Shalva Kohen
Arunavha Chanda
Kai-Zhan Lee
Emma Etherington

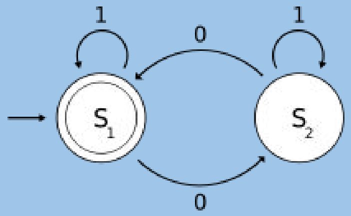
The problem: FSMs

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Test_Counter_VHDL is
    Port ( Clk_xxxHz :           in  std_logic;
           Step_Clk :            in  std_logic;
           Select_Clk :          in  std_logic;
           Clr, Count_Enable :   in  std_logic;
           Bcd0,Bcd1,Bcd2,Bcd3 : out std_logic_vector(3 downto 0));
end Test_Counter_VHDL;

architecture Behavioral of Test_Counter_VHDL is
   Signal Q:   std_logic_vector( 15 downto 0);
   Signal Clk: std_logic;
begin
   -- 2x1bit multiplexer: Clk_xxx or Step_Clk = [Btn0]
   Clk <= Clk_xxxHz when Select_Clk='1' else
          Step_Clk;

   process( Clk, Clr)
   begin
      if Clr='1' then
         Q <= (others => '0');     -- "0000000000000000"
      elsif rising_edge( Clk) then
         if Count_Enable='1' then
            Q <= Q+1;
         end if;
      end if;
   end process;

   Bcd3 <= Q(15 downto 12);
   Bcd2 <= Q(11 downto  8);
   Bcd1 <= Q( 7 downto  4);
   Bcd0 <= Q( 3 downto  0);

end Behavioral;
```

- 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

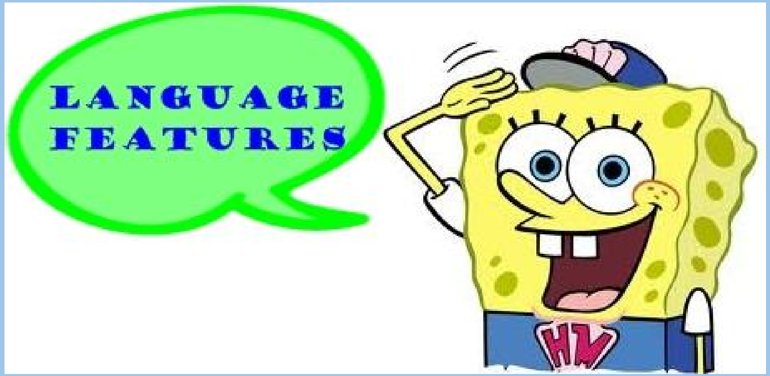
X: "Did you just change everything?"
Y: (Calmly) "Yeah."

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!"

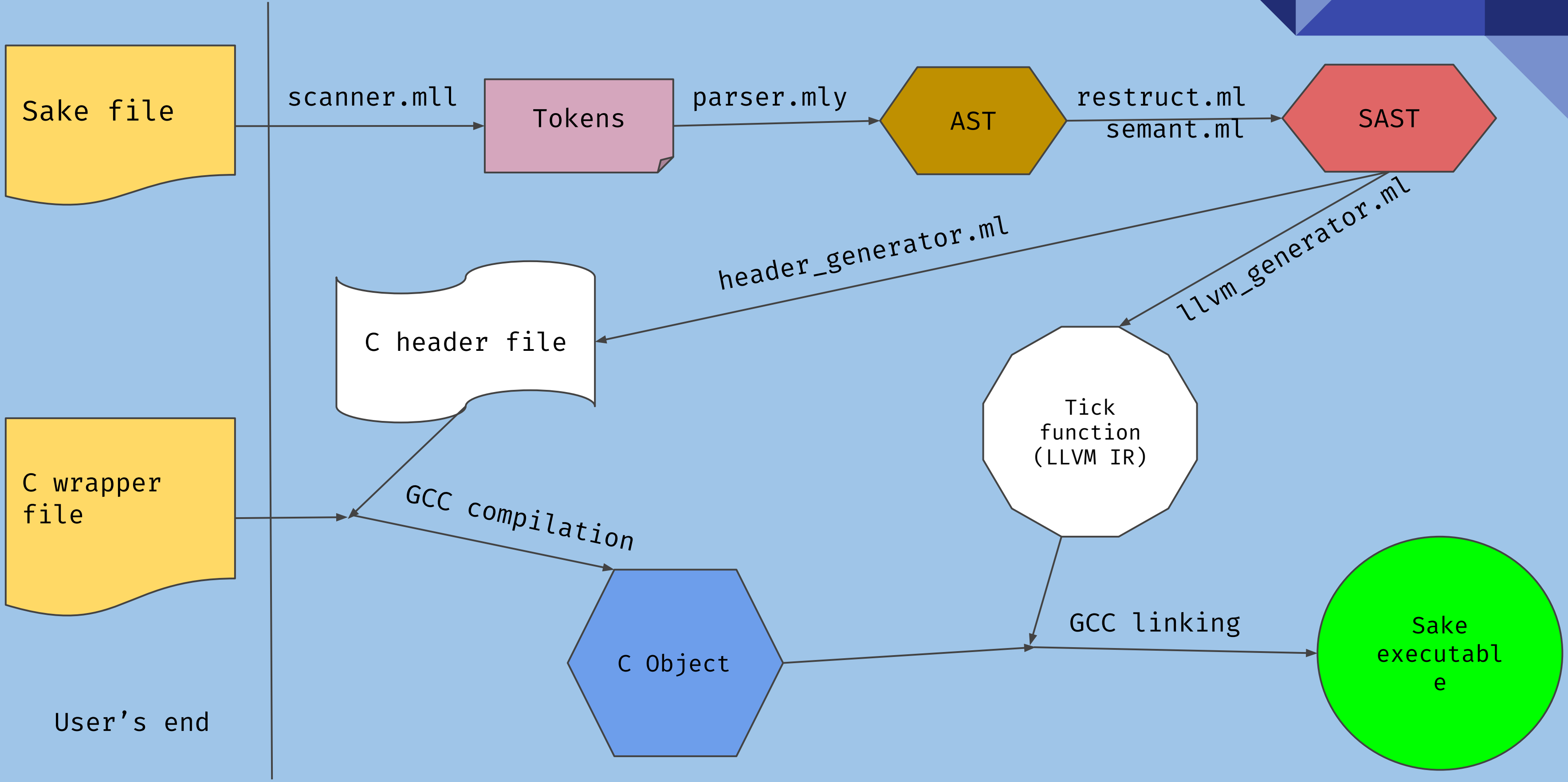
Features of Language



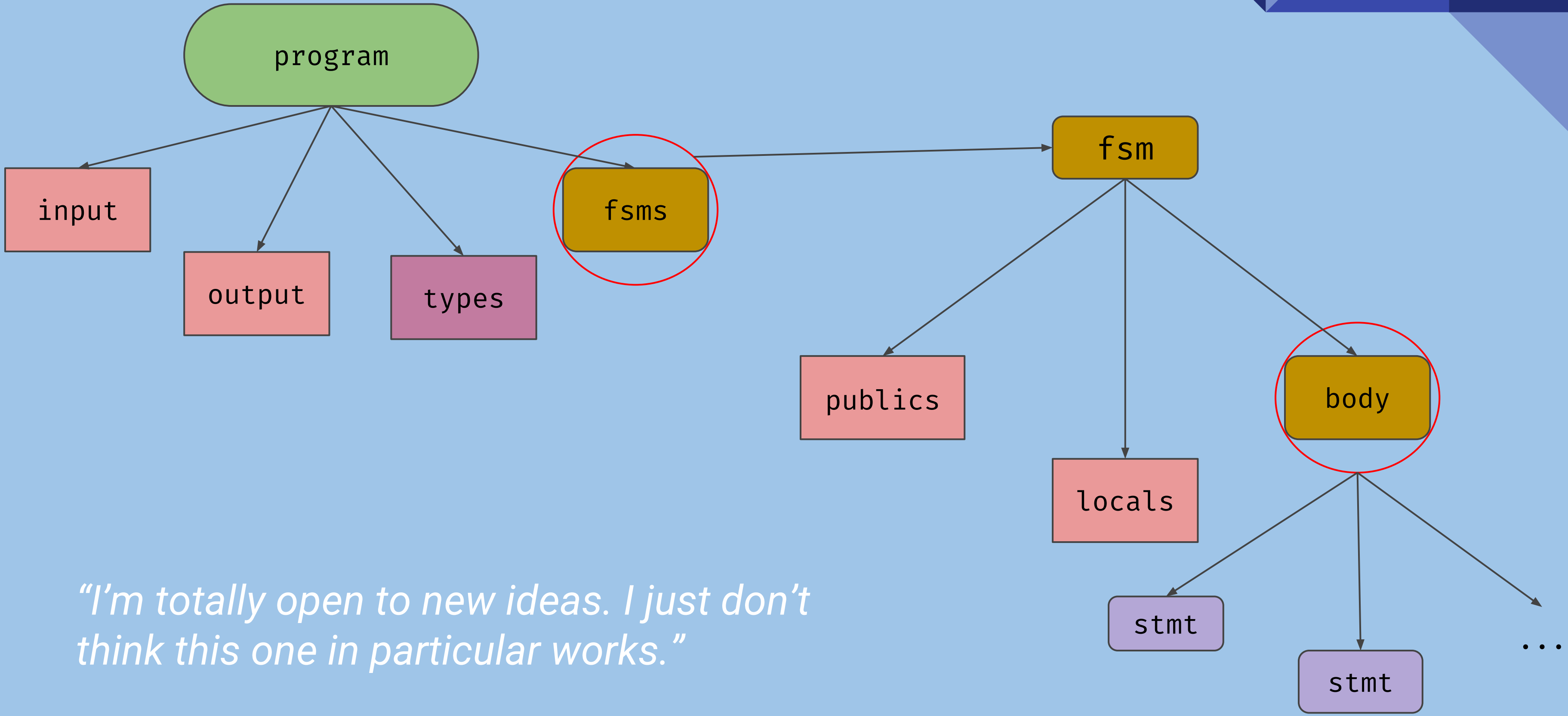
- Input and output lists and types
- Public variables: Read-global, write-local
- User-defined types
- Most intuitive features of both automata and C programming

"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."

System Architecture



Abstract Syntax Tree (AST)



“I’m totally open to new ideas. I just don’t think this one in particular works.”

Parser

```
$ make
ocamlyacc parser.mly
41 rules never reduced
216 shift/reduce conflicts, 460 reduce/reduce conflicts.
ocamlc -c parser.mli
ocamlc -c parser.ml
File "parser.mly", line 64, characters 31-35:
Error: Unbound value call
make: *** [Makefile:13: parser.cmo] Error 2
```

“Wait, so you’re saying [the] entire parser is a piece of crap?”

LLVM Generation: Tick

```
define { i32, i32 }* @test_halt_tick({ i32, i32 }*, { i32, i8* }*, { i32 }*)
```

```
entry:
  %null = icmp eq { i32, i8* }* %1, null
  br i1 %null, label %reset, label %check
```

```
reset:          ; preds = %entry
  %reset1 = getelementptr ...
  store i32 1, i32* %reset1
  %reset2 = getelementptr ...
  store i32 0, i32* %reset2
  ret { i32, i32 }* null
```

```
check:          ; preds = %entry
  %ptr = getelementptr ...
  %running = load i32, i32* %ptr
  %run = icmp ne i32 %running, 0
  br i1 %run, label %update, label %halted
```

```
halted: ; preds = %check
  ret { i32, i32 }* null
```

```
update:         ; preds = %check
  %state = alloca { i32, i32 }
  call void @memcpy(...)  ; copy to state
  call void @halting(...) ; FSM update call
  call void @memcpy(...)  ; copy from state
  ret { i32, i32 }* %0
```

- ➢ Two initial checks
  - ○ Reset: reset all values
  - ○ Halted: return 0
- ➢ Allocate memory, update states

```
define void @halting({ i32, i32 }*, { i32, i32 }*, { i32, i8* }*, { i32 }*)
```

```
entry:
  %halting = getelementptr ...
  %halting1 = load i32, i32* %halting
  switch i32 %halting1, label %"*halt" [
    i32 0, label %"*init"
    i32 1, label %One
    i32 2, label %Two
  ]
```

```
"*halt":  ; preds = %entry
  %ptr = getelementptr ...
  store i32 0, i32* %ptr
  ret void
```

```
"*init": ; preds = %entry
  br label %One
```

In FSM, branch to each state through switch

```
Two:                        ; preds = %entry
  %p9 = getelementptr ...
  %p10 = load i32, i32* %p9
  %tmp11 = icmp eq i32 %p10, 1
  br i1 %tmp11, label %then7, label %else8
  [... other blocks]
  ret void
```

```
One:         ; preds = %"*init", %entry
  %p = getelementptr ...
  %p2 = load i32, i32* %p
  %tmp = icmp eq i32 %p2, 1
  br i1 %tmp, label %then, label %else
  [... other blocks]
  ret void
```

"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

```
plt@ubuntu-plt:~/sake/ocaml$ ./traffic.sh
test_brokenTL...
TL 1: g         TL 2: g
TL 1: g         TL 2: g
TL 1: g         TL 2: g
TL 1: g         TL 2: y
TL 1: y         TL 2: r
TL 1: r         TL 2: g
TL 1: g         TL 2: g
TL 1: g         TL 2: y
TL 1: y         TL 2: r
TL 1: r         TL 2: g
OK
```

"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!!!