

Announcements

Contents

Introduce FSM.

DFA: graphical notations, pseudo-code, math notations.

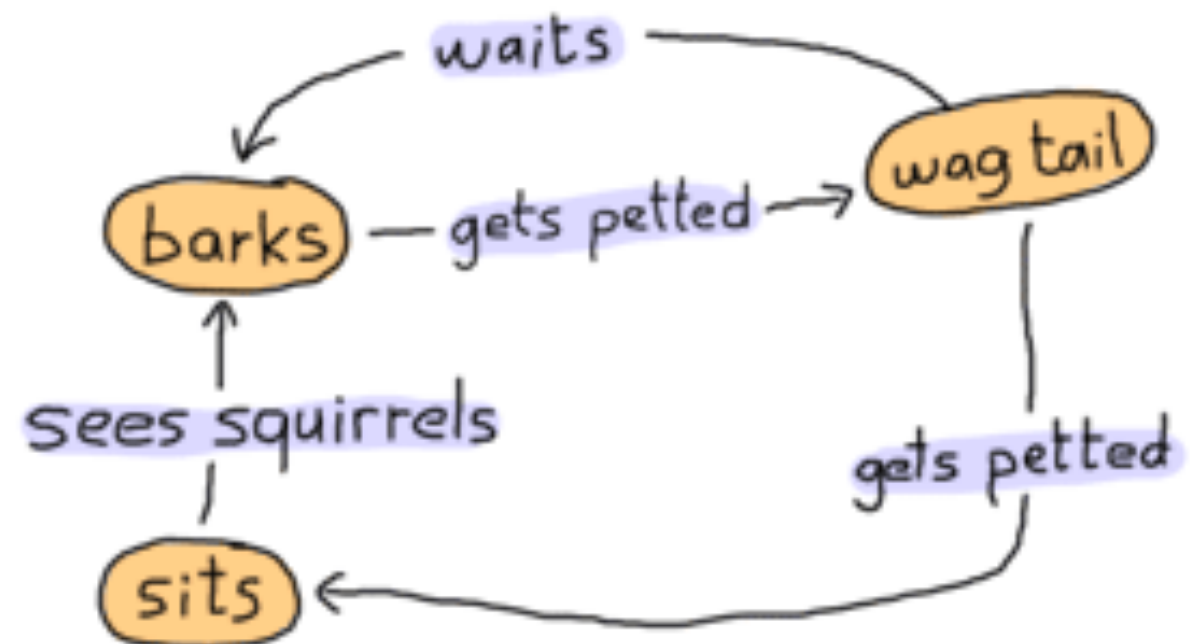
NFA: graphical notations.

P1 part 1 due next **Tuesday**

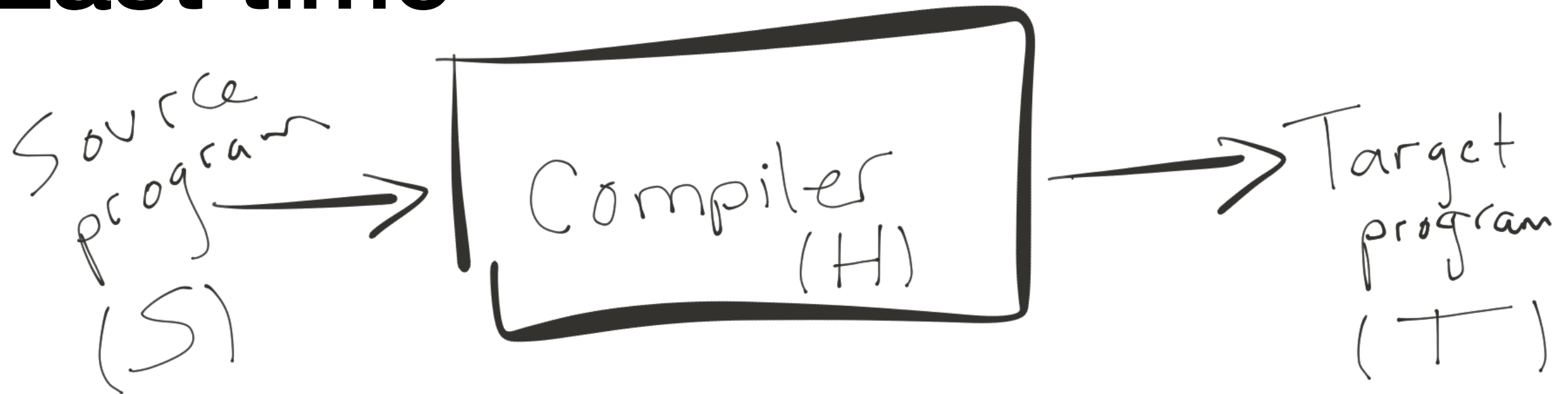
P1 part 2 due next **Friday**

Finite-state machines

CS 536



Last time



A compiler is a

- recognizer of language S (Source)

- a translator from S to T (Target)

- a program in language H (Host)

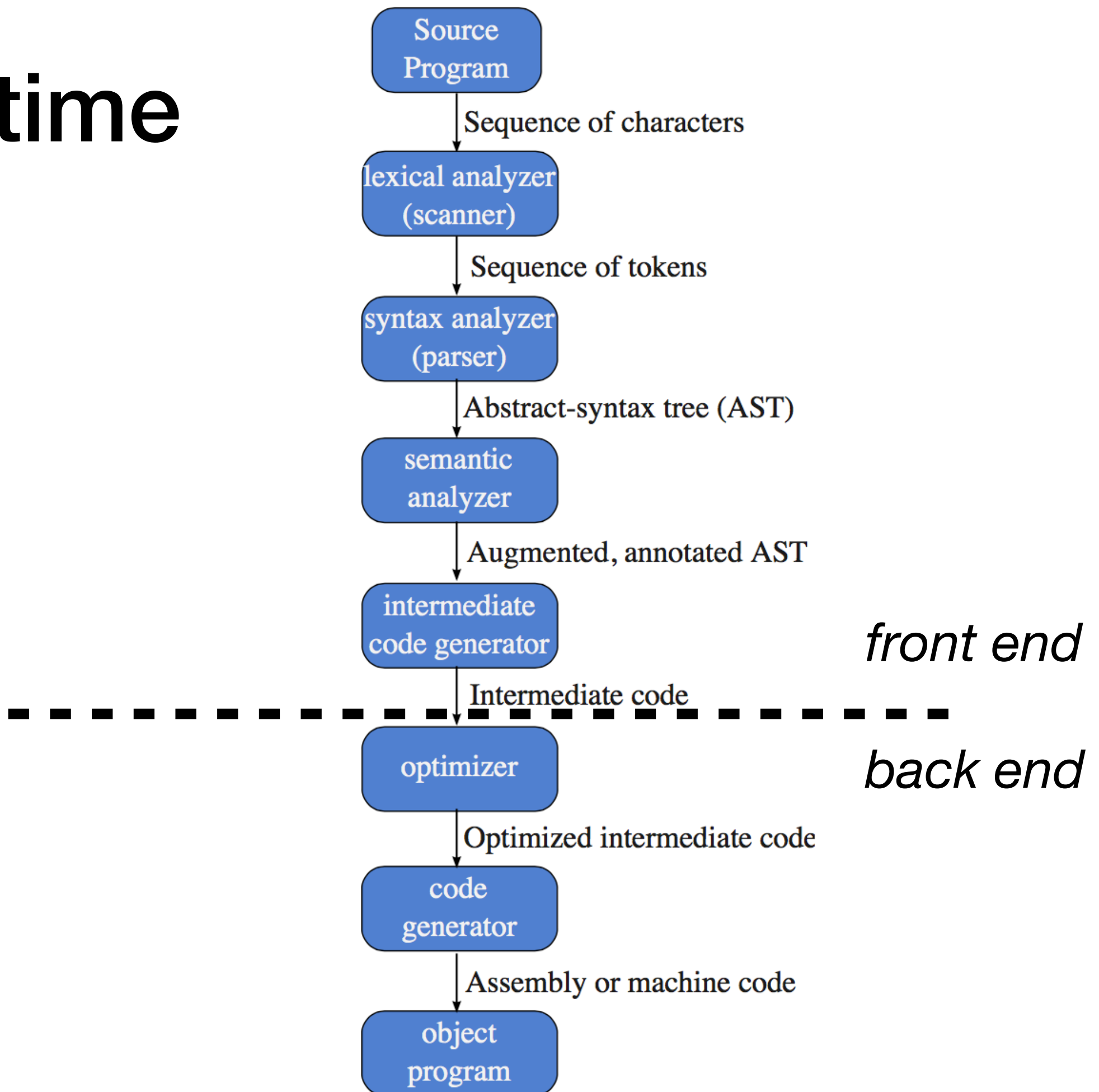
For example, gcc: S is C, T is x86, H is C

Last time

Why do we need a compiler?

- Processors can execute only binaries (machine-code/assembly programs)
- Writing assembly programs will make you want to reconsider your life choices
- Write programs in a nice(ish) high-level language like Java; compile to binaries

Last time



The scanner

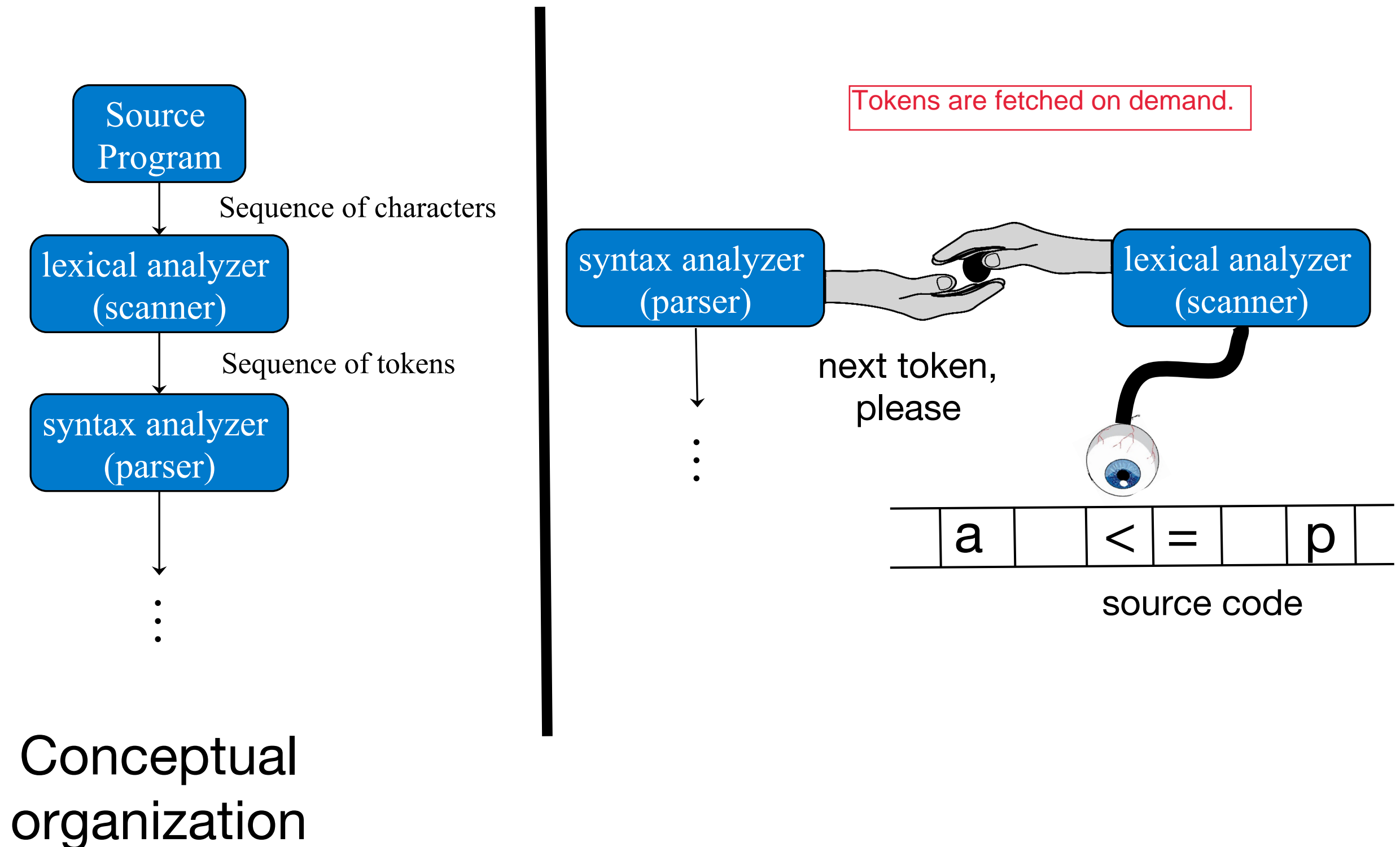
Translates sequence of chars into sequence of tokens

Each time scanner is called it should:

find longest sequence of chars corresponding to a token

return that token

Special linkage between scanner and parser in most compilers



Scanner generator

To reduce exhausting but actually redundant work. Automated.

Generates a scanner!!!

Needs one regular expression for each token

Needs regular expressions for things to ignore
comments, whitespace, etc.

To understand how it works, we need FSMs
finite state machines

FSMs: Finite State Machines

Aka finite automata

Input: string (seq of chars)

Output: accept / reject

i.e., input is legal in language

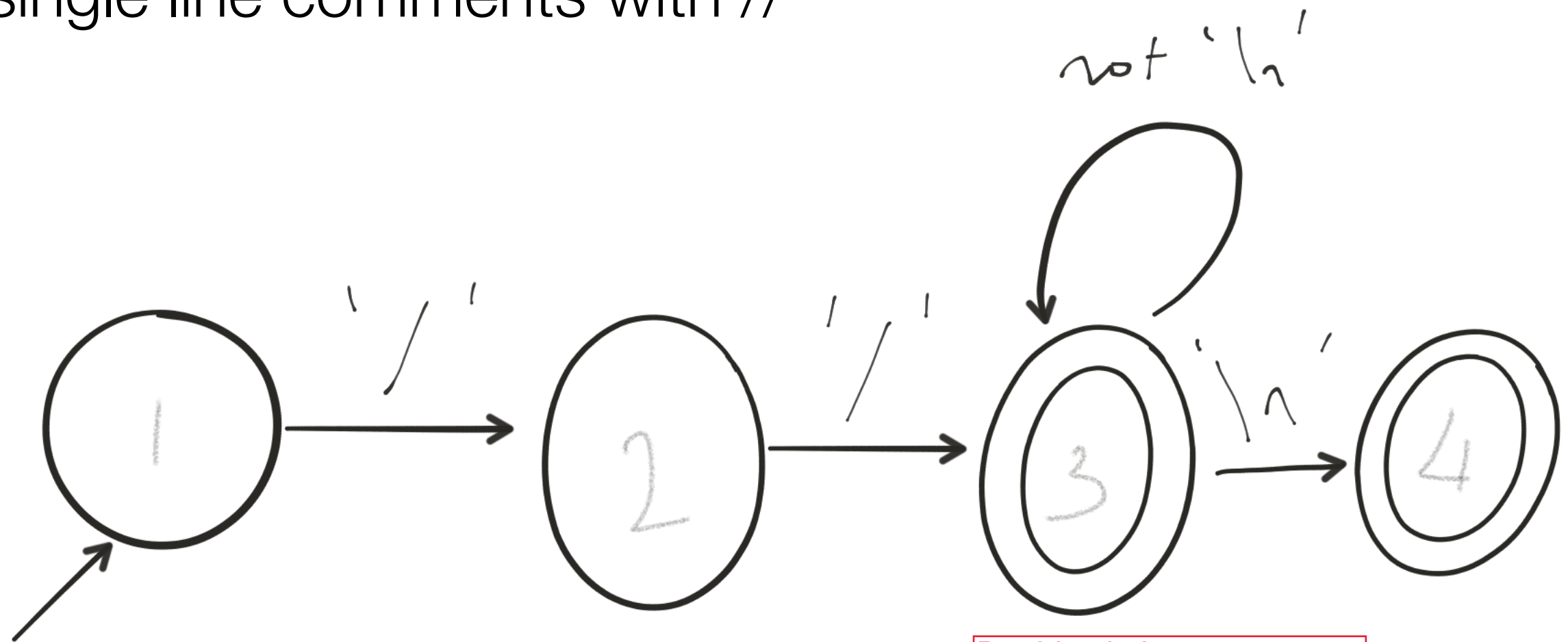
FSMs

Represent regular languages

Good enough for tokens in PLs

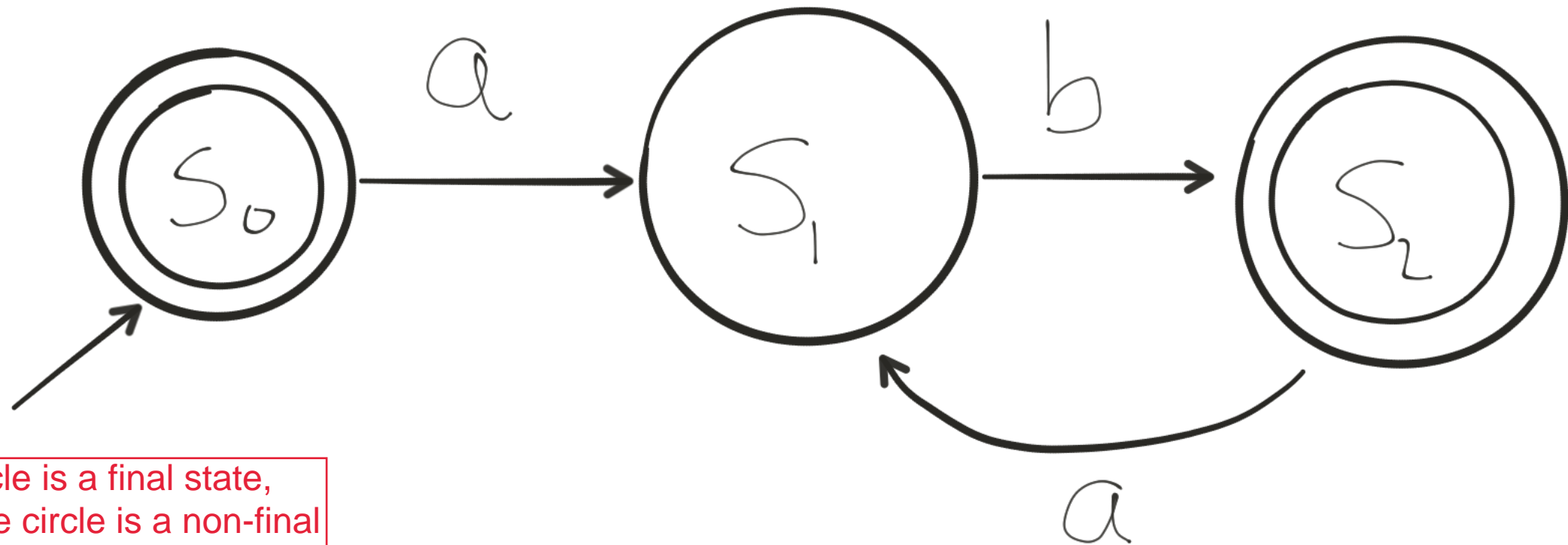
Example 1

single line comments with //



Double circle means a final-state: already valid single-line comment.

Example 2



Doubly circle is a final state, while single circle is a non-final state.

What language does this accept?

Zero or more occurrences of "ab". In regular expression: $(ab)^*$

Can you find an equivalent, but smaller, FSM?

How an FSM works

```
curr_state = start_state
```

```
let in_ch = current input char
```

```
repeat
```

```
    if there is edge out of curr_state with  
    label in_ch into next_state
```

```
        cur_state = next_state
```

```
        in_ch = next char of input
```

```
    o/w stuck // error condition
```

```
until stuck or input string is consumed
```

```
string is accepted iff entire string is  
consumed and final_states.contains(cur_state)
```

FSMs, formally

$$(Q, \Sigma, \delta, q, F)$$

finite set of states

the alphabet
(characters)

final states
 $F \subseteq Q$

start state
 $q \in Q$

transition function

$$\delta : Q \times \Sigma \rightarrow Q$$

Edges in the graph.

δ is a relationship where:
(An element of Q) \times (An element of Σ) \rightarrow (An element of Q)

FSMs, formally

$$(Q, \Sigma, \delta, q, F)$$

FSM accepts string

$$x_1 x_2 x_3 \dots x_n$$

$$\iff$$

$$\delta(\dots \delta(\delta(\delta(q, x_1), x_2), x_3) \dots, x_n) \in F$$

The language of FSM M is the set of all words it accepts,
denoted $L(M)$

FSM example, formally

$$(Q, \Sigma, \delta, q, F)$$

$$Q = \{s_0, s_1\}$$

$$\Sigma = \{a, b, c\}$$

$$q = s_0$$

$$F = \{s_0\}$$

$$\delta = s_0, a \rightarrow s_1$$
$$s_1, b \rightarrow s_0$$

	a	b	c
s0	s1		
s1		s0	

anything else, machine is stuck

Coding an FSM

```
curr_state = start_state
```

```
done = false
```

```
while (!done)
```

```
    ch = nextChar()
```

```
    next = transition[curr_state][ch]
```

```
    if (next == error || ch == EOF)
```

```
        done = true
```

```
    else
```

```
        curr_state = next
```

```
return final_states.contains(curr_state) &&
```

```
    next!=error
```

FSM types: DFA & NFA

Deterministic

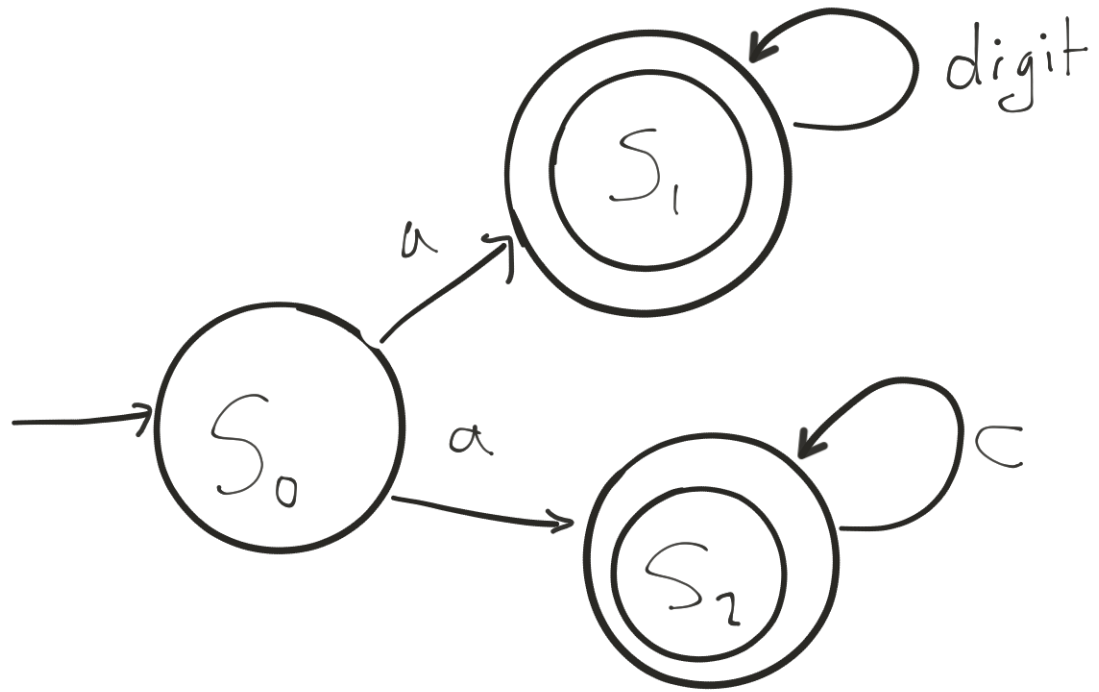
no state has > 1 outgoing edge with same label

Nondeterministic

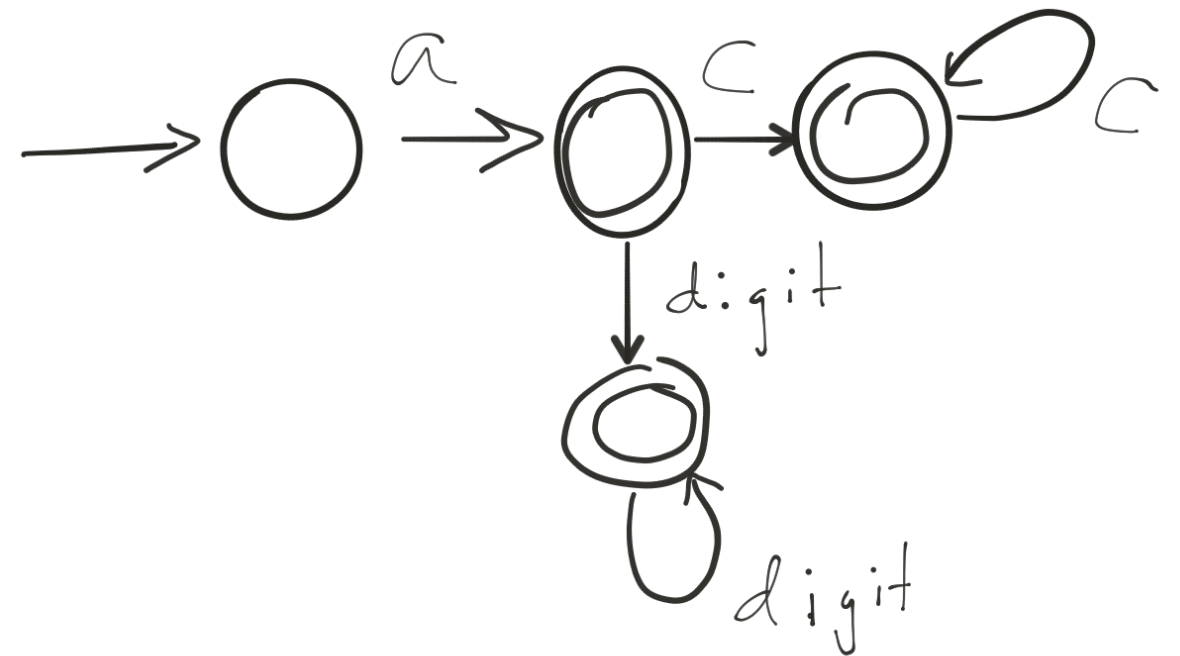
states may have multiple outgoing edges with same label
edges may be labelled with special symbol ϵ (empty string)

ϵ -transitions can happen without reading input

NFA example



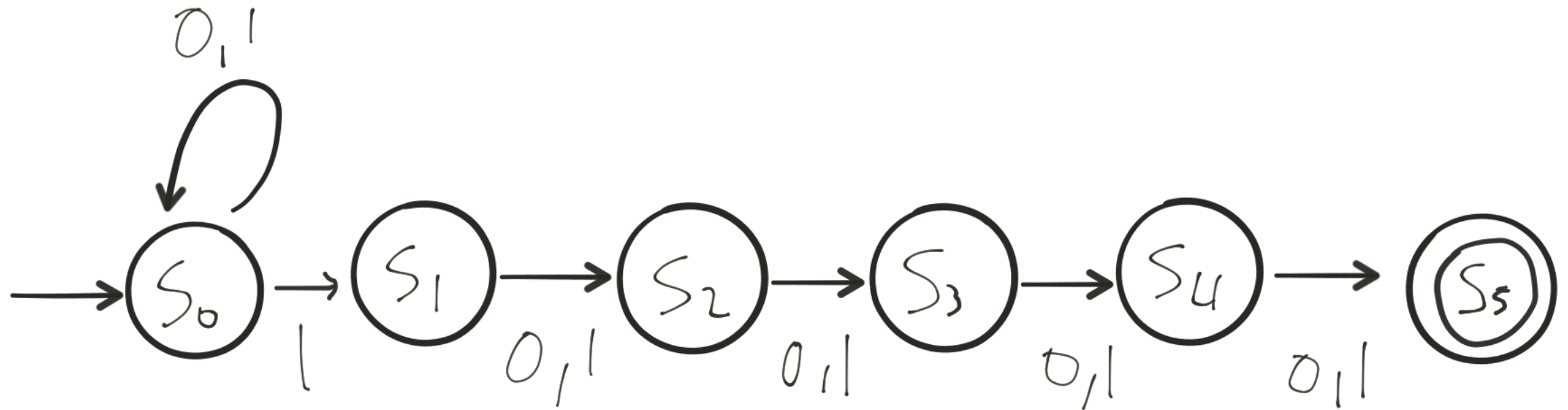
Equivalent DFA



Any NFA can be converted to a DFA.

Why NFA?

Much more compact



What does this accept?

Binary sequence whose 5-th last digit is 1.

An equivalent DFA needs 2^5 states

Extra example

Hex literals

must start with 0x or 0X

followed by at least one hex digit (0-9,a-f,A-F)

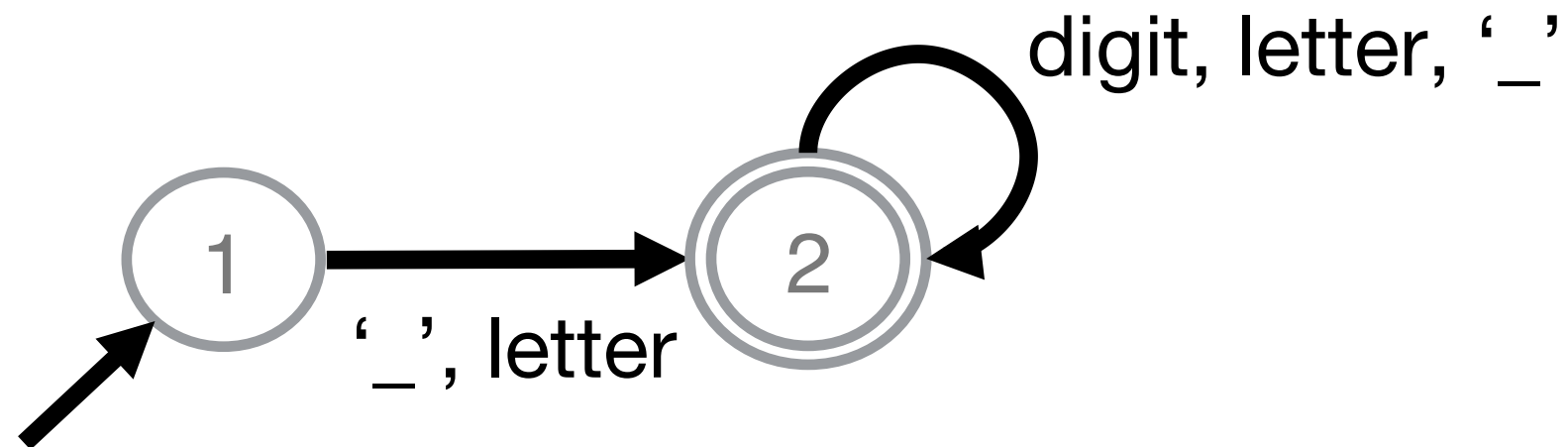
can optionally have long specifier (l,L) at the end

Extra example

A C/C++ identifier is a sequence of one or more letters, digits, or underscores. It cannot start with a digit.

Extra Example - Part 1

A C/C++ identifier is a sequence of one or more letters, digits, or underscores. It cannot start with a digit.



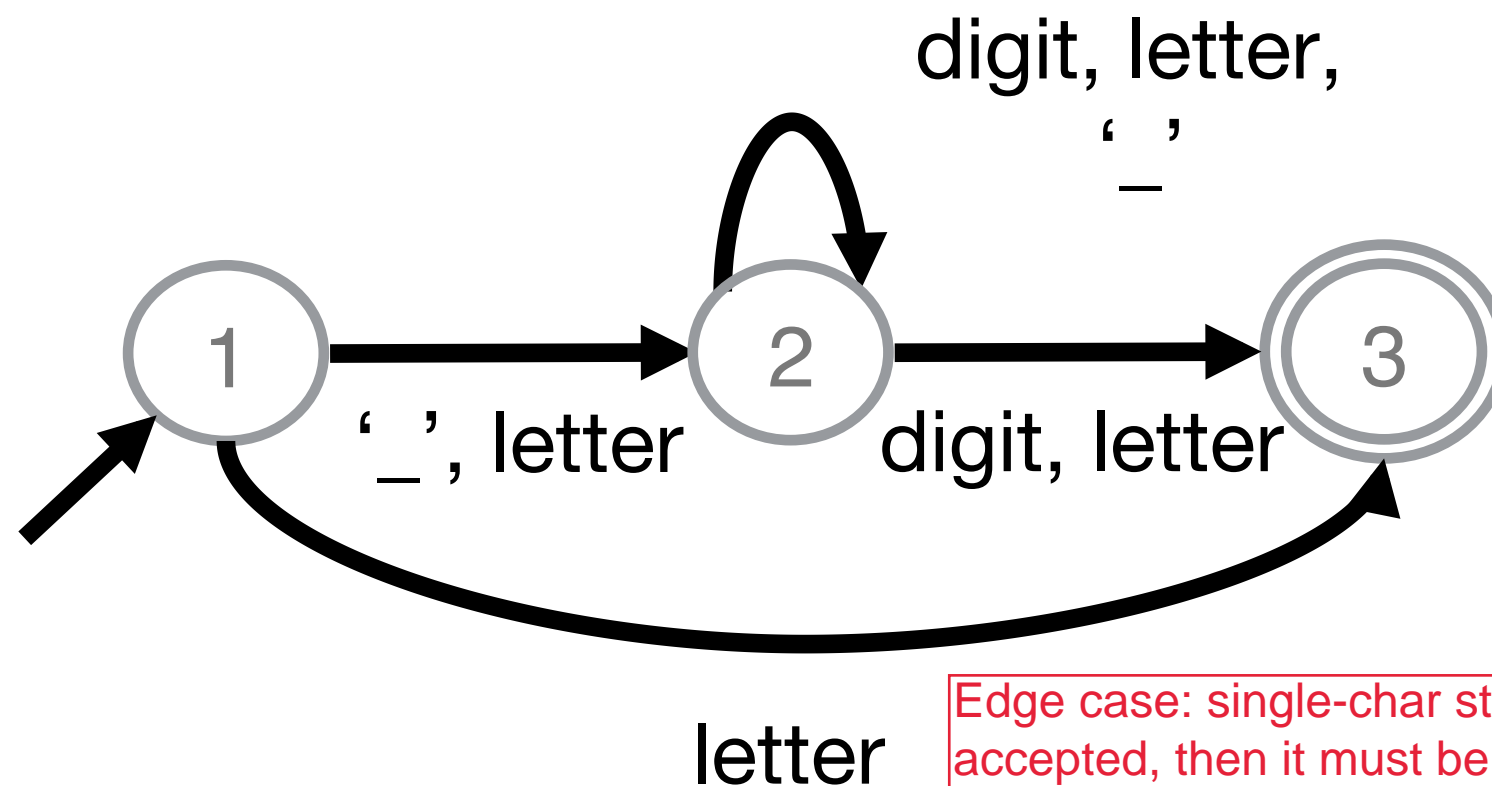
Extra example

A C/C++ identifier is a sequence of one or more letters, digits, or underscores. It cannot start with a digit.

What if you wanted to add the restriction that it can't end with an underscore?

Extra Example - Part 2

What if you wanted to add the restriction that it can't end with an underscore?



Edge case: single-char string. If a single-char word is accepted, then it must be a single-**letter** word.
General case: After finishing considering the edge case of single-char word, we can consider multi-char words, for which we are guaranteed to have a non-digit start and a non-underscore end.

Recap

The scanner reads stream of characters and finds tokens

Tokens are defined using regular expressions, which are finite-state machines

Finite-state machines can be non-deterministic

Next time: understand connection between deterministic and non-deterministic FSMs

automatatutor.com

Course ID: 219CS536S

Password: YDXXMP3E