Shalva Kohen
Arunavha Chanda
Kai-Zhan Lee
Emma Etherington
The problem: FSMs

➢ Basis of CS and CE
➢ Current standard for representation:
  ○ Unintuitive interface
    ■ Very long descriptions
    ■ Redundant behavior commands
  ○ Learning curve from C-like languages
    ■ Syntax
    ■ Style

“The less intelligent things you have to do, the more stupid things you have to do.”
The solution: FSMs!

➢ Our solution:
  ○ Language derived from OOP languages to describe and simulate FSMs
  ○ Duality:
    ■ Offers user-friendly interface for constructing FSMs
    ■ Retains imperative nature of OOP languages

X: “Did you just change everything?”
Y: (Calmly) “Yeah.”
Cool Things

➢ “Tick function” as clock
➢ Reset function
➢ State boundaries
➢ User-friendly program structure relating to FSM diagrams
➢ Automatic generation of header files!
➢ Concurrent execution of FSMs

“But clocks tick. Clocks don’t clock!”
Features of Language

- Input and output lists and types
- Public variables: Read-global, write-local
- User-defined types
- Most intuitive features of both automata and C programming

“So two things. First thing is it might work if I make this an unsigned int. Can I make this an unsigned int?”
“Sure. Go ahead.”
“Right. So the second thing is I don’t know how to make this an unsigned int.”
“I’m totally open to new ideas. I just don’t think this one in particular works.”
“Wait, so you’re saying [the] entire parser is a piece of crap?”
LLVM Generation: Tick

In FSM, branch to each state through switch

“Those weird little badooshkins...”
Test Suite

➢ Uses shell scripts similar to those of MicroC
➢ 3 Scripts
  ○ testall.sh
  ○ traffic.sh
  ○ adventure.sh
➢ Automatic generation of C wrappers
➢ 56 test cases
  ○ 34 positive tests
  ○ 22 negative tests
➢ Adventure Program

“OCaml is a weird language. But I am also weird, so it is a good match.”
Uses and Future Steps

➢ Applications
  ○ Testing state reachability
  ○ Simple Concurrent FSM execution
  ○ Master-Slave Concurrency Problems
  ○ Testing algorithmic state machines

➢ Future steps
  ○ Implementing Mealy machines and DFAs and NFAs
  ○ State minimization

“We do the thing, then the thing, and then a thing thing. Wait, there’s another thing.”
Lessons Learned

➢ Communicate
  ○ Know what everyone is doing
  ○ Make sure they are doing it per group specifications

➢ Plan
  ○ Think more about what the program will need before coding anything
  ○ Set an end goal for everyone to work towards

➢ Set Realistic Goals
  ○ Know the time constraints of each group member

➢ Working on the same platform

“We just made progress”
“We didn’t. The net movement has been very minimal”
DEMO TIME!!!