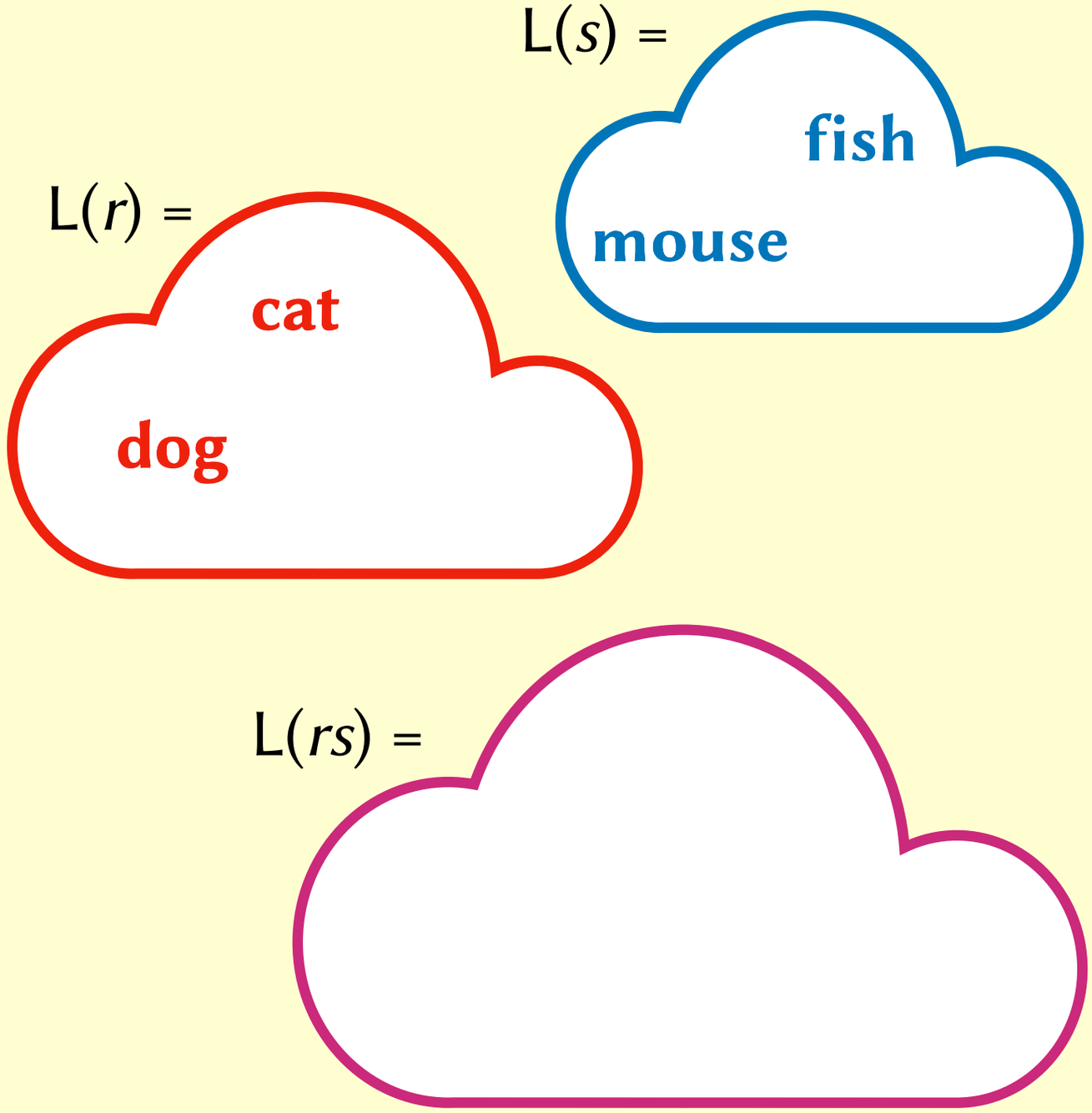


Language accepted by a regex

- $L(\mathbf{0}) = \emptyset$
- $L(\mathbf{1}) = \{ \varepsilon \}$
- $L(c) = \{ c \}$
- $L(r+s) = L(r) \cup L(s)$

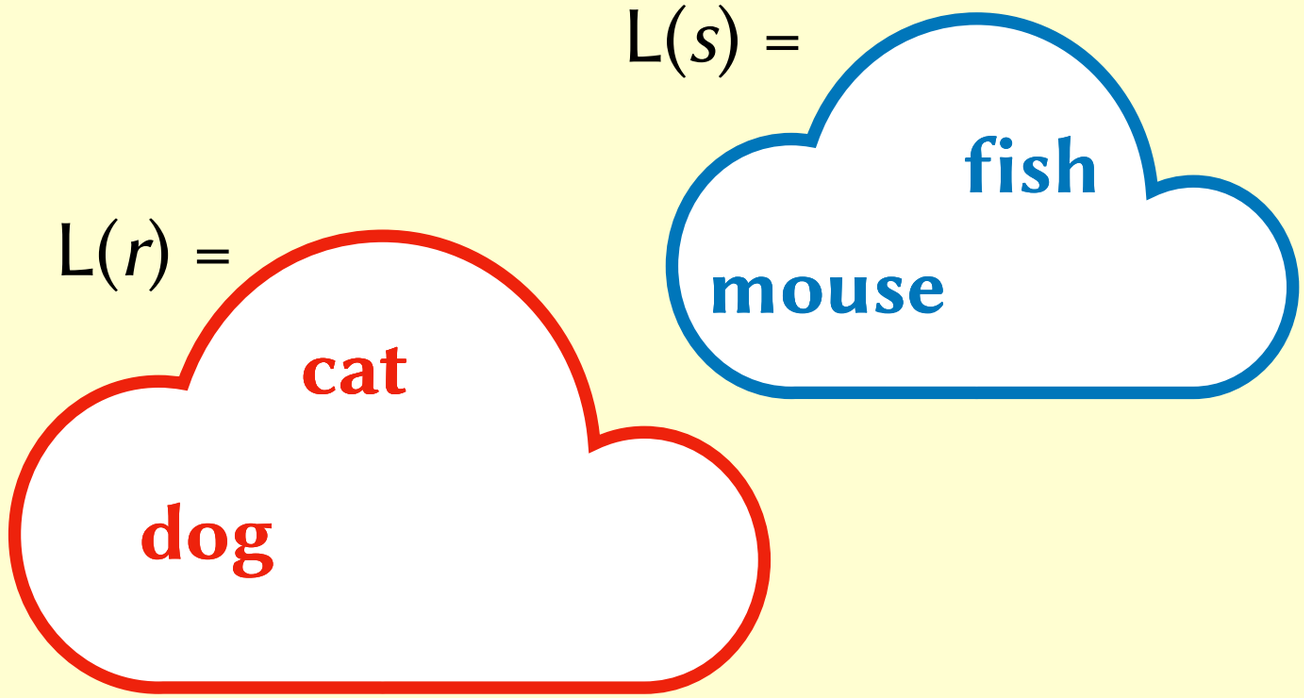
Language accepted by a regex

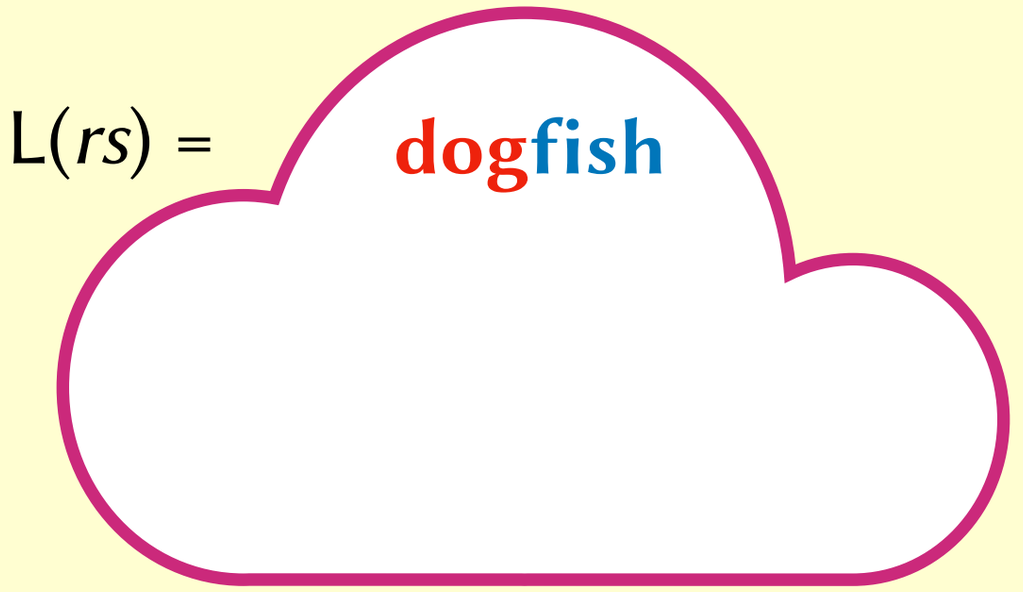
- $L(\mathbf{0}) = \emptyset$
- $L(\mathbf{1}) = \{ \varepsilon \}$
- $L(c) = \{ c \}$
- $L(r+s) = L(r) \cup L(s)$
- $L(rs)$



Language accepted by a regex

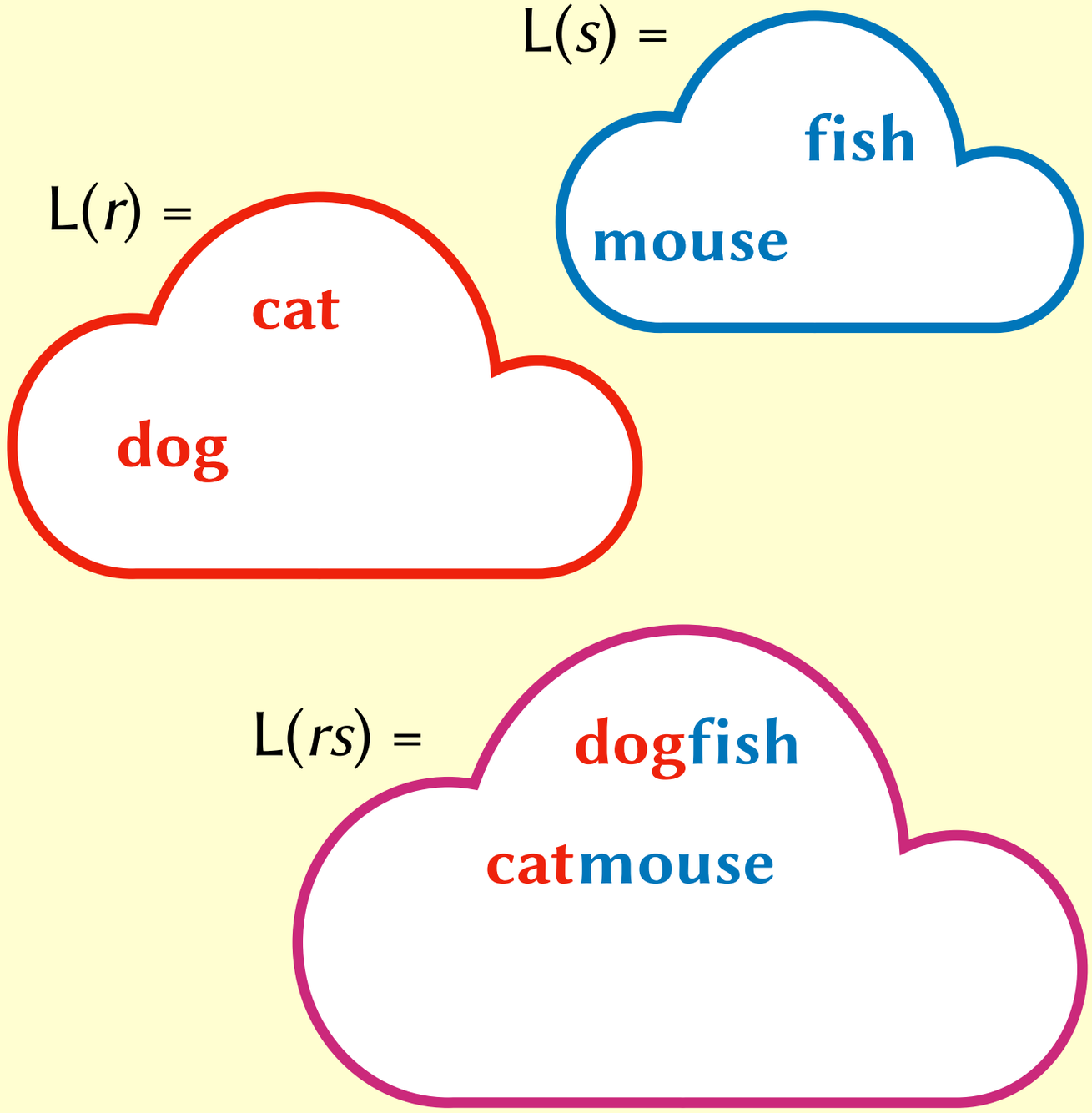
- L(**0**) = ∅
- L(**1**) = { ε }
- L(*c*) = { *c* }
- L(*r*+*s*) = L(*r*) ∪ L(*s*)
- L(*rs*)





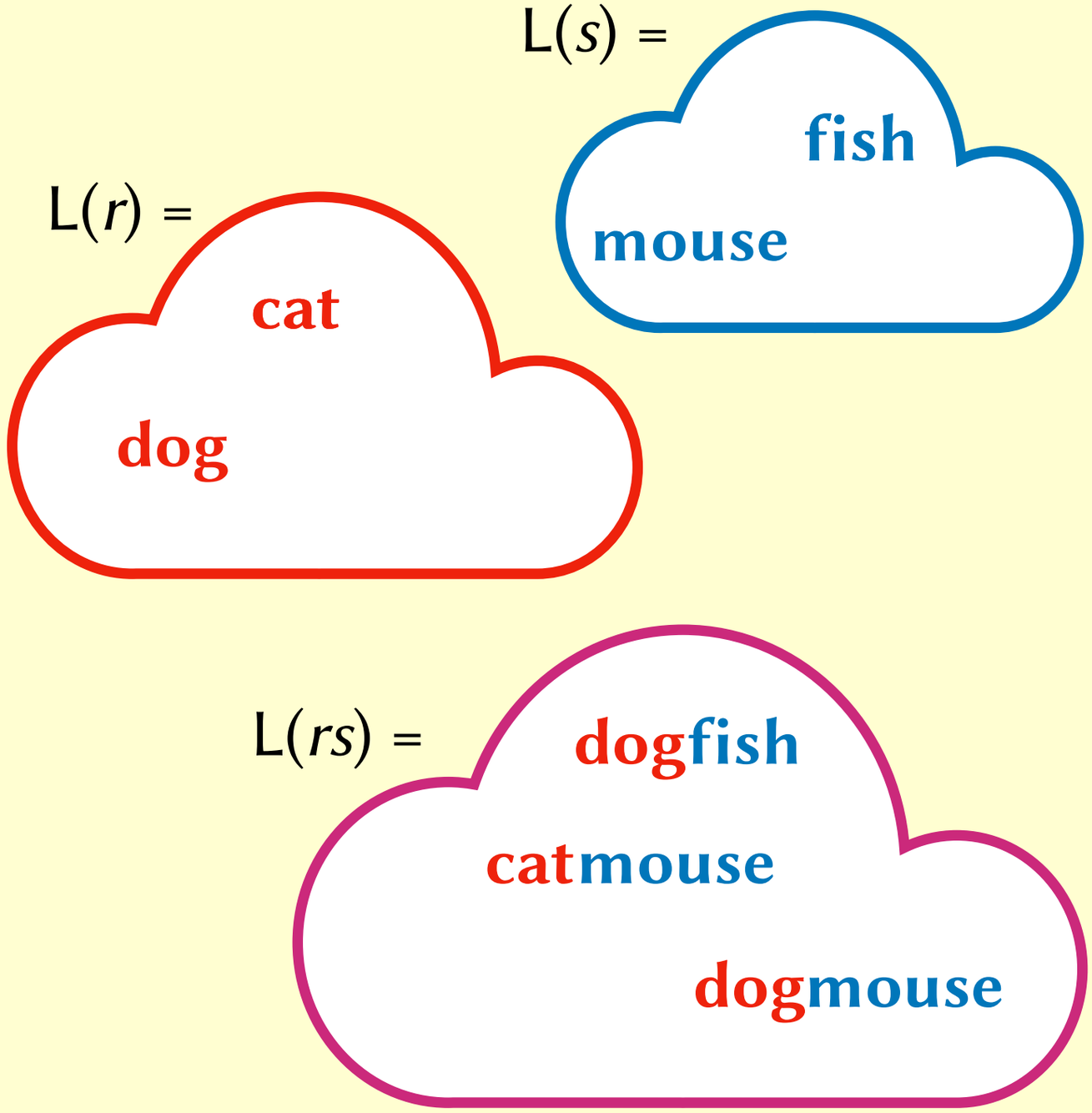
Language accepted by a regex

- $L(\mathbf{0}) = \emptyset$
- $L(\mathbf{1}) = \{ \varepsilon \}$
- $L(c) = \{ c \}$
- $L(r+s) = L(r) \cup L(s)$
- $L(rs)$



Language accepted by a regex

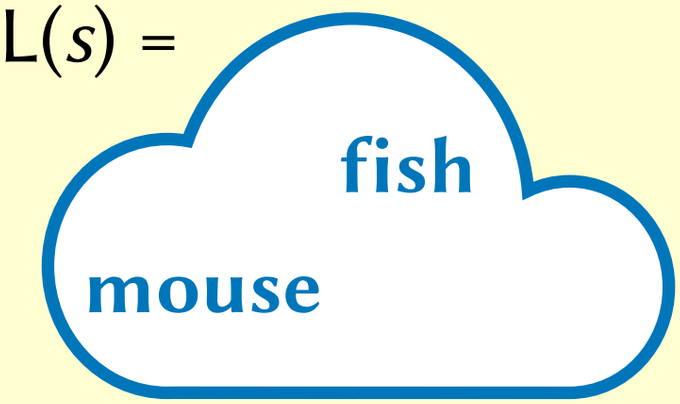
- $L(\mathbf{0}) = \emptyset$
- $L(\mathbf{1}) = \{ \varepsilon \}$
- $L(c) = \{ c \}$
- $L(r+s) = L(r) \cup L(s)$
- $L(rs)$



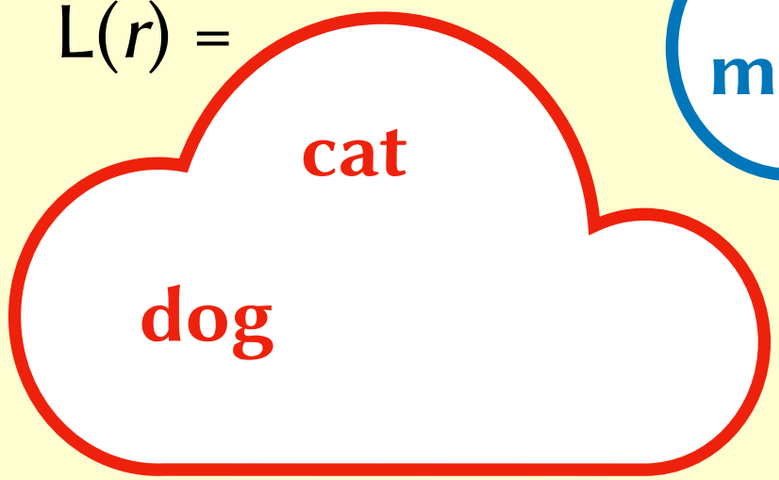
Language accepted by a regex

- $L(\mathbf{0}) = \emptyset$
- $L(\mathbf{1}) = \{ \varepsilon \}$
- $L(c) = \{ c \}$
- $L(r+s) = L(r) \cup L(s)$
- $L(rs)$

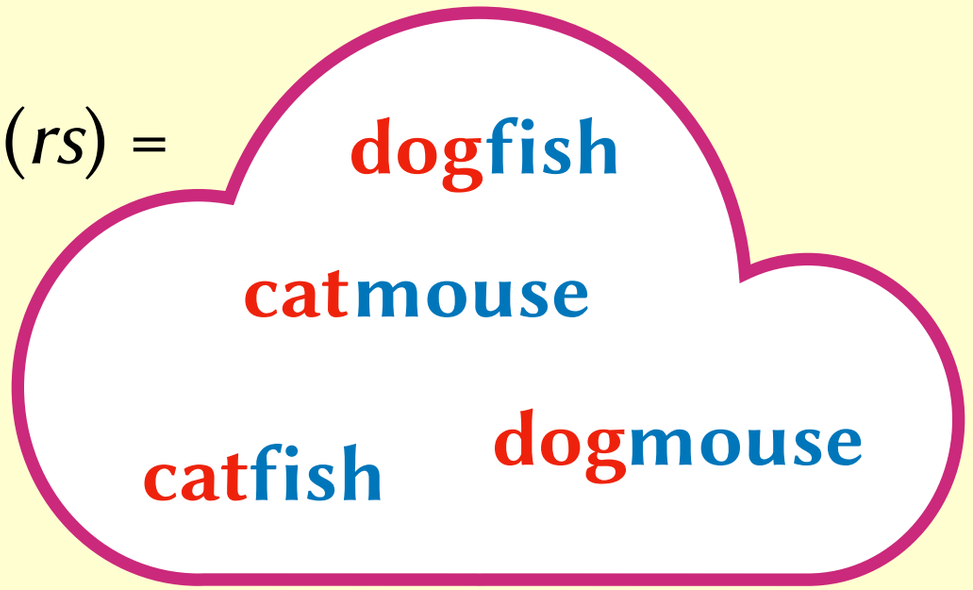
$L(s) =$



$L(r) =$



$L(rs) =$



Language accepted by a regex

- $L(\mathbf{0}) = \emptyset$
- $L(\mathbf{1}) = \{ \varepsilon \}$
- $L(c) = \{ c \}$
- $L(r+s) = L(r) \cup L(s)$
- $L(rs) = \{ w_1 w_2 \mid w_1 \in L(r) \wedge w_2 \in L(s) \}$

Language accepted by a regex

- $L(\mathbf{0}) = \emptyset$
- $L(\mathbf{1}) = \{ \varepsilon \}$
- $L(c) = \{ c \}$
- $L(r+s) = L(r) \cup L(s)$
- $L(rs) = \{ w_1 w_2 \mid w_1 \in L(r) \wedge w_2 \in L(s) \}$
- $L(r^*)$

Language accepted by a regex

- $L(\mathbf{0}) = \emptyset$
- $L(\mathbf{1}) = \{ \varepsilon \}$
- $L(c) = \{ c \}$
- $L(r+s) = L(r) \cup L(s)$
- $L(rs) = \{ w_1 w_2 \mid w_1 \in L(r) \wedge w_2 \in L(s) \}$
- $L(r^*) = L(\mathbf{1} + r + rr + rrr + rrrr + \ldots)$

Regex identities

- $r + s \equiv s + r$

Regex identities

- $r + s \equiv s + r$ (alternation is commutative)

Regex identities

- $r + s \equiv s + r$ (alternation is commutative)
- $\mathbf{0} + r \equiv r$

Regex identities

- $r + s \equiv s + r$ (alternation is commutative)
- $\mathbf{0} + r \equiv r$ (alternation has $\mathbf{0}$ as identity element)

Regex identities

- $r + s \equiv s + r$ (alternation is commutative)
- $\mathbf{0} + r \equiv r$ (alternation has $\mathbf{0}$ as identity element)
- $r + (s + t) \equiv (r + s) + t$

Regex identities

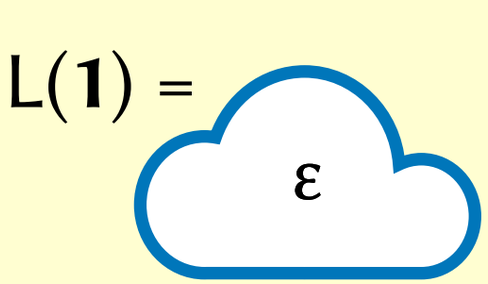
- $r + s \equiv s + r$ (alternation is commutative)
- $\mathbf{0} + r \equiv r$ (alternation has $\mathbf{0}$ as identity element)
- $r + (s + t) \equiv (r + s) + t$ (alternation is associative)

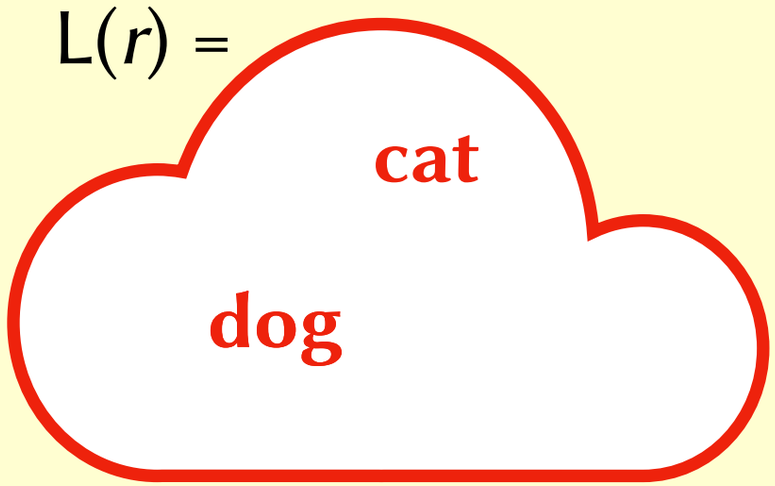
Regex identities

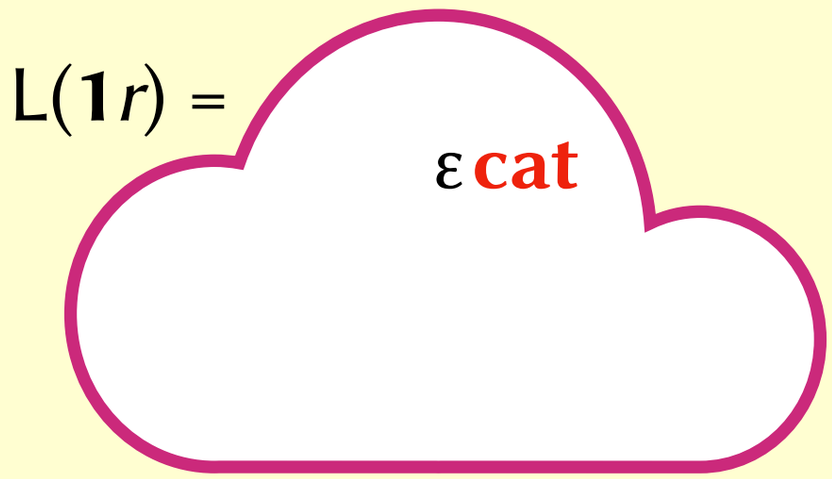
- $r + s \equiv s + r$ (alternation is commutative)
- $\mathbf{0} + r \equiv r$ (alternation has $\mathbf{0}$ as identity element)
- $r + (s + t) \equiv (r + s) + t$ (alternation is associative)
- $\mathbf{1}r \equiv r\mathbf{1} \equiv r$

Regex identities

- $r + s \equiv s + r$ (alternation is commutative)
- $\mathbf{0} + r \equiv r$ (alternation has $\mathbf{0}$ as identity element)
- $r + (s + t) \equiv (r + s) + t$ (alternation is associative)
- $\mathbf{1}r \equiv r\mathbf{1} \equiv r$







Regex identities

- $r + s \equiv s + r$ (alternation is commutative)
- $\mathbf{0} + r \equiv r$ (alternation has $\mathbf{0}$ as identity element)
- $r + (s + t) \equiv (r + s) + t$ (alternation is associative)
- $\mathbf{1}r \equiv r\mathbf{1} \equiv r$

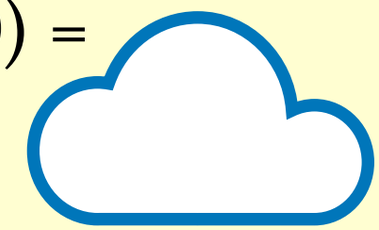
Regex identities

- $r + s \equiv s + r$ (alternation is commutative)
- $\mathbf{0} + r \equiv r$ (alternation has $\mathbf{0}$ as identity element)
- $r + (s + t) \equiv (r + s) + t$ (alternation is associative)
- $\mathbf{1}r \equiv r\mathbf{1} \equiv r$ (concatenation has $\mathbf{1}$ as identity element)

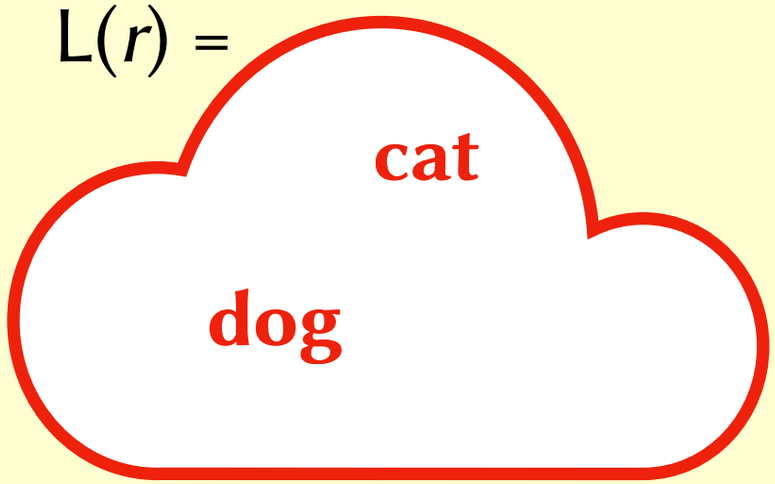
Regex identities

- $r + s \equiv s + r$ (alternation is commutative)
- $\mathbf{0} + r \equiv r$ (alternation has $\mathbf{0}$ as identity element)
- $r + (s + t) \equiv (r + s) + t$ (alternation is associative)
- $\mathbf{1}r \equiv r\mathbf{1} \equiv r$ (concatenation has $\mathbf{1}$ as identity element)
- $\mathbf{0}r \equiv r\mathbf{0} \equiv \mathbf{0}$

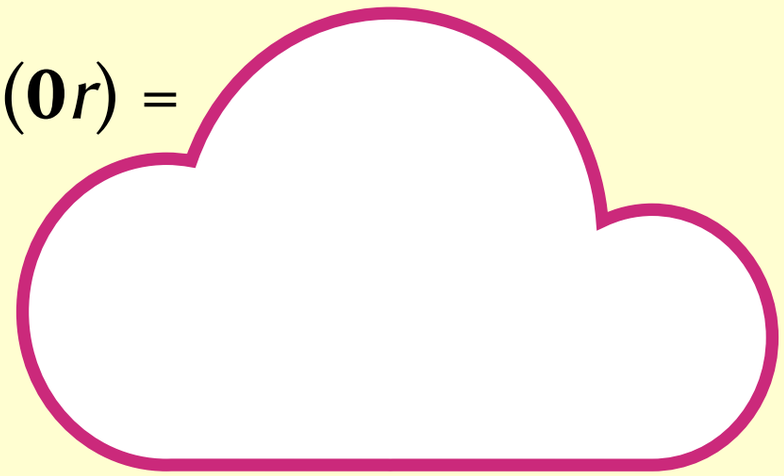
L(**0**) =



L(*r*) =



L(**0***r*) =



Regex identities

- $r + s \equiv s + r$ (alternation is commutative)
- $\mathbf{0} + r \equiv r$ (alternation has $\mathbf{0}$ as identity element)
- $r + (s + t) \equiv (r + s) + t$ (alternation is associative)
- $\mathbf{1}r \equiv r\mathbf{1} \equiv r$ (concatenation has $\mathbf{1}$ as identity element)
- $\mathbf{0}r \equiv r\mathbf{0} \equiv \mathbf{0}$ (concatenation has $\mathbf{0}$ as absorbing element)

Regex identities

- $r + s \equiv s + r$ (alternation is commutative)
- $\mathbf{0} + r \equiv r$ (alternation has $\mathbf{0}$ as identity element)
- $r + (s + t) \equiv (r + s) + t$ (alternation is associative)
- $\mathbf{1}r \equiv r\mathbf{1} \equiv r$ (concatenation has $\mathbf{1}$ as identity element)
- $\mathbf{0}r \equiv r\mathbf{0} \equiv \mathbf{0}$ (concatenation has $\mathbf{0}$ as absorbing element)
- $r(st) \equiv (rs)t$

Regex identities

- $r + s \equiv s + r$ (alternation is commutative)
- $\mathbf{0} + r \equiv r$ (alternation has $\mathbf{0}$ as identity element)
- $r + (s + t) \equiv (r + s) + t$ (alternation is associative)
- $\mathbf{1}r \equiv r\mathbf{1} \equiv r$ (concatenation has $\mathbf{1}$ as identity element)
- $\mathbf{0}r \equiv r\mathbf{0} \equiv \mathbf{0}$ (concatenation has $\mathbf{0}$ as absorbing element)
- $r(st) \equiv (rs)t$ (concatenation is associative)

Regex identities

- $r + s \equiv s + r$ (alternation is commutative)
- $\mathbf{0} + r \equiv r$ (alternation has $\mathbf{0}$ as identity element)
- $r + (s + t) \equiv (r + s) + t$ (alternation is associative)
- $\mathbf{1}r \equiv r\mathbf{1} \equiv r$ (concatenation has $\mathbf{1}$ as identity element)
- $\mathbf{0}r \equiv r\mathbf{0} \equiv \mathbf{0}$ (concatenation has $\mathbf{0}$ as absorbing element)
- $r(st) \equiv (rs)t$ (concatenation is associative)
- $r(s+t) \equiv rs + rt$

Regex identities

- $r + s \equiv s + r$ (alternation is commutative)
- $\mathbf{0} + r \equiv r$ (alternation has $\mathbf{0}$ as identity element)
- $r + (s + t) \equiv (r + s) + t$ (alternation is associative)
- $\mathbf{1}r \equiv r\mathbf{1} \equiv r$ (concatenation has $\mathbf{1}$ as identity element)
- $\mathbf{0}r \equiv r\mathbf{0} \equiv \mathbf{0}$ (concatenation has $\mathbf{0}$ as absorbing element)
- $r(st) \equiv (rs)t$ (concatenation is associative)
- $r(s+t) \equiv rs + rt$ (concatenation distributes over alternation)

Regex identities

- $r + s \equiv s + r$ (alternation is commutative)
- $\mathbf{0} + r \equiv r$ (alternation has $\mathbf{0}$ as identity element)
- $r + (s + t) \equiv (r + s) + t$ (alternation is associative)
- $\mathbf{1}r \equiv r\mathbf{1} \equiv r$ (concatenation has $\mathbf{1}$ as identity element)
- $\mathbf{0}r \equiv r\mathbf{0} \equiv \mathbf{0}$ (concatenation has $\mathbf{0}$ as absorbing element)
- $r(st) \equiv (rs)t$ (concatenation is associative)
- $r(s+t) \equiv rs + rt$ (concatenation distributes over alternation)
- $(r^*)^* \equiv r^*$

Regex identities

- $r + s \equiv s + r$ (alternation is commutative)
- $\mathbf{0} + r \equiv r$ (alternation has $\mathbf{0}$ as identity element)
- $r + (s + t) \equiv (r + s) + t$ (alternation is associative)
- $\mathbf{1}r \equiv r\mathbf{1} \equiv r$ (concatenation has $\mathbf{1}$ as identity element)
- $\mathbf{0}r \equiv r\mathbf{0} \equiv \mathbf{0}$ (concatenation has $\mathbf{0}$ as absorbing element)
- $r(st) \equiv (rs)t$ (concatenation is associative)
- $r(s+t) \equiv rs + rt$ (concatenation distributes over alternation)
- $(r^*)^* \equiv r^*$ (idempotency of iteration)

Extended regexes

$r+s$	alternation	
rs	concatenation	
r^*	r repeated 0 or more times	
r^+	r repeated 1 or more times	(same as rr^*)
$r^?$	r repeated 0 or 1 times	(same as $r + \mathbf{1}$)
`[abcdef]`	class of characters	(same as `a+b+c+d+e+f`)
`[a-f]`	range of characters	(same as `[abcdef]`)

Regex examples

- Valid times in HH:MM format:

  ([0-1][0-9] + [2][0-3]) `:` [0-5][0-9]

- **Example.** `16:20`

- UK postcodes:

  L **L**$^?$ **D** (**D** + **L**)$^?$ `_` **D** **L** **L**

  where **L** = [A-Z] and **D** = [0-9]

- **Example.** `SW7 2AZ`

Regex examples

- Integer constants:

$$\texttt{-}^{?}\ \mathbf{D}^{+}$$

- **Examples.** `42`, `-720`, `0`, `046`, `-007`, `-0`

- Fractional constants:

$$\texttt{-}^{?}\ \mathbf{D}^{+}\ (\texttt{.}\ \mathbf{D}^{+})^{?}$$

- **Examples.** `42`, `-720.00`, `42.195`, `00.3`
- **Non-examples.** `.5`, `-.75`, `5.`

Lex

- Lex was written at AT&T Labs in 1975.
- There is an open-source equivalent called Flex.

Structure of a Lex file

Definition section
%%
Rules section
%%
Code section

- Give options that tell Lex how to run
- Import C header files as necessary
- Define names for regexes
- List of regexes and associated actions
- Define any functions called in the rules section

Structure of a Lex file

Definition section
%%
Rules section
%%
Code section

```
%option noyywrap
%{
  #include <stdio.h>
%}
D [0-9]
L [A-Z]
%%
{L}{L}?{D}({D}|{L})?[ ]{D}{L}{L}\n {
    printf("Postcode: %s", yytext);
  }
.*\n {
    printf("Not a postcode: %s", yytext);
  }
%%
int main() {
    yylex();
}
```

Summary

- Lexing is about recognising **tokens** in a stream of characters.
- The form of tokens can be defined using **regular expressions**.
- Given a regex, we can define the **language** that it accepts.
- **Lex** is a language/tool for writing lexers.
- **Next lecture:** How does Lex tell whether a given word is accepted by a given regex?