DFA for MIPS

- It is easy to write a DFA to recognize an individual token type
- It is not difficult to combine these into one DFA that recognizes a word as some token type
- See the DFA for MIPS (lexer.cc or asm.rkt)
The scanning problem:

**Input:** string $w$ and a language $L$ (set of all valid tokens)

**Output:** $w_1 w_2 \cdots w_n = w$ where $w_i \in L$ for all $i$

There may be more than one possible answer: 0x1234abcd

We solve this problem by...
Simplified Maximal Munch

Input is a word $c_0c_1\cdots c_{k-1}$

\[ i = 0 \quad // \quad i \text{ is the index of the current character} \]

\[ \text{state} = \text{START} \]

\text{loop}

\[ \text{newstate} = \text{ERROR} \]

\text{if} \ (i < k ):

\[ \text{newstate} = \delta(\text{state}, c_i) \]

\text{if newstate} == \text{ERROR:}

\[ \text{if state is not a final state:} \]

\[ \quad \text{report an error and exit} \]

\text{if state is not WHITESPACE:}

\[ \quad \text{output appropriate token} \]

\[ \text{state} = \text{START} \]

\text{if} \ i == k:

\[ \quad \text{exit} \]

\text{else:}

\[ \quad \text{state} = \text{newstate} \]

\[ i = i + 1 \]
Big Picture of Compilation

Big picture:

- lexical analysis
  - scanning (done for you A3+4, your turn on A6)

- syntactic analysis (a.k.a. parsing)
  - tokens → intermediate rep (I.R.) (A3+4 Pass1, A6 = algorithm, A7 = parse WLP4)

- context-sensitive (semantic) analysis
  - Verify I.R. (A4: verify label refs, A8: verify variable + function refs, and do type checking)

- synthesis (a.k.a. code generation)
  - I.R. → binary (A3+4 Pass 2 for MIPS assembly, A9+10 for WLP4)

Other points:

- staging can improve error messages
- use the best tool for the job
- doing extra-work at an early stage is possible but over-complicating
Non-regular languages

Give a DFA over $\Sigma = \{a, b\}$ for

$$L = \{w: \text{numbers of a’s in } w = \text{number of b’s in } w\}$$

- Can’t be done
- CS241 Proof: Because the only way is to count, and a DFA has no memory besides which state it’s in
- CS360 Proof: Pumping Lemma
Why that languages is important

- L is not a regular language, but it is context-free
- WLP4 is not a regular language, but it is context-free
- Why?
Contex-Free Languages

- Context-free languages are built from:
  - finite sets
  - concatenation
  - union
  - recursion

- Recognizers for regular languages use a finite amount of memory

- Recognizers for context-free languages use a finite amount of memory plus one (unbounded) stack
Pushdown Automata

- Finite Automata plus a stack.

\[ M = \{ Q, \Sigma, \Gamma, \delta, q_0, \gamma_0, A \} \]

- \( \Gamma \) is the stack alphabet (doesn’t need to be the same alphabet)
- \( \gamma_0 \) is the starting stack (stack can’t be empty).
- \( \delta : Q \times (\Sigma \cup \{ \epsilon \}) \times \Gamma \to Q \times \Gamma^* \).
  - wat
  - \( \delta \) inputs are: state you’re in, next symbol in word, top of stack (popped by calling \( \delta \))
  - \( \delta \) outputs are: what state to change to, what 0 or more symbols to push back onto the stack
  - an \( \epsilon \) transition means ”end of input” not what it means in an epsilon-NFA

- There’s no common way to draw sweet diagrams like for a DFA, because the transitions have so many labels.

- Example: \( L = \{ w : \text{numbers of a’s in } w = \text{number of b’s in } w \} \)
Context-Free Grammars

A way to specify a context-free language.
A CFG Example

\[
\begin{align*}
S & \rightarrow aSb \\
S & \rightarrow D \\
D & \rightarrow cD \\
D & \rightarrow \epsilon
\end{align*}
\]
Discussion of example

- G: (Pretty much just the previous slide plus some context)

- \( L(G): a^n c^* b^n \) for all \( n \).

- A word: \( aabb \)

- A derivation: \( S \rightarrow aSb \rightarrow aaSbb \rightarrow aaDbb \rightarrow aabb \)

- Alternation and concatenation: \( S \) can be \( aSb \) or it can be \( D \) (alternation), \( aSb \) is concatenation

- Recursion vs. repetition: \( D \rightarrow cD \) is \( D^* \) (repetition), but \( S \)'s first rule isn't repetition. Recursion is strictly more powerful
Formal Definition

A context-free grammar (CFG) consists of

- \(N\) 
  Non-terminal symbols (symbols that cannot occur in the words of \(L\) and must be replaced)

- \(T\) 
  Terminal symbols (symbols that occur in the words of \(L\), and cannot be replaced)

- \(P\) 
  Production rules (ways to replace Non-terminal symbols)

- \(S\) 
  Start symbol \(\in N\)
Example, more formally

From the example, what is N, T, P, S?

\[ N = \{ S, D \} \quad T = \{ a, b, c \} \quad S = S \]
\[ D = \{ \]
\[ S \rightarrow aSb \\
S \rightarrow D \\
D \rightarrow cD \\
D \rightarrow \epsilon \]
\[ \} \]
Example: balanced parentheses

Example words:

CFG:

A sample derivation:
Binary expressions

Words are composed of binary numbers (no leading zeros, other than 0) with + or - signs in infix notation.

Example words: 1001, 10+1, 11-11110+0

CFG: