Should be done

Lab hours and Office hours

Sign up for the mailing list at , starting to send important info to list
– http://groups.google.com/group/cs453-spring-2011

Read Ch 1 and skim Ch 2 through 2.6, read 3.3 and 3.4

Start working with the MiniSVG start up code
– You should at least step through the existing code with a debugger before Thursday’s class

Do Soon

Quiz 1 is due tonight, was posted Tuesday night

Continue working with the MiniSVG start up code
– You should at least step through the existing code with a debugger before Thursday’s class

Send email to mstrout@cs.colostate.edu if you would like to help put together Meggy Jr devices this Sunday
– 2-3pm in rm 425
– Will only be able to accommodate the first 8 students, still 3 slots left
– Will open it up for grad students and other undergrads on Thursday
– We will have one other session sometime next week Tuesday or Thursday night

HW1 due Wednesday

Plan for Today

Interpreter and Compiler Structure, or Software Architecture

Overview of Programming Assignments
– The MiniSVG interpreter and MeggyJava compiler we will be building.

Structure of a Typical Compiler

Analysis

character stream

lexical analysis

tokens

“words”
syntactic analysis

AST

“sentences”

semantic analysis

annotated AST

Synthesis

IR code generation

IR

optimization

IR

code generation

target language

interpreter
Example MeggyJava program

```java
import meggy.Meggy;

class PA3Flower {
    public static void main(String[] whatever){
        // Upper left petal, clockwise
        Meggy.setPixel((byte)2, (byte)4, Meggy.Color.VIOLET);
        Meggy.setPixel((byte)2, (byte)1, Meggy.Color.VIOLET);
        ...
    }
}
```

Atmel assembly for Meggy.setPixel() call

```assembly
main:
    call _Z18MeggyJrSimpleSetupv
    ldi r24, 2
    push r24
    ldi r24, 4
    push r24
    ldi r22, 6
    push r22
    # Push Meggy.Color.VIOLET onto the stack.
    ldi r22, 6
    push r22
    # Pop the arguments into registers in reverse order.
    pop r20
    pop r22
    pop r24
    call _Z6DrawPxhhh
    call _Z12DisplaySlatev
```

Structure of the MeggyJava Compiler

```
Analysis
character stream
  lexical analysis
  tokens “words”
  syntactic analysis
  AST “sentences”
  AST and symbol table

Synthesis
  code gen
  Atmel assembly code
  PA2: MeggyJava and Atmel warmup
  PA3: setPixel compiler
  PA4: add control flow
  PA5: add functions
  PA6: add variables and objects
  PA7: add arrays
```

Structure of the MiniSVG Interpreter (PA1)

```
Analysis
character stream
  lexical analysis
  tokens
  syntactic analysis
  calls to report and draw routines
```

Rectangle: (0,0) 500x500 color: GREEN
Circle: (120,150) radius:60 color: WHITE
Circle: (350,150) radius:60 color: WHITE
Circle: (120,150) radius:30 color: BLUE
Circle: (350,150) radius:30 color: BLUE
Circle: (120,150) radius:10 color: BLACK
Circle: (350,150) radius:10 color: BLACK
Circle: (250,300) radius:100 color: RED
Line: (250,350) (350,350) color: BLACK
Line: (0,100) (100,100) color: BLACK
Line: (100,200) (200,200) color: BLACK
Line: (300,400) (400,400) color: BLACK
Line: (500,500) (500,500) color: BLACK
Line: (50,100) (100,100) color: BLACK
Line: (150,200) (200,200) color: BLACK
Line: (200,400) (400,400) color: BLACK
Line: (400,200) (400,200) color: BLACK
Line: (450,200) (400,200) color: BLACK
About The Slides on Languages and Finite Automata

Slides Originally Developed by Prof. Costas Busch (2004)
- Many thanks to Prof. Busch for developing the original slide set.
Adapted with permission by Prof. Dan Massey (Spring 2007)
- Subsequent modifications, many thanks to Prof. Massey for CS 301 slides
Adapted with permission by Prof. Michelle Strout (Spring 2011)
- Adapted for use in CS 453

Languages

A language is a set of strings

String: A finite sequence of letters

Examples: “cat”, “dog”, “house”, ...

Defined over a fixed alphabet:

\[ \Sigma = \{ a, b, c, \ldots, z \} \]
Empty String

A string with no letters: \( \varepsilon \)

Observations:
\[ |\varepsilon| = 0 \]
\[ \varepsilon w = w \varepsilon = w \]
\[ \varepsilon abba = abba \varepsilon = abba \]

Regular Expressions

Regular expressions describe regular languages

Example:
\((a | (b)(c))^*\)

describes the language

\[ L((a | (b)(c))^*) = \{ \varepsilon, a, bc, aa, abc, bca, \ldots \} \]

Recursive Definition for Specifying Regular Expressions

**Primitive regular expressions:** \( \emptyset, \varepsilon, \alpha \)

**Given regular expressions** \( r_1 \) and \( r_2 \)

\[
\begin{align*}
    r_1 & \mid r_2 \\
    (r_1)(r_2) & \\
    r_1^* & \\
    (r_1) & \\
\end{align*}
\]

**Are regular expressions**
Finite Automaton

Input

String

Output

String

Finite Automaton

Finite Accepter

Input

String

Output

"Accept" or "Reject"

Finite Automaton

Finite Accepter

Transition Graph

Abba - Finite Accepter

Initial Configuration

Input String

\[ a \ b \ b \ a \]

Initial Configuration

Input String

\[ a \ b \ b \ a \]

Initial Configuration

Input String

\[ a \ b \ b \ a \]
Output: "accept"

Input finished

String Rejection
Would it be possible to accept the empty string?
Another Example

\[
\begin{array}{c}
q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \\
q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2
\end{array}
\]
Input finished

```
| a | a | b |
```

Output: "accept"

```
\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \]
```

Rejection

```
| b | a | b |
```

```
\[ q_0 \xrightarrow{b} q_1 \xrightarrow{a,b} q_2 \]
```

```
\[ q_0 \xrightarrow{b} q_1 \xrightarrow{a,b} q_2 \]
```
Formalities

Deterministic Finite Accepter (DFA)

\[ M = (Q, \Sigma, \delta, q_0, F) \]

- \( Q \) : set of states
- \( \Sigma \) : input alphabet
- \( \delta \) : transition function
- \( q_0 \) : initial state
- \( F \) : set of final states

Input Alphabet

\[ \Sigma = \{a, b\} \]
Set of States \( Q \)

\[ Q = \{ q_0, q_1, q_2, q_3, q_4, q_5 \} \]

Initial State \( q_0 \)

Set of Final States \( F \)

\[ F = \{ q_4 \} \]

Transition Function \( \delta \)

\[ \delta : Q \times \Sigma \rightarrow Q \]
\[ \delta(q_0, a) = q_1 \]

\[ \delta(q_0, b) = q_5 \]

\[ \delta(q_2, b) = q_3 \]
Transition Diagrams

Definition
-A finite automata specialized for lexical analysis

Differences from finite state automata
-Finding more than one string in a single input stream
-Do not accept until hit a character with no out transition
-Asterisk notation indicates the need to put last character back in input
-Do not show sink states

Initial Configuration

Input String

Recognize token abba when attempt to process 3rd a
Demonstration of Longest Match and Priority