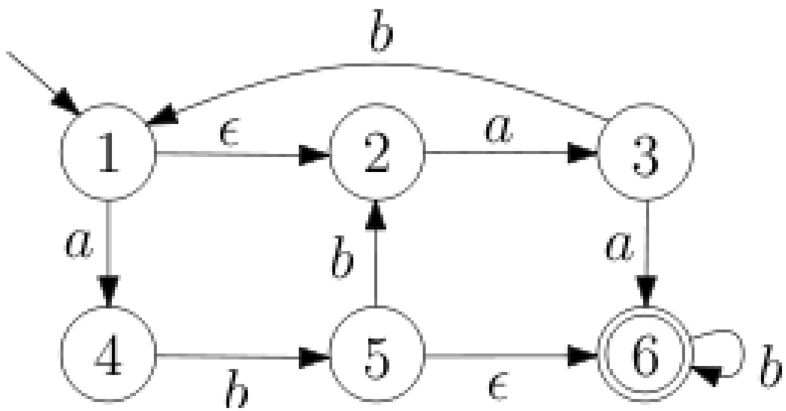
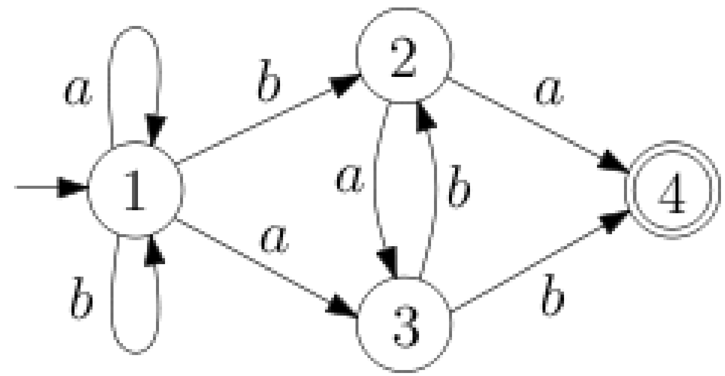
Exercise

Convert the following NFAs to deterministic finite automata.

a) 

b) 

Automated Construction of Lexers

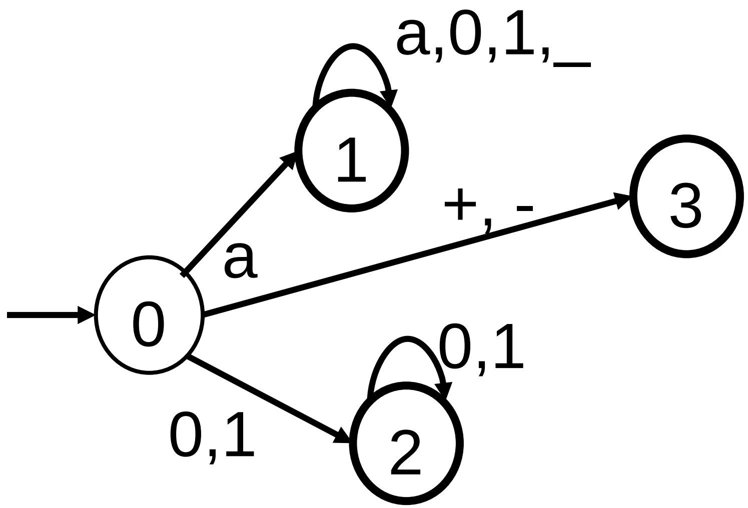
- let r_1, r_2, ..., r_n be regular expressions for token classes
  - <ID: a (a | 0 | 1 | _)*>
  - <INT: (0 | 1) (0 | 1)*>
  - <OP: + | - >

- consider combined regular expression: (r_1 | r_2 | ... | r_n)

  a (a | 0 | 1 | _)* | (0 | 1) (0 | 1)* | (+ | -)

Automated Construction of Lexers

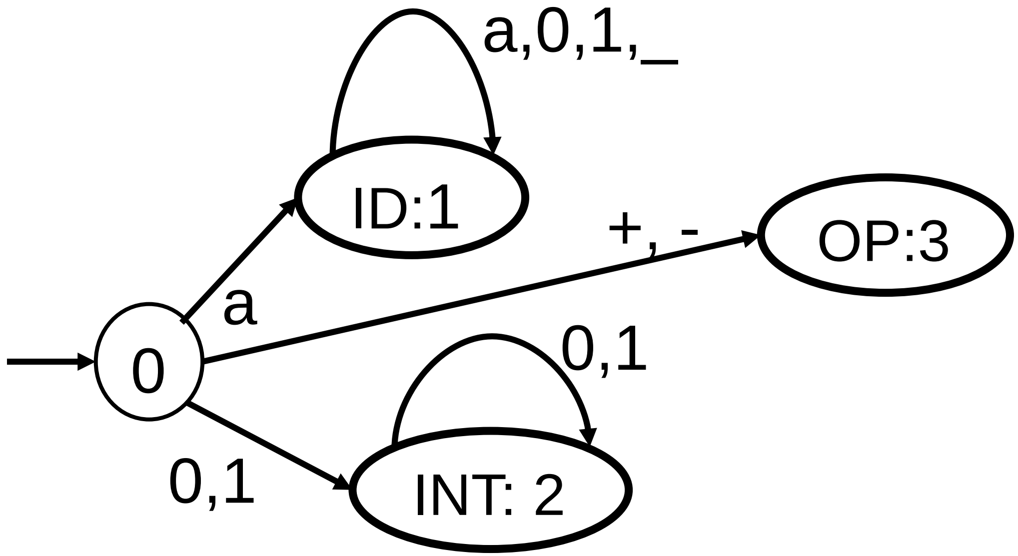
- Convert the regular expression to automaton



- For each accepting state of r_i specify the token class *i* being recognized

Automated Construction of Lexers

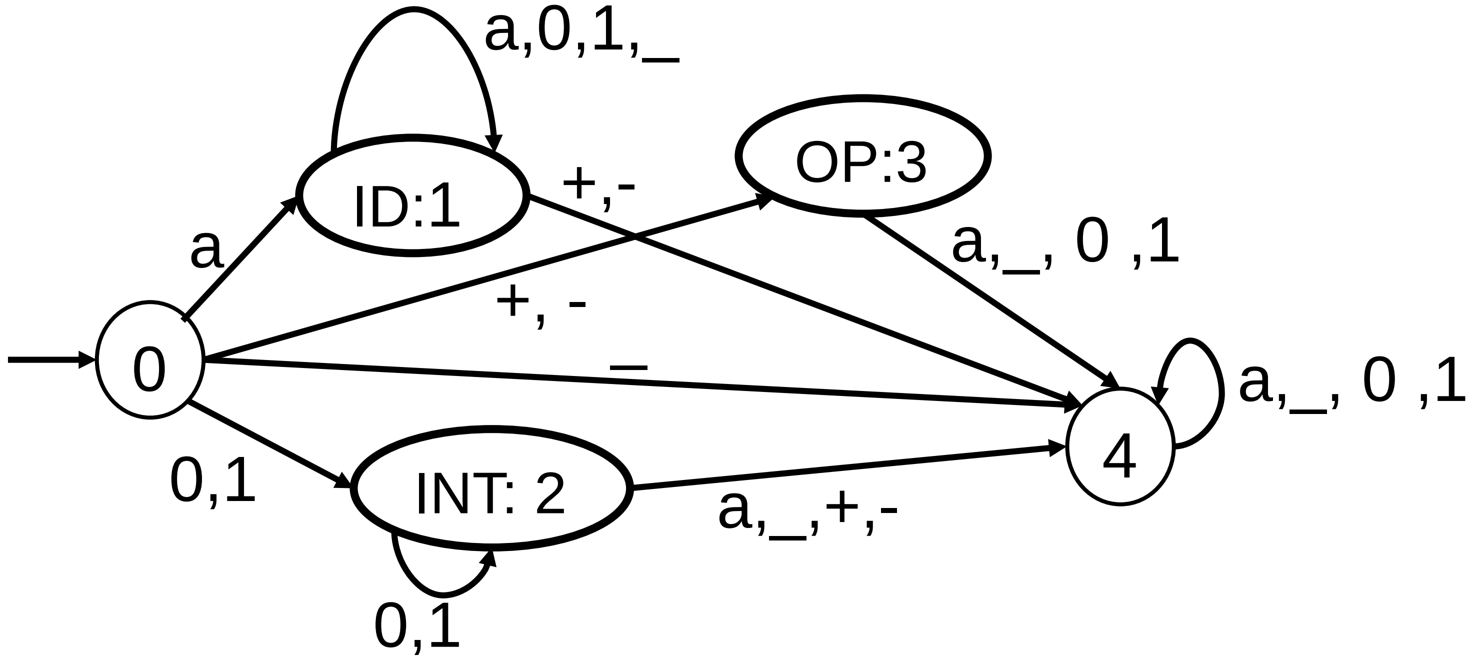
- Convert the regular expression to automaton



- For each accepting state of r_i specify the token class *i* being recognized

Automated Construction of Lexers

- Eliminate epsilon transitions and determinize
- Minimize the resulting automaton to reduce its size

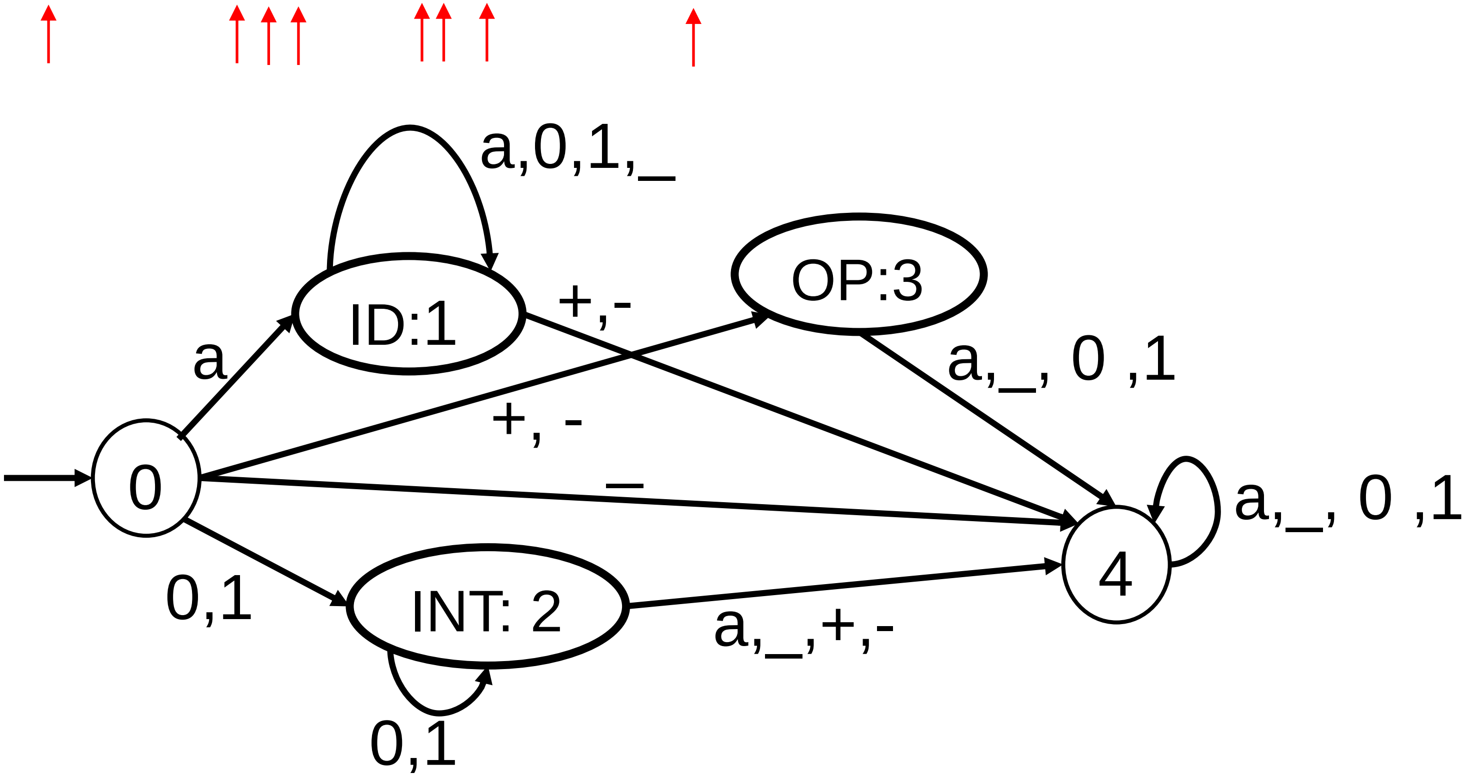


From $(r_1|r_2|\ldots|r_n)$ to a Lexer

- **Longest match rule:** remember last token and input position for a last accepted state
- When no accepting state can be reached (effectively: when we are in a trap state)
  - revert position to last accepted state
  - return last accepted token
- *Why can't we simply use $(r_1|r_2|\ldots|r_n)^*$?*

Example

- *Tokenize the following*
  - *a10110+0110-a0_10_*



Exercise

Build lexical analyzer for the following two tokens using longest match. The first token class has a higher priority:

binaryDigit ::= (**z** | 1) *

ternaryDigit ::= (0 | 1 | 2) *

1111z1021z1 →

binaryDigit: 1111z1

ternaryDigit: 021

binaryDigit: z1

Realistic Exercise: Integer Literals of Scala

- Integer literals are in three forms in Scala: decimal, hexadecimal and octal. The compiler discriminates different classes from their beginning.
  - Decimal integers are started with a non-zero digit.
  - Hexadecimal numbers begin with 0x or 0X and may contain the digits from 0 through 9 as well as upper or lowercase digits from A to F.
  - If the integer number starts with zero, it is in octal representation so it can contain only digits 0 through 7.
  - l or L at the end of the literal shows the number is Long.
- Draw a single DFA that accepts all the allowable integer literals.
- Write the corresponding regular expression.

Exercise

- Let L be the language of strings over {<, =} defined by regexp (<|=| <====*). That is, L contains <,=, and words $<=^n$ for n >= 3.
- Construct a DFA that accepts L
- Describe how the lexical analyzer will tokenize the following inputs.

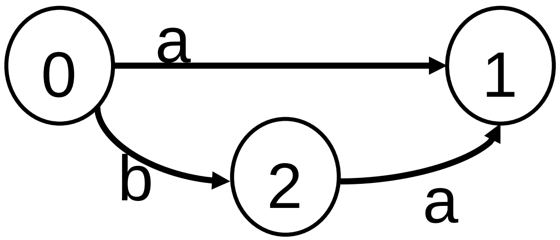
  1) <=====

  2) ==<==<==<==<==

  3) <=====<

Automata to Regular Expressions

- Every path in the automata corresponds to a RE



RE: a

RE: ab

- R_{pq}^{X} : RE corresponding to all paths from state ‘p’ to state ‘q’ that goes through only states in ‘X’
  - $R_{01}^{\emptyset}$ = a
  - R_{01}^{2} = ba

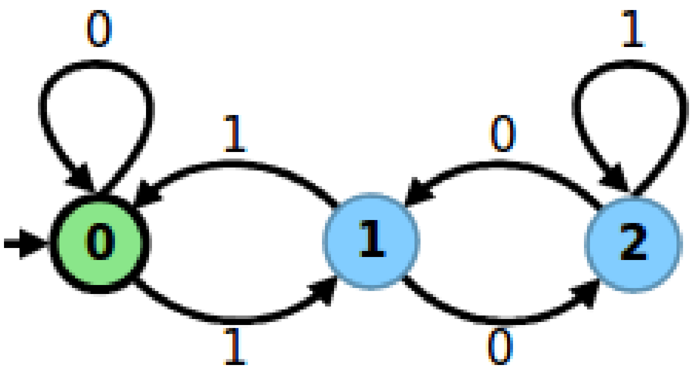
Automata to Regular Expressions

- $R_{pq}^{X} = R_{pq}^{X-\{u\}} + R_{pu}^{X-\{u\}} \left(R_{uu}^{X-\{u\}}\right)^{*} R_{uq}^{X-\{u\}}$

- $R_{pq}^{\emptyset} = a_1 + a_2 + \cdots + a_n \;,\; \delta(p, a_i) = q$
- $R_{pp}^{\emptyset} = a_1 + a_2 + \cdots + a_n + \epsilon$

- R_{sf}^{Q} is the required regular expression

Automata to Regular Expressions

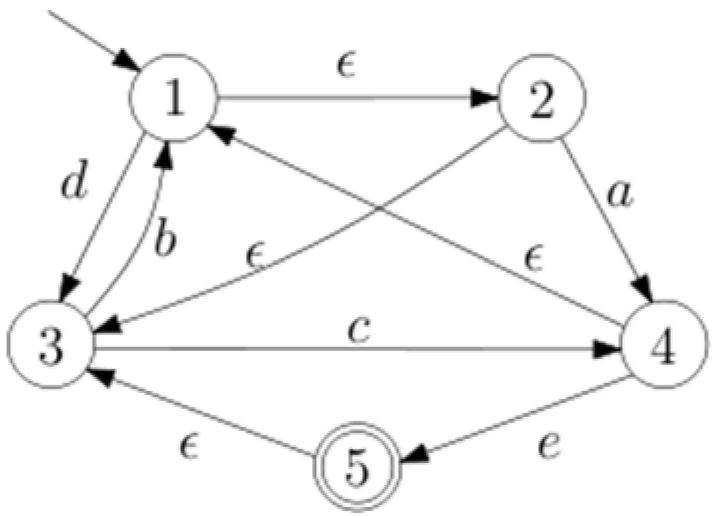


$$0^* + 0^*1(10^*1)^*10^* + 0^*1(10^*1)^*0\begin{pmatrix} 1^* \\ +0(10^*1)^*0 \end{pmatrix}^* 0(10^*1)^*10^*$$

- $R_{00}^{\{0,1,2\}} = R_{00}^{\{0,1\}} + R_{02}^{\{0,1\}}\left(R_{22}^{\{0,1\}}\right)^* R_{20}^{\{0,1\}}$
- $R_{00}^{\{0,1\}} = R_{00}^{\{0\}} + R_{01}^{\{0\}}\left(R_{11}^{\{0\}}\right)^* R_{10}^{\{0\}}$ $\quad 0^* + 0^*1(10^*1)^*10^*$
- $R_{00}^{\{0\}} = 0^*$
- ….

Exercise

- Convert the following automaton to RE



First Half of a Regular Language

Let L be a language. Define half(L) to be

$\{x \mid \text{for some } y \text{ such that } |x| = |y|, xy \text{ is in } L\}$.

That is, half(L) is the set of first halves of strings in L. Prove that if L is regular then so is half(L).

More Questions

- For which of the following languages can you find an automaton or regular expression:
  - Sequence of open or closed parentheses of even length? E.g. (), ((,)),)()))(, ... yes
  - as many digits before as after decimal point? No
  - Sequence of balanced parentheses

    ((()) ()) - balanced

    ( )) (( ) - not balanced

    No
  - Comments from // until LF Yes
  - Nested comments like /* ... /* */ ... */ No

Proof that $\{ a^n b^n \mid n >= 0 \}$ is not Regular

Say there exists a DFA with K states, i.e, $|Q|=K$

Feed it a, aa, aaa, ... Let q_i be state after reading a^i

$q_0, q_1, q_2, \ldots, q_K$

This sequence has length K+1 -> atleast one state must repeat

$q_i = q_{i+p}$ $p > 0$

Then the automaton should accept $a^{i+p}b^{i+p}$.

But then it must also accept

$$a^i \, b^{i+p}$$

because reading a^i leads to the same state as a^{i+p} .

So it does not accept the given language.

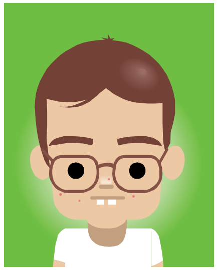
Pumping Lemma

If L is a regular language, then **there exists** a positive integer *p* (the pumping length) s.t. for **every string** $s \in L$, $|s| \geq p$, **there exists** a partition of s into three pieces, $s = x\,y\,z$,

- $|y| > 0$
- $|xy| \leq p$

such that $\boldsymbol{\forall i \geq 0}.\ xy^iz \in L$

Pumping Lemma as a Game



- Choose a ‘p’

- Pick a s in L, |s|>= p

- Split s as xyz s.t. |y|>0, |xz| <=p

- Find an i s.t. xy^iz not in L

Let’s try again: { a^nb^n | n >= 0 }

Limitations of Regular Languages

- Every automaton can be made deterministic
- Automaton has finite memory, cannot count
- If a string is too long, the automaton will repeat its behavior