Finite-state machines

CS 536
Last time

1. Source Program
2. Sequence of characters
3. Lexical analyzer (scanner)
4. Sequence of tokens
5. Syntax analyzer (parser)
6. Abstract-syntactic tree (AST)
7. Semantic analyzer
8. Augmented, annotated AST
9. Intermediate code generator
10. Intermediate code
11. Optimizer
12. Optimized intermediate code
13. Code generator
14. Assembly or machine code
15. Object program

---

front end

back end
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
Scanner generator

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 //
Example 2

What language does this accept?

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 cur_state = final_state
FSMs, formally

\[(Q, \Sigma, \delta, q, F)\]

- **finite set of states** \(Q\)
- **the alphabet (characters)** \(\Sigma\)
- **start state** \(q \in Q\)
- **final states** \(F \subseteq Q\)
- **transition relation** 
  \[\delta : Q \times \Sigma \rightarrow Q\]
FSMs, formally

\[(Q, \Sigma, \delta, q, F)\]

FSM accepts string

\[x_1x_2x_3 \ldots x_n\]

\[\iff\]

\[\delta(\ldots \delta(\delta(q, x_1), x_2), x_3) \ldots, x_n) \in F\]

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

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

\(Q = \{s_0, s_1\}\)
\(\Sigma = \{a, b, c\}\)
\(q = s_0\)
\(F = \{s_0\}\)
\(\delta = s_0, a \rightarrow s_1\)
\(s_1, b \rightarrow s_0\)

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 curr_state == final_state
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 $\varepsilon$ (empty string)
$\varepsilon$-transitions can happen without reading input
NFA example

Equivalent DFA
Why NFA?

Much more compact

What does this accept?

An equivalent DFA needs $2^5$ states
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.

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