CS536 Lecture 3

Tuesday 27 January 2015

Reminders:

- P1 Part 1 due tomorrow night (no late submissions), Part 2 due Feb 2.
- HW1 assigned, due next Tuesday.

Last class:

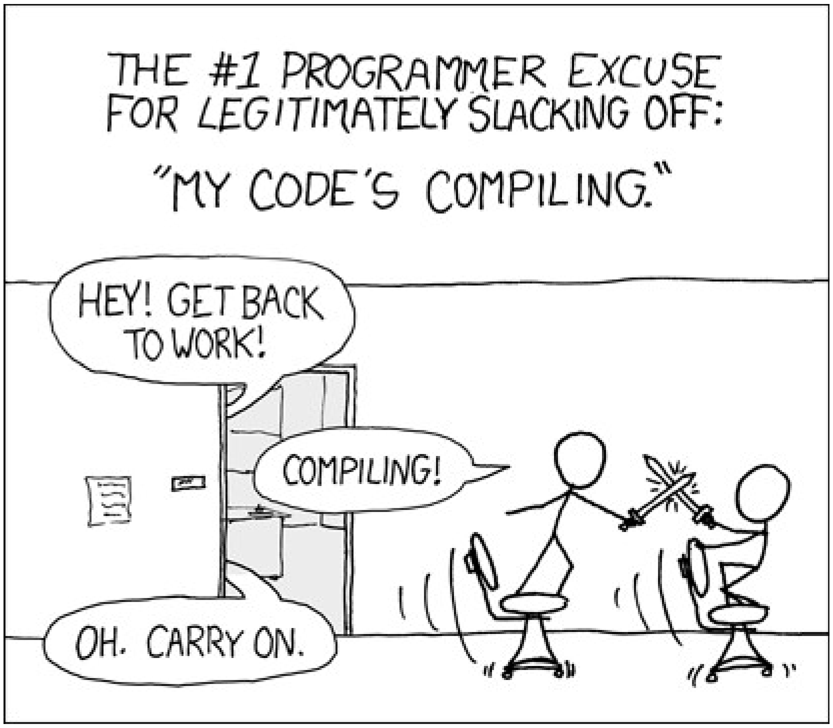
- Phases of a Compiler (synthesis)
- Finite State Machines

Today:

- Finite State Machines
    - Formal definition
    - Deterministic FSMs and non-deterministic FSMs
- Regular Expressions

Last Time

(Synthesis) Phases of a Compiler



Finite State Machines

Formal Definition

An FSM is a 5-tuple $(S, \Sigma, \delta, s_0, F)$ where

S:

Σ:

δ:

$s_0 \in S$:

$F \subseteq S$:

An FSM is defined to <u>accept</u> a string ____________________________ **iff**

Error situations:

Example: Hex Literals (cont'd)

Recall our FSM:

Formal definition:

S:

Σ:

s_0:

F:

δ: state transition table

	0	1-9	a-f	A-F	x	X	l	L
S_0								
S_1								
S_2								
S_3								
S_4								
S_e								

Coding a more generalized FSM

Using a state transition table:

This works provided the FSM is deterministic.

Deterministic and non-deterministic FSMs

Deterministic: No state has more than one outgoing edge with the same label.

Non-deterministic:

- States can have more than one edge with the same label.
- Edges may be labelled with the empty string ε.
- If a string yields any path through the machine to a final state, accept.

Example: An FSM that recognizes keywords `for`, `if`, and `int`

Example: Hex literals in Java

Why NFAs?

Compactness:

- NFA that accepts strings ending in 101:

- NFA accepting strings composed of 0s and 1s where the fifth last digit is 1.

- NFA accepting all strings over {a, b, c} that are missing at least one letter.

- NFA that accepts strings where the length is divisible by 2 or 3

Using an NFA

At each step, keep track of set of possible states, and the set of states the machine might transition to.

Example:

(Surprising?) Claim: DFAs are just as powerful as NFAs

In other words, the collection of languages DFAs can accept is the same as the one NFAs can accept.

Intuition of proof:

- Show that every NFA can be modelled by a DFA (every DFA is already an NFA).
- Do this by building a DFA that tracks the set of states the NFA could be in.
- Add an edge from a state α (α is some set of states of the original NFA) on character `c` to state β if β represents the union of states that all states in α could possible transition to on character `c`.
- Cardinality of DFA: at most $2^{|S|}$, therefore still finite. Why $2^{|S|}$?

Example: An FSM that accepts strings composed of 0s and 1s, and where the third last character is 0.

DFAs are equal in power to NFAs: Dealing with ε-transitions

ε-closures: