Saké final project report

Shalva Kohen: sak2232 (Language Guru)
Arunavha Chanda: ac3806 (Manager)
Kai-Zhan Lee: kl2792 (System Architect)
Emma Etherington: ele2116 (Tester)

TABLE OF CONTENTS

1. Introduction:

Behind all models of computation and computer science lies the concept of automata, or Finite State Machines (FSMs). The current standard for describing, simulating, and testing FSMs is set by hardware description languages such as VHDL and Verilog. However, there are many problems with these languages: they are clunky, contain redundancies, and are difficult to learn. The learning curve tends to be especially steep for programmers familiar with C-like languages due to differences in syntax, style, and program flow.

That is where we decided to step in and build our language designed for FSMs. SAKÉ, named after the four creators of the language, is a simple language designed to describe and simulate both individual and concurrent Moore finite state machines (FSM). The syntax closely follows that of languages like C++, Java, and Python, but the structure itself is similar to FSM designs with input and output variables, user-defined types and FSM blocks. In each FSM block, the user has the liberty of defining states and transitions, having a state list that is executed imperatively, or both! In this way, our language combines the best and most intuitive features of object-oriented programming languages and FSM design.

2. Language Tutorial:

Compilation Instructions

To compile and run programs first you must run `make`:

```
make
ocamlbuild -clean
Finished, 0 targets (0 cached) in 00:00:00.
00:00:00 0    (0   ) STARTING
-------- |rm -rf *.cmx *.cmo *.s *.ll *.out *.exe *.o *.diff *.err *.gch ../testing/*.h
../testing/*.gch *.h
rm -rf sake parser.ml parser.mli scanner.ml *.cmo *.cmi *.output alltests.log trafficLights.log
adventureStory.log
ocamlbuild -use-ocamlfind -pkgs llvm,llvm.analysis -cflags -w,+a-4-40..42 sake.native
+ /usr/bin/ocamlyacc parser.mly
Finished, 31 targets (0 cached) in 00:00:01.
cc    -c -o print.o print.c
```

This builds and compiles the source files and creates the executable `sake.native`, which is a symbolic link to the `sake.native` executable file which is stored in `sake/ocaml/_build` directory. All intermediary files that are created by `make` are stored in this directory.

Once you have `sake.native`, there are a couple ways to run programs.

To run all test programs, except traffic light FSM programs, in the testing directory, you can run the `./testall.sh` script.

This script takes each test file and runs the following commands to compile and run each SAKÉ program.

```
./sake.native <test_program_name>.sk <test_program_name>
mv <test_program_name>.h ../testing/
llc <test_program_name>.ll > <test_program_name>.s
gcc -c ../testing/<test_program_name>.c ../testing/<test_program_name>.h
gcc -o <test_program_name>.exe  <test_program_name>.s
       <test_program_name>.o print.o
./<test_program_name>.exe  >   <test_program_name>.out
```

The first line, `./sake.native <test_program_name>.sk <test_program_name>`, compiles the SAKÉ program and, with the help of `llvm_generator (.ml)` and `header_generator (.ml)`, creates an LLVM file (.ll) and a C header file (.h). The header file is then moved to the testing directory and use for the compilation of the C wrapper file. The LLVM file is compiled using `llc` to produce assembly code for the current system architecture. The last step of the process to create the executable is to link the assembly code with the C object and also print.o, a C object file created by main containing the implementation of the printf function that is used in SAKÉ code. Once the executable is formed it can be run. The script will redirect the output to a out file that can then be used for comparison with expected output. If you simply want output to be sent to standard out, remove the > redirection. The test script will also remove all intermediary files for each test that is successful. if you chose to run compilation manually, running `make clean` will remove this intermediary created files from the directory.

These steps can all be individually run on the command line to the same effect. The test script also allows for easy automation (discussed in Section 6: Test Plan). All testing files are kept in the directory sake/testing. You must run `make` and the command line arguments from `sake/ocaml`.

Creating a Sake program

The user first creates a .sk file according the rules defined in the LRM. Creation proceeds as follows:

1. Indicate what input(s) the FSM will accept and what output(s) it will spit out.
2. Optional: declare custom types.
3. Declare FSM(s), taking the following steps for each:
    a. Declare public variables, which are to be saved between ticks.
    b. Declare local variables, to be reinitialized at the beginning of each tick.
    c. Write statements to be executed on each tick.

For example…

myadd.sk:

```
	~ step 1: input and output
	input[int i, int max]
	output[int o]

	~ step 2: a user-defined type, which can take the values Sake, Is, or Awesome.
	type unusedType = Sake | Is | Awesome

	~ step 3: an FSM! Note that multiple FSMs may be declared in succession.
	fsm addAndSub {
		~ step 3a: public variable(s)
		~ Variables are initialized to 0 unless otherwise specified
		public int curr

		~ step 3b: an unused local variable, set to 42 at the start of every tick
		int life = 42

		~ step 3c: state declaration
		state Add
			/~ In this scope, `addAndSub.curr` is equivalent to `curr`.
	However,
			 ~ this example demonstrates how one might access public variables
			 ~ from other FSMs.
			 ~/
			o = curr = addAndSub.curr + i
			if (curr + i >= max) {
				goto Sub  ~ transition
			}
			goto Add
		state Sub
			o = curr = curr - i
			halt
	}
```

The above program is a simple adder and subtractor. The user defines an input vector and an output above. In the FSM, the input's goes through its multiples in the `Add` state until the output goes past "max". Note that public variables don't update their values until the next clock cycle, which is why we still have to test with (curr + i >= max). This is to remain consistent with general FSM behavior. Then the program transitions to the Sub state where the input is subtracted from the output one final time before the fsm halts. This toy example finds the largest multiple of "i" less than "max".

To run this program, the user writes a basic C wrapper file to call the tick function. First, as initialization, the C wrapper must create the input, output, and state structs and call the tick function once, with input and output of `NULL`, to dynamically initialize and reset the FSM in memory. To use the tick function, the user only has to set input as desired to call tick again.

`myadd.c`:

```
#include <stdio.h>
#include "myadd.h"

int main() {
	struct myadd_input input;
	struct myadd_output output;
	struct myadd_state states;

	// The FSM must be reset before anything else
	myadd_tick(&states, NULL, NULL);

	// Set input and tick
	input.i = 8;
	input.max = 100;
	myadd_tick(&states, &input, &output);


	// Access output
	printf("The first iteration of my FSM gave me: %d\n", output.o);
}
```

The above example will print out 8. Why? Because tick is only called once, meaning the FSM will make one iteration and then stop. That one iteration is when the input i is added to the output o. If we were to call tick again, then the program would keep going through multiples, looping on the Add state. On crossing 100, the program would enter state Sub, subtract 8 from 104, then halt. Our output then would 96. If the user wanted to keep the FSM running continuously she could just put the tick function in a while loop.

3. Language Reference Manual:

Language Basics:

Types:

The user will be able to use certain primitive types to define their variables, inputs, and outputs of the fsm. These types are: int, char, string, bool

As indicated, an int will be 4 bytes long and represents a non-decimal number. A char is a 1-byte character defined by ASCII. A string is an immutable array of chars with an arbitrary length. A bool is a logical true or false value. The user will also be able to specify their own types using the type keyword. Finally, the user will be able to represent multiple elements of any type by using a list.

identifiers, Declaration, and Casting:

Variable identifiers, or names, begin with a lowercase letter; the characters afterwards may consist of uppercase or lowercase letters, and digits. A formal regex for a variable name is the following:

```
[a-z][a-zA-Z0-9]*
```

Enum value identifiers, which will be discussed in greater depth, begin with a capital letter; the characters after follow the same patterns as in variable names:

```
[A-Z][a-zA-Z0-9]*
```

Variables are explicitly typed; there are primitive types, and users may declare custom types as well. Variables are set to zero by default if an initial value is not specified:

```
<type> <variable_name> = <expression>
<type> <variable_name>
```

Literals:

Type	Regex (extended)	Description
bool	`true|false`	A bool literal is represented using the values "true" and "false". `true || false`
char	`' . '`	A character is 1-byte long value representing an ASCII symbol. `'a'`
int	`[0-9]+`	Int literals are notated in base 10 by a series of digits. Declarable values have a lower bound of -2^31 and an upper bound of 2^31. Int literals are always positive and can be negated by a unary negative sign. `0 00 01 02 3562`
string	`" .*"`	A string literal represents a contiguous set of bytes containing the values specified in the string itself. Standard escape characters may be used in string literals. `"Hello World!\n"`

Type Definition:

Custom types can be defined with the `type` keyword. Type identifiers are identical to variable identifiers. Their values may consist either of enum values or of expressions that all evaluate to a single type.

```
type <typename> = <enum> | <enum> …
```

```
type Color = Red | Green | Blue | Yellow | Magenta
```

Operators:

Operator	Description
=	Used to set the value of a variable. `<variable> = <value>`

+, -, *, /	Arithmetic operators for adding any of the primitive types, or any of the custom types based on the primitive types. In addition, + and - are unary operators that respectively do nothing to and negate their operands.
	
	`1 + 2`
	`- 3`

|	Or operators that is used to create groups of types and states. `type <new_type> = <value1> | <value2> | ...`

[ ]	Used to specify input and output variables in FSM definitions:
	
	`input [<type> <variable name>, ...]`
	`output [<type> <variable name>, ...]`

&&, ||, !	Boolean operators for evaluating boolean expressions; these symbols represent the and, or, not operators, respectively. These expressions evaluate to either 0 or 1. Boolean arithmetic with these operators can only be performed on integers. `true || false && !(3 < 2)`

<, <=, ==, !=, >, >==>	Relational and equality comparisons, which return integer values. `1 != 2 && 2 == 3`

Keywords:

Keyword	Description
fsm	The user must specify the fsm keyword to indicate the start of a FSM declaration. `fsm <fsm_name> { /~ FSM Specifications ~/ }`
state	The user must specify the state keyword within the fsm declaration to indicate a state boundary. A state boundary marks the start of a state, and therefore both a potential entry point of and a necessary exit point of a tick function call `state <state_name>`
input	The user must specify the form of the input stream to be interpreted by the fsm by using the input keyword followed by the input form in square brackets. `input [<type> <variable_name>, ...]`
output	The user must specify the form of the output of the fsm by using the output keyword followed by the input form in square brackets. `output [<type> <variable_name>, ...]`
type	The user can define new variable types or reuse old ones, as described above, by using the type keyword. The type specification is delineated with parallel bars. `type Color = Red | Green | Blue | Yellow`

if, else	The keys words if, and else can be used to create ladder statements that can be used to specify blocks of code that should only be evaluated if a condition evaluates to true.
while	The while keyword can be used to specify a loop that should repeatedly execute a block of code while the condition evaluates to true.
for	The for keyword can be used to specify a loop that should execute a finite number of times. It may only be used to execute code a set number of times.
public	The public keyword indicates that a variable defined in an fsm can be read but not written to in other fsms. It can only be written to within the fsm that it was defined.
halt	The halt keyword tells the fsm to stop running, immediately returning `NULL` on all future calls to the tick function with the current FSM.

switch, case	The switch keyword is used within the body of an FSM specification. The switch keyword starts the transition specification. Within the specification the case keyword is used to specify actions based on specific values. See control flow section for a more detailed explanation of switch, case.
	
	`switch(<input>,<input>, ...) {`
	`  case <value>, <value>... : <statement>`
	`}`

goto	The goto keyword is used within the state transition specification to specify which state to transition to. `goto state_destination`
printf("<string>", expression, ...)	Calls the C library printf function with the given arguments. The first argument must be a string literal.

Wrapper Function:

Every time the tick function is called, the FSM will iterate once. Its arguments are the struct of states the FSM is made of, the input to the fsm, and the output to the fsm. For a filename of `f.sk`, the generated function prototype in the header file would be:

```
struct f_state *f_tick(struct f_state *, struct f_input *, struct f_output *);
```

Comments:

```
~ This is a line comment
/~ This is a block comment ~/
```

Control flow:

If-else Statements:

If-else blocks can be used to check conditions before evaluating blocks of code. The code block under the if statement will only be executed if the condition evaluates to true. If there is an else statement, it will execute is none of the if statements evaluates to true. Else statements are not required after if statements.

```
if condition {
	/~ code block ~/
}

if condition {
	/~ code block ~/
} else {
	/~ code block ~/
}
```

While Loop:

A while loop can be used to repeatedly execute a block of code while the condition evaluates to true. If the condition evaluates to false the first time the while loop is met, then the block of code will never be executed.

```
while condition {
	/~ code block ~/
}
```

For Loop:
A for loop can be used to execute a block of code a finite number of times. A for loop can be applied to iterate through a set of states, or to execute from a start value to an end value with a set interval.

```
/~ start, end, step are integral ~/
for s in start:end:step {
	/~ code block ~/
}
```

Switch case:
A user can define a switch case statement. This is very useful when defining transitions to different states. The user puts the value they wish to switch on in the parenthesis and then defines a list of cases when the input corresponds to a certain value. Within a case, the user can define a statement, or a list of statements, they want to perform based on the certain value.

```
switch (<input>, <input>, ...) {
	case <value>, ... : <statement>
	                    <statement>
	case <value>, ... : <statement>

}
```

Goto:
If the user wants to transition to a certain state, they can use the goto keyword to do so. By writing

```
	goto <state_name>
```

Upon a goto statement, the tick function will set the current state appropriately and return, leaving the user with the set outputs.

FSM Specific Rules:

Structure of Language:

There are several components to the Sake language. The first is the Sake file itself. This file dictates the behavior of the fsms we wish to create. Here, states are defined and the fsms are created. A C wrapper is also used. Within the wrapper, the user runs the fsm. A header file is automatically created that the C file uses. In the C file, there is a special function called <fsm_name>_tick. Every time tick is called the fsm makes one iteration given an input that the user gives.

Structure of Sake file:

Our language is designed so the user can create both regular and concurrent FSMs. The following structure is meant to be done in the sake file. Each point will be elaborated on in the following sections.

1. Input/Output Declaration
2. (Optional) The user defines any types they think they might want to use
3. FSM declaration
    a. Public variable declaration
    b. Local variable declaration
    c. Statements: Within statements the user can define state boundaries, control flow, and anything else pertaining to the function of the fsm

Input/Output Declaration:

To specify the type and quantity of inputs within the FSM, the user should use the input keyword and place in brackets and type and name of their input. If they wish to have more than one input they can do so using a comma separated list. The same thing is done for output declaration. However, instead of using the keyword input they use output. This is an example of a basic input and output statement:

```
input [int i , bool b]
output [String s]
```

Type Declaration:

The user can define their own types. These act like enums in other languages. They are defined by using a bar-separated list.

```
type <type_name> = Type1 | Type2 | Type3 | …
```

FSM Declaration:

The user needs to specify the FSM they wish to create using the fsm keyword. Everything within the {} will pertain to the FSM.

```
fsm <fsm_name> {
        /~
         ~ Specification of states, and
        ~ other fsm properties
        ~/
}
```

Public and local variables

The user must first declare their public and local variables before defining any FSM behaviors. As mentioned above, a public variable is one that other fsms can see and access whereas a local variable is specific to the FSM in which it was defined in.

```
public int i = 0
public char c
string loc = “this is a local variable!”
```

States and statements:

An FSM needs states, and luckily they are very easy to use in Sake! After the local and public variables are defined, the user can define their states and define the behavior in a state using the control flow statements defined in the section above. To create a state, use the keyword state followed by a name you want it to be.

```
state MyState
```

Now every statement declared afterwards will be in MyState. To move to another state the user can write a goto statement, or if they write another state definition after the current one the fsm will automatically transition to the new state. One thing to note: there is no start state declaration. This is because the first state defined is automatically the start state.

Concurrency:

The user is also able to define concurrent FSMs. If a user has multiple FSMs then the user can access the state of an FSM by using the specific FSM name. This way, decisions in an FSM can be made based on the states of other FSMs.

```
fsm fsmOne {
        int i = 0

        state One
        if (fsmTwo==Two && i == 0){
                <more statements…>
        }
}
fsm fsmTwo {
        <variable definitions…>

        state Two
                goto Three
        state Three
                halt
}
```

The C Wrapper:

This is where the user simulates the FSM! When the user creates a Sake file, a header file is automatically created. Within the C file, the program resets the FSM once then calls tick once using the structs defined in the header file. This means that if the user does not define an special behavior, the fsm will iterate once on no input.

Defining input and output

The header file creates structs for the input, output, and current state of the FSM. The arguments of the tick function are the pointers to these structs. Every time tick is called these structs are updated to reflect the new output and state. To simulate tick the user needs to define the input. This is done by taking the input struct and accessing the variable(s) that the user declared as input in the Sake file.

myfsm.sk:

```
input[int i]
output[int o]
<other sake code…>
```

myfsm.c:

```
#include <stdio.h>
#include "myfsm.h"
```

```
int main() {
	struct myfsm_input input;
	struct myfsm_output output;
	struct myfsm_state states;

	myfsm_tick(&states, NULL, NULL); // reset

	// define input and access output
	input.i = 0
	myfsm_tick(&states, &input, &output);
	int theOutput = output.o;
}
```

Syntax:

Order

The Sake language is parsed by new lines. This means that line breaks are very important in this language. The order of statements is also very important, and the user must follow the order defined above in the structure of an FSM. Here is what the file should look like along with new lines:

```
<input declaration>
<output declaration>
Newline
<type declaration> (optional)
Newline
<FSM declaration>
<public variables> (optional)
<local variables> (optional)
Newline
<statements>
<closing bracket of FSM>
<new FSM declaration…>
```

Variable naming

States, type names, and types begin with an uppercase letter. Variable names begin lowercase.

4. Project Plan:

4.1 Process

4.1.1 PLANNING AND SPECIFICATION

We met twice weekly and had a very active group messaging system as well. One of the tenets of our group was constant communication. If any decision was to be made, everyone had to be informed: even if it slowed things down a little, we decided this had to be done.

Our work took place in phases with each phase having a defined goal. For the first 2 weeks, it was to define the functionality of our language: to decide on what we want our language to be able to do and achieve.

Next, we spent 2 weeks hammering out the specific details of our language: Building the Abstract Syntax Tree, defining the syntactic details that we want and some of the semantic behavior. All of this went into constructing our Language Reference Manual. Our language did end up looking a bit different from the original Manual, though this was because we kept realizing new features, redundancies and better ideas along the way, and the language design got improved and transformed multiple times.

The next month was spent in developing a minimum viable product that would have one FSM capable of doing arithmetic and printing to the screen. Once we had that, the last month-and-a-half was devoted to creating the code generator, adding test cases, implementing semantic checking and adjusting everything along the way when we wanted new features added.

4.1.2 DEVELOPMENT AND TESTING

The development and testing was very integrated. In the time from the "Hello World" demo to the finished product, the code generator and tester worked almost in tandem: The testing suite would have a new file added that would test one feature. That would be the goal of the code generator. Once we would have the back end able to implement the feature, we would check if the test passed. Once it did, that test would get added to the final test suite. This way, we did both unit testing and integration testing.

Once semantic checking started being implemented, the same process happened, where we kept adding tests to test the checker's ability to deal with errors and exceptions until it could pass them all.

4.2 Style Guide

- Use 1-space padding binary operators (e.g. `x +. y`, `let x = 3`, `x ** 32`)
- Don't put spaces around unary operators (e.g. `x--`, `x[0]`, `*x`)

- Comments are written as follows: `(* your comment here *)`
- Tabs, not spaces
- Declaratory terms (`let ... =, try ...`) require indentation following until termination
- Terminating terms (`in, with`) are not preceded by a newline
- All cases of `match` and `function` expressions are indented and preceded by |
- Line break and explanatory comments after every major step of code

4.3 Project Timeline

Date	Stage achieved
January 25, 2017	First meeting and role assignments
January 27, 2017	Language basics and goal decided
February 8, 2017	Language Proposal made
February 19, 2017	Idea of tick function born
February 22, 2017	First Language Reference Manual created
March 2, 2017	AST written
March 20, 2017	Parser and scanner finalized (for then)
March 28, 2017	Full stack working for Hello World demo
April 6, 2017	Header generator creates working headers
April 10, 2017	First set of the test suite committed
May 4, 2017	LLVM generator, semantic checker, test suite completed
May 9, 2017	Final report completed

4.4 Roles and Responsibilities

Member	Responsibilities
Emma Etherington	**In charge of:** ● Handling regression test suite

	• Creating new tests • Header generation • C Wrappers
Shalva Kohen	**In charge of:** • Parser • Language design decisions • Language Reference Manual **Helped with:** • AST/SAST • LLVM generation • Scanner • Test suite
Arunavha Chanda	**In charge of:** • Team decisions and communication • Semantic checker • Scanner • SAST • Finalizing report and presentation **Helped with:** • AST
Kai-Zhan Lee	**In charge of:** • System architecture and design • AST • LLVM generation • Header generation • Code listing generation **Helped with:** • Finalizing report • Test suite • C Wrappers

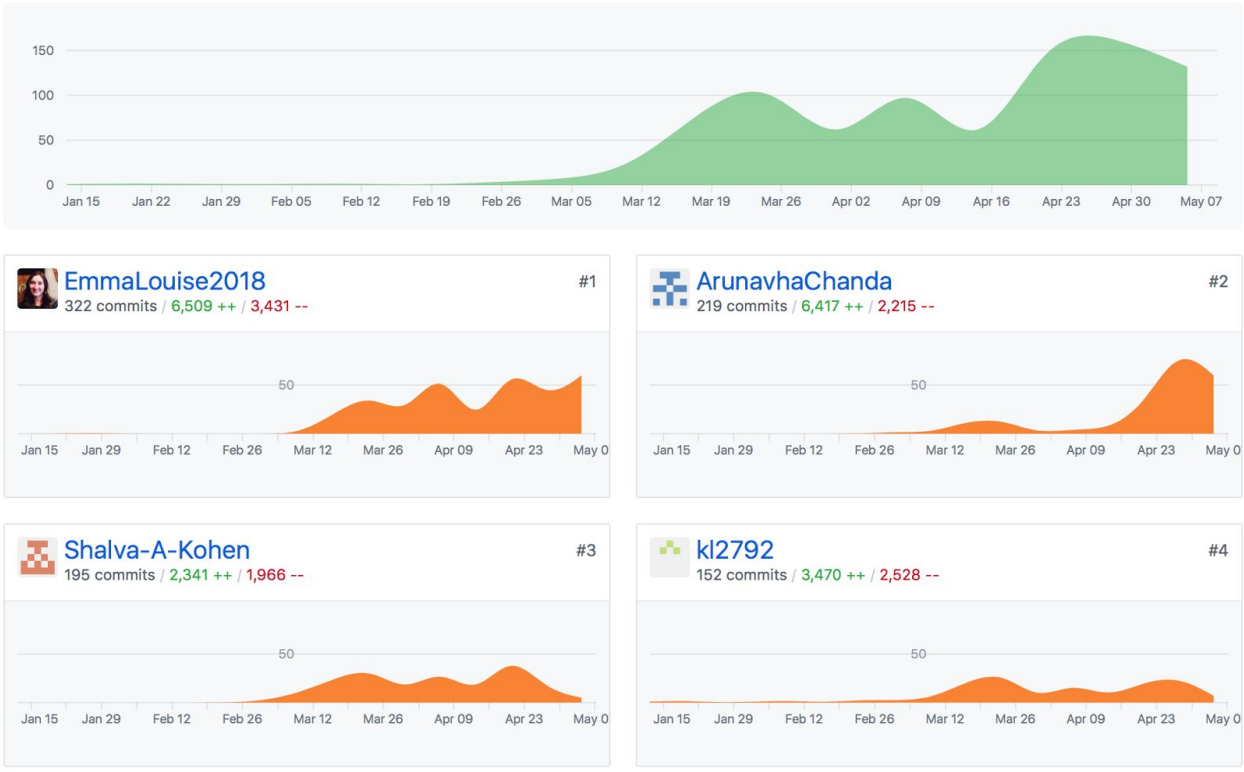
4.5 Software Development Environment

We used the following software while developing Saké:

1. **Ubuntu 15.10:** The operating system used for development
2. **OCaml 4.01.0:** Used to build compiler. Ocamllex and Ocamlyacc additionally used for scanner and parser.
3. **LLVM 3.6:** Used for code generation backend.
4. **Sublime Text 2, Vim:** Used as text editors for programming
5. **GCC 5.2.1:** Used to compile C wrapper files in project.
6. **GitHub:** Used as version-control system

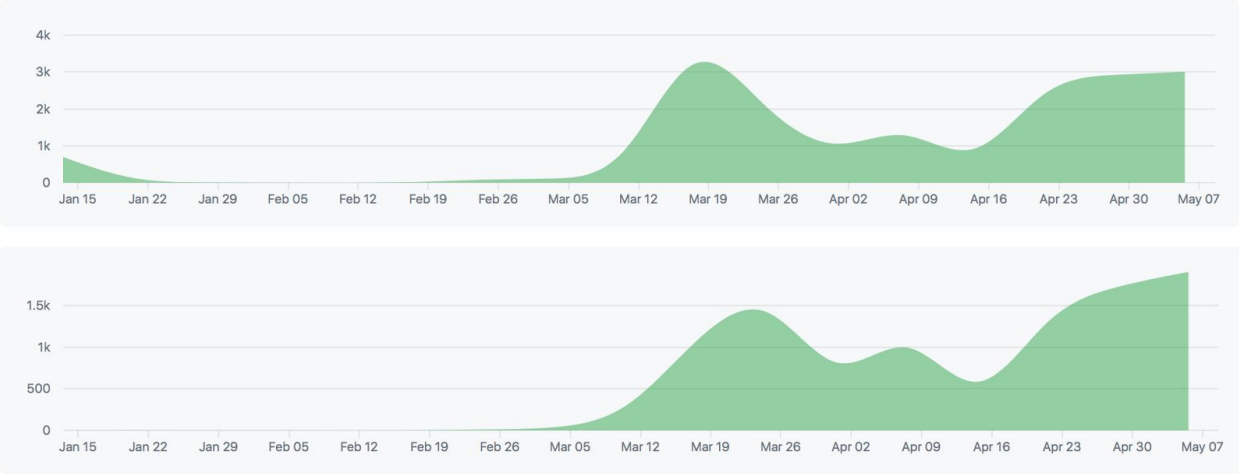
4.6 Project Log

Smoothed Commit Frequency (1074 commits total)

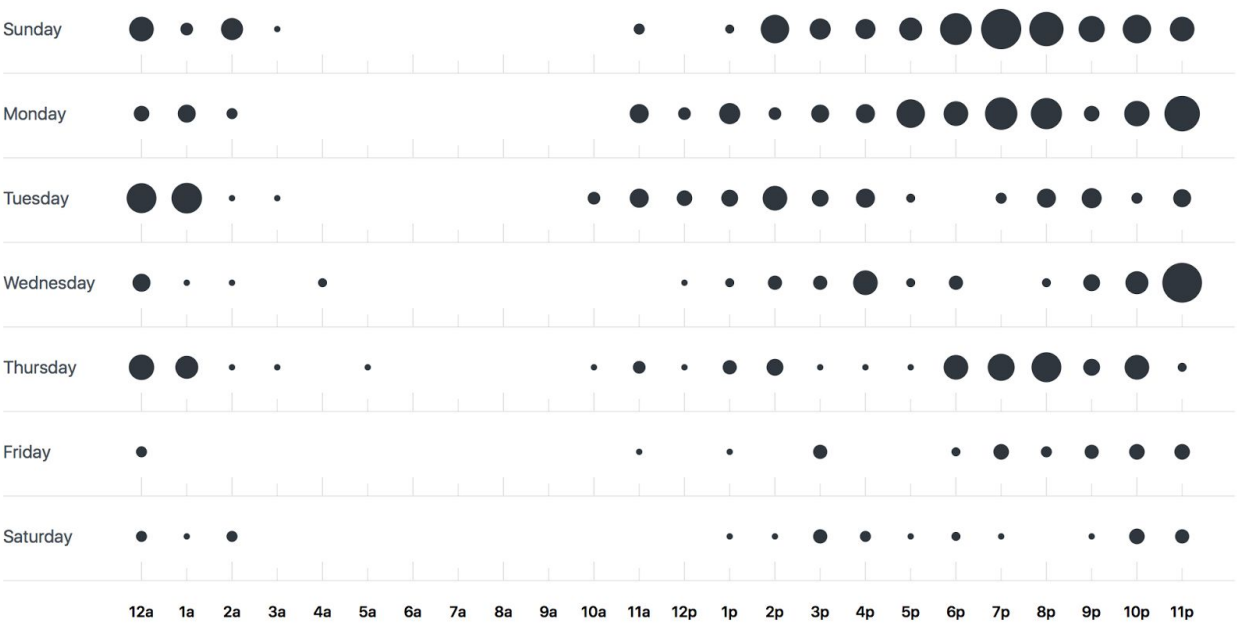


Everyone in the group contributed substantially to the project in terms of lines of code and ideas, as evidenced by statistics. (Note: Number of commits, additions, and deletions are not reflective of participation or effort, because both teammates' commit rates and the amount of code and effort required for each task varied widely).

Smoothed Weekly Addition and Deletion Frequency

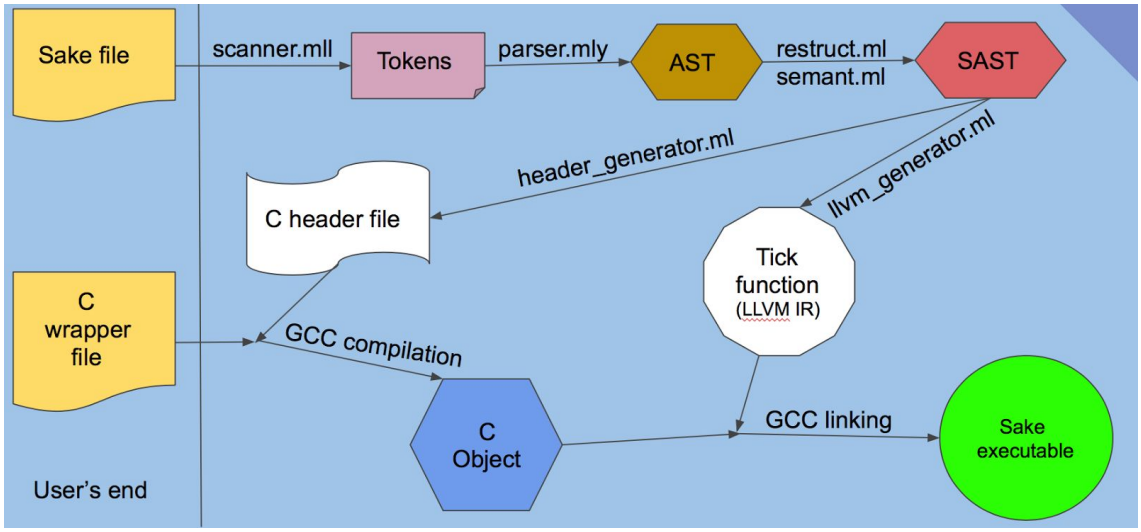


Group Punch Card



5. Architectural Design:

5.1 Design Diagram



5.2 Interfaces between components

As is typical, Saké code is tokenized, parsed into an abstract syntax tree, semantically checked, and converted into a semantically-checked abstract syntax tree. However, because our language only describes a function for executing an FSM, we generate both LLVM IR for executing said function and a header file for use of the function in a C wrapper. Compiling our LLVM IR and the C wrapper with the header file, we link the resulting object files and thus create our final executable.

5.3 Who Implemented what?

scanner.mll: Arunavha (with Shalva)	**parser.mly:** Shalva (with Emma)	**ast.ml:** Kai-Zhan (with Shalva)
restruct.ml: Arunavha	**semant.ml:** Arunavha	**sast.ml:** Arunavha (with Kai-Zhan)
llvm_generator.ml: Kai-Zhan (with Shalva)	**header_generator.ml:** Kai-Zhan and Emma	**Test suite:** Emma (with Shalva and Kai-Zhan)

6. Test Plan:

Testing was a vital part of our process in ensuring that all parts of the compiler were properly functioning.

6.1 Scanner & Parser Unit Testing:

After the scanner and parser were implemented, we performed unit tests using Menhir to ensure that our scanner was defined correctly, and our parser correctly implemented the grammar and parsed each of the tokens correctly.

6.2 Automation:

The test suite is comprised of 3 test scripts, and **56** test cases. There are **34** positive test cases that test a variety of SAKE programs that range from the simplest basic finite state machine to complex concurrent FSM programs. There are **22** negative test cases, which are designed and used to test both the parser and semantic checker. Positive cases pass if they match the expected output. Negative cases pass if they match the expected error message outputs. If a negative case fails to produce an error, it is considered a failure and there is a flaw in our grammar that is allowing undesirable code to pass through to the code generators.

Our language was designed to be run with external wrapper classes, written in C. The wrapper program provides user input to our FSM by calling the tick function which is generated by `llvm_generator.ml`. Each wrapper program is unique to its SAKE program, and relies on a header file that is generated by header_generator for struct and function definitions.

As such, each success test case comprises of a SAKE program (.sk), wrapper program (.c), and expected output file (.out). During compilation, the SAKE program is compiled to produce an LLVM file (.ll) and a C header file (.h). The LLVM file is then compiled using llc to produce assembly code. The header file includes input, output, and state struct definitions along with the tick function prototype and is thus used in the compilation of the C wrapper program. Once the LLVM and wrapper files are compiled into object files, they are linked together to produce an executable.

A failure test case is comprised of a SAKE program (.sk) and expected error file (.err). A failed test case should never reach the code generation stage of compilation. If an error has not been thrown by either the scanner, parser, or semantic checker by this stage there is an error in our compiler.

6.3 Test Scripts:

The main test script is `testall.sh`. The script, which is based on the MicroC test script, is designed to test all success and failure cases within the test case directory that are titled either `test_*.sk` or `fail_*.sk`. The main test script runs a variety of simple to complex tests.

When testall.sh runs, it prints either `OK` or `FAILED` next to the name of each test case that it runs. When a test case is successful, it will remove all intermediary files that were created during compilation. Users can run `./testall.sh -k` to preempt this feature. `testall.sh` can also take a specific program as its input on the command line. This will only run the specific test specified rather than all success and failure cases. For example:

```
./testall.sh ../testing/test_trafficLightEndTL.sk
test_trafficLightEndTL...OK
```

As stated above, our language relies on wrappers written in C to compile and run properly. For simple test cases, the user might not always want to generate a wrapper file for the program but instead use a generic template wrapper. For example, `test_42.sk`:

```
fsm fortytwo {

       printf("%s\n", 42)
}
```

`test_42.sk` is a simple FSM that contains a single print statement. In this scenario, our language interprets the entire body of the FSM as a single state which will be run each time tick is called. For this test, the user may want to use a generic template wrapper program that simply calls tick once after resetting the tick. In the scenario where the user either chooses not to create a wrapper file, or forgets to create one, testall.sh will generate a generic template wrapper which can then be used in linking.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "test_<testName>.h"

int main() {
              struct test_<testName>_input i;
struct test_<testName>_state s;

              ~ Reset tick
test_<testName>_tick(&s, NULL, NULL);

~ Call tick once
```

```
test_<testName>_tick(&s, &i, NULL);

            return 0;
}
```

When a test case fails, it will print out a short statement explaining the failure, such as "`test_<test_name>.out differs`". For a more in depth analysis, users can open `alltests.log` which is a log of the last execution of `testall.sh`. It contains a more in depth analysis of all commands run during compilation and potential failure messages. For example,

```
###### Testing fail_hello
_build/sake.native ../testing/fail_hello.sk fail_hello 2> fail_hello.err >>
alltests.log
diff -b fail_hello.err ../testing/fail_hello.err > fail_hello.diff
###### SUCCESS

###### Testing test_hogTL
_build/sake.native ../testing/test_hogTL.sk test_hogTL
Fatal error: exception Semantic.SemanticError("undeclared enum value R1")
###### FAILED
```

There are two smaller scripts, `traffic.sh` and `adventure.sh` that test a subset of test cases.

Traffic.sh tests all the traffic light FSMs in the testing directory, displays their output to the screen, and error checks against expected out. These tests are separated out, because there are sleep calls from within the wrappers which cause the tests to take longer to run.

The `adventure.sh` script was created to compile and run our `adventure.sk` program, which requires dynamic input from the users and, as such, is separated from the automated test programs which run on set input. `adventure.sh` runs the same compilation steps as `testall.sh`, but receives input from standard in and outputs to standard out.

Both `traffic.sh` and `adventure.sh` also generates logs, `trafficLights.log` and `adventureStory.log` respectively, containing information about the last execution of respective scripts. Both scripts can also take the command line input of a specific test file they wish to run.

6.4 Unit Testing:

After we started implementing code generation, we also performed unit testing on SAKE programs to ensure that the header generator was correctly implemented and creating header files that could be linked with wrapper programs written in C. These tests were running using the `testall.sh` script which produced the header files that we could then compare to expected header files.

As the LLVM generator was being developed, we also developed simple short SAKÉ programs to test the compilation and linking of LLVM, header, and C wrapper files. These tests covered basic concepts like variable declaration, empty FSMs, keywords, literals, and operators to name but a few. The focused on specific areas of our language which allowed us to pinpoint errors being generated by semantic checking and code generation. These tests cases were then developed and expanded upon for integration testing.

6.5 Integration Testing:

With integration tests, we build on and wrote more complex SAKÉ programs to test the implementation of our language and generated tick function. These tests included longer single FSMs like traffic light, concurrent FSMs that were both independent and dependent on each other, and halting test. Sample test cases include:

6.5.1 Traffic Light

6.5.1.1 Source files:

test_trafficLightTL.sk

```
input[int i]
output[char out]

fsm trafficLight {

              state Red
                     switch(i) {
                                   case 1: out = ‘g’
                                                 goto Green
                                   case 0: out = ‘r’
                                                 goto Red
                                          }
              state Yellow
                     out = ‘r’
                     goto Red
              state Green
                     switch(i) {
                                   case 1: out = ‘g’
                                                 goto Green
                                   case 0: out = ‘y’
                                                 goto Yellow
                                          }
}
```

test_trafficLightTL.c

```
#include <stdio.h>
#include <stdlib.h>
#include "test_trafficLightTL.h"

int main() {
struct test_trafficLightTL_input in;
              struct test_trafficLightTL_state s;
              struct test_trafficLightTL_output o;

              test_trafficLightTL_tick(&s, NULL, NULL);

char *input = "00011111000";

char temp[1];

while (*input) {
temp[0] = input[0];
in.i = atoi(temp);

test_trafficLightTL_tick(&s, &in, &o);
```

```
printf("Light color: %c\n", o.out);

 input++;
}

              return 0;
}
```

6.5.12 Target file:

test_trafficLightTL.ll

```
; ModuleID = 'sake'

declare i32 @printf(i8*, ...)

declare void @memcpy({ i32, i32 }*, { i32, i32 }*, i64)

define void @trafficLight({ i32, i32 }*, { i32, i32 }*, { i32 }*, { i8 }*) {
entry:
  %trafficLight = getelementptr inbounds { i32, i32 }* %1, i32 0, i32 1
  %trafficLight1 = load i32* %trafficLight
  switch i32 %trafficLight1, label %"*halt" [
    i32 0, label %"*init"
    i32 3, label %Green
    i32 1, label %Red
    i32 2, label %Yellow
  ]

"*init":                                          ; preds = %entry
  br label %Red

"*halt":                                          ; preds = %entry
  %ptr = getelementptr inbounds { i32, i32 }* %0, i32 0, i32 0
  store i32 0, i32* %ptr
  ret void

Red:                                              ; preds = %"*init", %entry
  %i = getelementptr inbounds { i32 }* %2, i32 0, i32 0
  %i2 = load i32* %i
  switch i32 %i2, label %merge [
    i32 1, label %case
    i32 0, label %case4
  ]

Yellow:                                           ; preds = %entry
  %out8 = getelementptr inbounds { i8 }* %3, i32 0, i32 0
  store i8 114, i8* %out8
  %trafficLight9 = getelementptr inbounds { i32, i32 }* %0, i32 0, i32 1
  store i32 1, i32* %trafficLight9
  ret void
```

```
Green:                                            ; preds = %entry
  %i11 = getelementptr inbounds { i32 }* %2, i32 0, i32 0
  %i12 = load i32* %i11
  switch i32 %i12, label %merge10 [
    i32 1, label %case13
    i32 0, label %case16
  ]

merge:                                            ; preds = %Red
  %trafficLight7 = getelementptr inbounds { i32, i32 }* %0, i32 0, i32 1
  store i32 2, i32* %trafficLight7
  ret void

case:                                             ; preds = %Red
  %out = getelementptr inbounds { i8 }* %3, i32 0, i32 0
  store i8 103, i8* %out
  %trafficLight3 = getelementptr inbounds { i32, i32 }* %0, i32 0, i32 1
  store i32 3, i32* %trafficLight3
  ret void

case4:                                            ; preds = %Red
  %out5 = getelementptr inbounds { i8 }* %3, i32 0, i32 0
  store i8 114, i8* %out5
  %trafficLight6 = getelementptr inbounds { i32, i32 }* %0, i32 0, i32 1
  store i32 1, i32* %trafficLight6
  ret void

merge10:                                          ; preds = %Green
  ret void

case13:                                           ; preds = %Green
  %out14 = getelementptr inbounds { i8 }* %3, i32 0, i32 0
  store i8 103, i8* %out14
  %trafficLight15 = getelementptr inbounds { i32, i32 }* %0, i32 0, i32 1
  store i32 3, i32* %trafficLight15
  ret void

case16:                                           ; preds = %Green
  %out17 = getelementptr inbounds { i8 }* %3, i32 0, i32 0
  store i8 121, i8* %out17
  %trafficLight18 = getelementptr inbounds { i32, i32 }* %0, i32 0, i32 1
  store i32 2, i32* %trafficLight18
  ret void
}

define { i32, i32 }* @test_trafficLightTL_tick({ i32, i32 }*, { i32 }*, { i8 }*) {
entry:
  %null = icmp eq { i32 }* %1, null
  br i1 %null, label %reset, label %check

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

```
  store i32 0, i32* %reset2
  ret { i32, i32 }* null

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

update:                                           ; preds = %check
  %state = alloca { i32, i32 }
  call void @memcpy({ i32, i32 }* %state, { i32, i32 }* %0, i64 mul nuw (i64 ptrtoint
(i32* getelementptr (i32* null, i32 1) to i64), i64 2))
  call void @trafficLight({ i32, i32 }* %state, { i32, i32 }* %0, { i32 }* %1, { i8 }*
%2)
  call void @memcpy({ i32, i32 }* %0, { i32, i32 }* %state, i64 mul nuw (i64 ptrtoint
(i32* getelementptr (i32* null, i32 1) to i64), i64 2))
  ret { i32, i32 }* %0

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

6.5.1.3 Output:

test_trafficLightTL.out

```
Light color: r
Light color: r
Light color: r
Light color: g
Light color: g
Light color: g
Light color: g
Light color: g
Light color: y
Light color: r
Light color: r
```

6.5.2 Concurrent test

6.5.2.1 Source files:

test_concurrent.sk

```
input[int i]
output[int o]

fsm one {
              public int j = 4
              int r = two.k

              state DependK
                     r = two.k
                     printf(“%d\n”, r)
                     goto DependK
}
fsm two {
              public int k = 3

              state SetK
                     if (I == 2) {
                                   k = 45
                     }
                     printf(“%d\n”, k)
                     goto SetK
}
```

test_concurrent.c

```
#include <stdio.h>
#include "test_concurrent.h"

int main() {
              struct test_concurrent_input i;
              struct test_concurrent_state s;

              test_concurrent_tick(&s, NULL, NULL);

        i.i = 42;
        test_concurrent_tick(&s, &i, NULL);

        i.i = 2;
        test_concurrent_tick(&s, &i, NULL);

        i.i = 42;
        test_concurrent_tick(&s, &i, NULL);

              return 0;
}
```

6.5.2.1 Target files:

test_concurrent.ll

```
; ModuleID = 'sake'

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt1 = private unnamed_addr constant [4 x i8] c"%d\0A\00"

declare i32 @printf(i8*, ...)

declare void @memcpy({ i32, i32, i32, i32, i32 }*, { i32, i32, i32, i32, i32 }*, i64)

define void @one({ i32, i32, i32, i32, i32 }*, { i32, i32, i32, i32, i32 }*, { i32 }*,
{ i32 }*) {
entry:
  %r = alloca i32
  %two_k = getelementptr inbounds { i32, i32, i32, i32, i32 }* %1, i32 0, i32 4
  %two_k1 = load i32* %two_k
  store i32 %two_k1, i32* %r
  %one = getelementptr inbounds { i32, i32, i32, i32, i32 }* %1, i32 0, i32 1
  %one2 = load i32* %one
  switch i32 %one2, label %"*halt" [
    i32 0, label %"*init"
    i32 1, label %DependK
  ]

"*init":                                          ; preds = %entry
  br label %DependK

"*halt":                                          ; preds = %entry
  %ptr = getelementptr inbounds { i32, i32, i32, i32, i32 }* %0, i32 0, i32 0
  store i32 0, i32* %ptr
  ret void

DependK:                                          ; preds = %"*init", %entry
  %two_k3 = getelementptr inbounds { i32, i32, i32, i32, i32 }* %1, i32 0, i32 4
  %two_k4 = load i32* %two_k3
  store i32 %two_k4, i32* %r
  %r5 = load i32* %r
  %printf = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @fmt,
i32 0, i32 0), i32 %r5)
  %one6 = getelementptr inbounds { i32, i32, i32, i32, i32 }* %0, i32 0, i32 1
  store i32 1, i32* %one6
  ret void
}

define void @two({ i32, i32, i32, i32, i32 }*, { i32, i32, i32, i32, i32 }*, { i32 }*,
{ i32 }*) {
entry:
  %two = getelementptr inbounds { i32, i32, i32, i32, i32 }* %1, i32 0, i32 2
```

```
  %two1 = load i32* %two
  switch i32 %two1, label %"*halt" [
    i32 0, label %"*init"
    i32 1, label %SetK
  ]

"*init":                                          ; preds = %entry
  br label %SetK

"*halt":                                          ; preds = %entry
  %ptr = getelementptr inbounds { i32, i32, i32, i32, i32 }* %0, i32 0, i32 0
  store i32 0, i32* %ptr
  ret void

SetK:                                             ; preds = %"*init", %entry
  %i = getelementptr inbounds { i32 }* %2, i32 0, i32 0
  %i2 = load i32* %i
  %tmp = icmp eq i32 %i2, 2
  br i1 %tmp, label %then, label %else

merge:                                            ; preds = %else, %then
  %k3 = getelementptr inbounds { i32, i32, i32, i32, i32 }* %0, i32 0, i32 4
  %k4 = load i32* %k3
  %printf = call i32 (i8*, ...)* @printf(i8* getelementptr inbounds ([4 x i8]* @fmt1,
i32 0, i32 0), i32 %k4)
  %two5 = getelementptr inbounds { i32, i32, i32, i32, i32 }* %0, i32 0, i32 2
  store i32 1, i32* %two5
  ret void

then:                                             ; preds = %SetK
  %k = getelementptr inbounds { i32, i32, i32, i32, i32 }* %0, i32 0, i32 4
  store i32 45, i32* %k
  br label %merge

else:                                             ; preds = %SetK
  br label %merge
}

define { i32, i32, i32, i32, i32 }* @test_concurrent_tick({ i32, i32, i32, i32, i32
}*, { i32 }*, { i32 }*) {
entry:
  %null = icmp eq { i32 }* %1, null
  br i1 %null, label %reset, label %check

reset:                                            ; preds = %entry
  %reset1 = getelementptr inbounds { i32, i32, i32, i32, i32 }* %0, i32 0, i32 0
  store i32 1, i32* %reset1
  %reset2 = getelementptr inbounds { i32, i32, i32, i32, i32 }* %0, i32 0, i32 1
  store i32 0, i32* %reset2
  %reset3 = getelementptr inbounds { i32, i32, i32, i32, i32 }* %0, i32 0, i32 2
  store i32 0, i32* %reset3
  %reset4 = getelementptr inbounds { i32, i32, i32, i32, i32 }* %0, i32 0, i32 3
  store i32 4, i32* %reset4
  %reset5 = getelementptr inbounds { i32, i32, i32, i32, i32 }* %0, i32 0, i32 4
```

```
  store i32 3, i32* %reset5
  ret { i32, i32, i32, i32, i32 }* null

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

update:                                           ; preds = %check
  %state = alloca { i32, i32, i32, i32, i32 }
  call void @memcpy({ i32, i32, i32, i32, i32 }* %state, { i32, i32, i32, i32, i32 }*
%0, i64 mul nuw (i64 ptrtoint (i32* getelementptr (i32* null, i32 1) to i64), i64 5))
  call void @one({ i32, i32, i32, i32, i32 }* %state, { i32, i32, i32, i32, i32 }* %0,
{ i32 }* %1, { i32 }* %2)
  call void @two({ i32, i32, i32, i32, i32 }* %state, { i32, i32, i32, i32, i32 }* %0,
{ i32 }* %1, { i32 }* %2)
  call void @memcpy({ i32, i32, i32, i32, i32 }* %0, { i32, i32, i32, i32, i32 }*
%state, i64 mul nuw (i64 ptrtoint (i32* getelementptr (i32* null, i32 1) to i64), i64
5))
  ret { i32, i32, i32, i32, i32 }* %0

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

6.5.2.3 Output:

test_concurrent.out

```
3
3
3
45
45
45
```

6.6 Test Cases:

Testall.sh running all test cases in test directory that are titled either test_*.sk or fail_*.sk

```
./testall.sh
test_42...OK
test_block...OK
test_bool...OK
test_char...OK
test_comment...OK
test_concurrent...OK
test_conjunctions...OK
test_emptyfsm...OK
test_enums...OK
test_for...OK
test_fsmhello...OK
test_halt...OK
test_header...OK
test_hello...OK
test_ifelse...OK
test_multiSwitch...OK
test_multiSwitch2...OK
test_nestedFor...OK
test_nestedIf...OK
test_nestedIf2...OK
test_nestedWhile...OK
test_onePub...OK
test_printing...OK
test_string...OK
test_switch...OK
test_switch1...OK
test_variables...OK
test_while...OK
fail_assign...OK
fail_comments...OK
fail_dupEnums...OK
fail_dupFsm...OK
fail_dupGlobal...OK
fail_dupLocal...OK
fail_dupPublic...OK
fail_dupStates...OK
fail_dupVars...OK
fail_empty...OK
fail_fsmhello...OK
fail_hello...OK
fail_illegalAssign...OK
fail_illegalBin...OK
fail_illegalOp...OK
fail_illegalPredicate...OK
fail_noState...OK
fail_printing...OK
fail_typeMismatch...OK
```

```
fail_undeclared...OK
fail_underscore...OK
```

Traffic.sh running the traffic light test cases in the directory which are distinguished by TL at end of name.

```
./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

test_hogTL...
TL 1: r             TL 2: r
TL 1: g             TL 2: r
TL 1: g             TL 2: r
TL 1: g             TL 2: r
TL 1: y             TL 2: r
TL 1: r             TL 2: r
TL 1: r             TL 2: g
TL 1: r             TL 2: g
TL 1: r             TL 2: g
TL 1: r             TL 2: g
TL 1: r             TL 2: y
TL 1: r             TL 2: r
TL 1: g             TL 2: r
TL 1: g             TL 2: r
TL 1: g             TL 2: r
TL 1: y             TL 2: r
TL 1: r             TL 2: r
TL 1: r             TL 2: g
TL 1: r             TL 2: y
TL 1: r             TL 2: r
OK

test_loopingTL...
TL 1: r             TL 2: r
TL 1: g             TL 2: r
TL 1: g             TL 2: r
TL 1: g             TL 2: r
TL 1: g             TL 2: r
TL 1: y             TL 2: r
TL 1: r             TL 2: r
TL 1: r             TL 2: g
```

```
TL 1: r             TL 2: g
TL 1: r             TL 2: g
TL 1: r             TL 2: g
TL 1: r             TL 2: g
TL 1: r             TL 2: g
TL 1: r             TL 2: g
TL 1: r             TL 2: g
TL 1: r             TL 2: g
TL 1: r             TL 2: g
TL 1: r             TL 2: g
TL 1: r             TL 2: y
TL 1: r             TL 2: r
TL 1: g             TL 2: r
TL 1: g             TL 2: r
TL 1: g             TL 2: r
TL 1: g             TL 2: r
TL 1: g             TL 2: r
TL 1: g             TL 2: r
TL 1: g             TL 2: r
TL 1: y             TL 2: r
TL 1: r             TL 2: r
TL 1: r             TL 2: g
TL 1: r             TL 2: g
TL 1: r             TL 2: g
OK

test_trafficLightEndTL...
Light color: r
Light color: r
Light color: r
Light color: g
Light color: g
Light color: g
Light color: g
Light color: g
Light color: y
Light color: r
Light color: r
OK

test_trafficLightTL...
Light color: r
Light color: r
Light color: r
Light color: g
Light color: g
Light color: g
Light color: g
Light color: g
Light color: y
Light color: r
Light color: r
OK
```

```
test_unreachableTL...
Light color: g
Light color: g
Light color: g
Light color: y
Light color: y
Light color: y
Light color: y
Light color: y
Light color: y
Light color: y
Light color: y
OK
```

Each testing script and all of the test cases are copied in the appendix.

7. Lessons Learned:

7.1 Shalva Kohen

Communication is key. In a project so design-based it is vital that every group knows what they should be doing and implementing things correctly. Otherwise, they waste time implementing a feature that should have been done a different way which in turn impacts the rest of group. I didn't know how much I would have to just confirm with others that what I was doing was what we had all intended. Furthermore, understanding the emotional state of your teammates is important. If a group member is upset, then they will harbor those feelings until they explode at the most inopportune time. Therefore, issues pertaining to group dynamic should be dealt with immediately, else the productivity of the group is effected.

On a more technical level, I realized the importance of planning. Obviously we were learning the material as we coded the project so it was hard to plan what we could and could not do ahead of time. But in the future when I am tasked with a project, I should plan out every little detail and understand everything pertaining to that task before I actually start coding. This will reduce bugs, redundancy, and make the group overall more efficient because debugging time will go down.

7.2 Arunavha Chanda

Communicate, communicate, communicate. Being manager in a group with 3 of the finest students I have ever known was no easy job, but what really smoothed the path a lot was our willingness to communicate. When everyone who excels individually comes together for a design-based group project, clashing egos and views are natural. What is most important is the ability to curtail that ego, throw your stubbornness out the window and be completely open in judgment. You will still not agree, but you will learn to value other people's opinions and learn to see merits in different ways of doing things. This project teaches a very important lesson in that, even if you disagree with someone, if everyone else agrees with them, there may be multiple correct answers. There doesn't always have to be only one way to it. Being willing to accommodate others' views is key.

In terms of the work itself, I realized that one of the hardest parts of a project like this is not knowing what's going on in another person's division at times. You don't feel the 360-degree level of control as in an individual project and can panic. This is why communication and understanding between group members is so important: not just for harmony, but also efficiency.

7.3 Kai-Zhan Lee

Regarding technical design and implementation: constant clarity in the construction, planning, and delegation of tasks. Even if all group members are working at top efficiency on a project, work and personal visions for a project will inevitably conflict without a clear end goal. Therefore, in addition to learning how to efficiently write, navigate, and plan the development of OCaml code, including that of the LLVM API, I have learned that it is best not to be overly and unnecessarily creative in designing a new language; when implementing less vital details, it is better to build off existing frameworks than it is to go off on one's own. Though it is easy to be misled into designing everything from scratch by the sense of liberty one derives from building one's own language, making incremental but significant additions to existing work is far more conducive to creating a clear and robust language.

With respect to teamwork, I have also learned that it is important both for a healthy group social dynamic and for productivity to be especially sensitive to others' feelings when explaining reasoning that contradicts their claims. Of course, it is important to compromise with design-related choices, but if the issue at hand is logically rooted, any frustrations or negative feelings that arise from contradiction must be addressed promptly.

7.4 Emma Etherington

Start early. Communicate. Never be afraid to disagree with your teammates and ask them to justify their reasoning. All work on the same platform. Don't lose your temper. Those are beyond a doubt, the most important things I learnt from this project.

The choices you make on a semester long project are truly like digging your own graves. If you group is working on different platforms, especially if they use different versions of Ocaml or LLVM, you are going to have big problems. Code that works on later versions might not work on early ones. My advice to any future group would be don't be lead astray or down rabbit holes by your teammates. Communication is key to ensuring all teammates are on the same page when it comes to what needs to be versus what you might want to implement. As much as you might not agree with or understand what another group member is doing or thinking, they probably think they have a good reason for doing it. Hear them out, and *actually* listen to what they have to say before disagreeing with them. It can be super hard to work in a group, especially if all the members are you friends. Attempt to be patient, and if that doesn't work, then just make a better argument. At the end of the day, it doesn't matter whose idea it was or who raised their voice the loudest, the ideas that have the best arguments and cases are the ones that should get implemented.

8. Appendix:

8.1 AST: ast.ml

```
(*
 * Abstract syntax tree for the Sake language
 *
 * Author: Kai-Zhan Lee
 * Credzz: Shalva Kohen, critical additions for proper parsing
 *)
type op = Add | Sub | Mul | Div | Eq | Neq | Lt | Le | Gt | Ge | And | Or
type uop = Neg | Not
type dtype = (* built-in primitives + custom user type *)
  | Bool | Int | Char | String
  | Enum of string (* enum typename *)
type lvalue = dtype * string
type expr =
  | BoolLit of bool
  | CharLit of char
  | IntLit of int
  | StringLit of string
  | Variable of string
  | EnumLit of string
  | Access of string * string
  | Uop of uop * expr
  | Binop of expr * op * expr
  | Assign of string * expr
  | Printf of string * expr list
  | Empty
type stmt =
  | Block of stmt list
  | State of string
  | If of expr * stmt * stmt
  | For of string * int * int * int * stmt
  | While of expr * stmt
  | Switch of expr * (expr * stmt list) list
  | Expr of expr
  | Goto of string (* for FSM transitions *)
  | Halt
type type_decl = {
  type_name   : string;
  type_values : string list;
}
type fsm_decl = {
  fsm_name  : string;
  fsm_public: (dtype * string * expr) list;
  fsm_locals: (dtype * string * expr) list;
  fsm_body  : stmt list;
}
```

```
type program = {
  input : lvalue list;
  output: lvalue list;
  types : type_decl list;
  fsms  : fsm_decl list;
}
```

8.2 Scanner: scanner.mll

```
(*
 * scanner.mll to scan in tokens
 * Authors: Arunavha Chanda and Shalva Kohen
 *)

{ open Parser
  let unescape u =
       Scanf.sscanf("\"" ^ u ^ "\"") "%S%!" (fun x -> x)
}

let spc = [' ' '\t']
let cap = ['A'-'Z']
let low = ['a'-'z']
let ltr = (cap | low)
let dgt = ['0'-'9']
let aln = (ltr | dgt)

rule token = parse
    spc { token lexbuf }
  | "/~" { comment lexbuf }
  | '~' { line_comment lexbuf }
  | '.' { DOT }
  | '_' { UNDER }
  | '|' { BAR }
  | '\n'     { NLINE }
  | '('      { LPAREN }
  | ')'      { RPAREN }
  | '{'      { LBRACE }
  | '}'      { RBRACE }
  | '['      { LSQUARE }
  | ']'      { RSQUARE }
  | ';'      { SEMI }
  | ':'      { COLON }
  | ','      { COMMA }
  | '+'      { ADD }
  | '-'      { SUB }
  | '*'      { MUL }
  | '/'      { DIV }
  | '='      { ASSIGN }
  | '"'      { QUOTES }
  | "=="     { EQ }
  | "!="     { NEQ }
  | '<'      { LT }
  | "<="     { LE }
  | ">"      { GT }
  | ">="     { GE }
  | "&&"     { AND }
  | "||"     { OR }
  | "!"      { NOT }
  | "if"     { IF }
```

```
  | "else"   { ELSE }
  | "for"    { FOR }
  | "in"     { IN }
  | "while"  { WHILE }
  | "int"    { INT }
  | "bool"   { BOOL }
  | "void"   { VOID }
  | "char"   { CHAR }
  | "string"   { STRING }
  | "true"   { TRUE }
  | "false"  { FALSE }
  | "printf"  { PRINTF }
  | "fsm" { FSM }
  | "type" { TYPE }
  | "goto" { GOTO }
  | "switch" { SWITCH }
  | "case" { CASE }
  | "goto" { GOTO }
  | "state" { STATE }
  | "start" { START }
  | "input" { INPUT }
  | "output" { OUTPUT }
  | "public" { PUBLIC }
  | "halt" { HALT }
  | cap aln*  as lxm { TYPENAME (lxm) }
  | low aln* as lxm { ID (lxm) }
  | dgt+ as num { INTLIT (int_of_string num) }
  | '''([^''']  as ch_litr)'''            { CHARLIT(ch_litr)}
  | '"'([^'"']* as st_litr)'"'            { STRINGLIT(unescape st_litr)}
  | eof { EOF }
  | _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }
and comment = parse
    "~/\n" { token lexbuf }
  | _    { comment lexbuf }
and line_comment = parse
    '\n' { token lexbuf }
  | _    { line_comment lexbuf }
```

8.3 Parser: parser.mly

```
/* Author: Shalva Kohen */
%{ open Ast %}

%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA ASSIGN BAR COLON QUOTES DOT LSQUARE
RSQUARE NLINE UNDER
%token ADD SUB MUL DIV
%token EQ NEQ LT LE GT GE AND OR NEG NOT TRUE FALSE
%token IF ELSE FOR WHILE IN
%token INT BOOL VOID CHAR STRING
%token EOF

/* tokens specific to our language */
%token TYPE SWITCH CASE GOTO FSM STATE START INPUT OUTPUT PUBLIC
%token PRINTF HALT

/* ASSOCIATIVITY */
%nonassoc NOELSE
%nonassoc ELSE
%left RETURN
%left COMMA
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LE GE
%left ADD SUB
%left MUL DIV
%right NOT NEG

/* literals */
%token <int> INTLIT
%token <char> CHARLIT
%token <string> STRINGLIT
%token <string> ESCAPE
%token <string> ID
%token <string> TYPENAME

%start program
%type <Ast.program> program

%%
/* grammar */
dtype:
| BOOL { Bool }
| INT { Int }
| CHAR { Char }
| STRING { String }
| TYPENAME { Enum($1) }

lvalue:
```

```
 dtype ID { $1, $2 }

/* expressions */
expr:
| INTLIT { IntLit($1) }
| TRUE { BoolLit(true) }
| FALSE { BoolLit(false) }
| CHARLIT { CharLit($1) }
| STRINGLIT { StringLit($1) }
| ID { Variable($1) }
| TYPENAME { EnumLit($1) }
| SUB expr %prec NEG { Uop(Neg, $2) }
| NOT expr { Uop(Not, $2) }
| expr ADD expr { Binop($1, Add, $3) }
| expr SUB expr { Binop($1, Sub, $3) }
| expr MUL expr { Binop($1, Mul, $3) }
| expr DIV expr { Binop($1, Div, $3) }
| expr EQ expr { Binop($1, Eq, $3) }
| expr NEQ expr { Binop($1, Neq, $3) }
| expr LT expr { Binop($1, Lt, $3) }
| expr LE expr { Binop($1, Le, $3) }
| expr GT expr { Binop($1, Gt, $3) }
| expr GE expr { Binop($1, Ge, $3) }
| expr AND expr { Binop($1, And, $3) }
| expr OR expr { Binop($1, Or, $3) }
| ID ASSIGN expr { Assign($1, $3) }
| PRINTF LPAREN STRINGLIT COMMA actuals_list RPAREN { Printf($3, List.rev $5) }
| PRINTF LPAREN ESCAPE COMMA actuals_list RPAREN { Printf($3 ^ "\n", List.rev $5) }
| ID DOT ID { Access($1, $3) }

/* statements */
stmt:
| LBRACE NLINE stmt_list RBRACE NLINE { Block(List.rev $3) }
| STATE TYPENAME NLINE { State($2) }
| IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
| IF LPAREN expr RPAREN stmt ELSE stmt { If($3, $5, $7) }  /* with else */
| FOR ID IN LPAREN INTLIT COLON INTLIT COLON INTLIT RPAREN stmt { For($2, $5, $7, $9,
$11) }
| WHILE LPAREN expr RPAREN stmt { While($3, $5) }
| expr NLINE { Expr($1) }
| SWITCH LPAREN expr RPAREN LBRACE cstmt_list RBRACE NLINE { Switch($3, List.rev $6) }
| GOTO TYPENAME NLINE { Goto ($2) }
| HALT NLINE { Halt }

cstmt:
  CASE expr COLON stmt_list { $2, List.rev $4 }

type_decl:
  TYPE TYPENAME ASSIGN string_opt
  {{
    type_name = $2;
    type_values = $4;
  }}
```

```
fsm_decl:
  FSM ID LBRACE NLINE public_opt local_opt NLINE stmt_list RBRACE NLINE
{{
  fsm_name = $2;
  fsm_public = List.rev $5;
  fsm_locals = List.rev $6;
  fsm_body = List.rev $8;
}}

program:
| INPUT LSQUARE lvalue_list RSQUARE NLINE
  OUTPUT LSQUARE lvalue_list RSQUARE NLINE NLINE
  type_opt fsm_list EOF
  {{
    input = List.rev $3;
    output = List.rev $8;
    types = $12;
    fsms = List.rev $13;
  }}
| type_opt fsm_list EOF
  {{
    input = [];
    output = [];
    types = $1;
    fsms = List.rev $2;
  }}


/* list definitions */
actuals_list:
| expr { [$1] }
| actuals_list COMMA expr { $3 :: $1}

stmt_list:
| /* nothing */ { [] }
| stmt_list stmt { $2 :: $1 }

cstmt_list:
| NLINE { [] }
| cstmt_list cstmt {  $2 :: $1 }

string_opt:
| /* nothing */ { [] }
| string_list { List.rev $1 }

string_list:
| TYPENAME { [$1] }
| string_list BAR TYPENAME { $3 :: $1 }

lvalue_list:
| lvalue { [$1] }
| lvalue_list COMMA lvalue { $3 :: $1 }

dstexpr:
```

```
| dtype ID ASSIGN expr { $1, $2, $4 }
| dtype ID { $1, $2, Empty }

public_opt:
| /* nothing */ { [] }
| public_list NLINE { List.rev $1 }

public_list:
| PUBLIC dstexpr { [$2] }
| public_list NLINE PUBLIC dstexpr { $4 :: $1 }

local_opt:
| /* nothing */ { [] }
| local_list NLINE { List.rev $1 }

local_list:
| dstexpr { [$1] }
| local_list NLINE dstexpr { $3 :: $1 }

type_opt:
| /* nothing */ { [] }
| type_list NLINE NLINE { List.rev $1 }

type_list:
| type_decl { [$1] }
| type_list type_decl { $2 :: $1 }

fsm_list:
| /* nothing */ { [] }
| fsm_list fsm_decl { $2 :: $1 }
```

8.4 SAST: sast.ml

```
(* SAST *)
(* Author: Arunavha Chanda *)
type op = Add | Sub | Mul | Div | Eq | Neq | Lt | Le | Gt | Ge | And | Or
type uop = Neg | Not
type dtype = (* built-in primitives + custom user type *)
  | Bool | Int | Char | String | Enum of string (* just the name of the enum *)
type expr =
  | BoolLit of bool
  | CharLit of char
  | IntLit of int
  | StringLit of string
  | Variable of string
  | Uop of uop * expr
  | Binop of expr * op * expr
  | Assign of string * expr
  | Printf of string * expr list
  | Empty
type stmt =
  | Block of stmt list
  | State of string
  | If of expr * stmt * stmt
  | For of string * (int * int * int) * stmt
  | While of expr * stmt
  | Switch of expr * (expr * stmt list) list
  | Expr of expr
  | Goto of string (* for FSM transitions *)
  | Halt
type type_decl = {
  type_name   : string;
  type_values : string list;
}
type fsm_decl = {
  fsm_name  : string;
  fsm_states: (string * int) list;
  fsm_locals: (dtype * string * expr) list;
  fsm_body  : stmt list;
}
type program = {
  input : (dtype * string) list;
  output: (dtype * string) list;
  public: (dtype * string * expr) list;
  types : type_decl list;
  fsms  : fsm_decl list;
}
type variable_decl = (string * dtype)
type symbol_table = {
  parent : symbol_table option;
  mutable variables : variable_decl list
}
type translation_environment = {
  scope : symbol_table;   (* symbol table for vars *)
}
```

8.5 Code generator: llvm_generator.ml

```
(*
 * The function translate converts a Sast.program to an Llvm.llmodule.
 *
 * Author: Kai-Zhan Lee
 * Credzz: Shalva Kohen for A.Switch structure and minor bug fixes.
 *)

module L = Llvm
module A = Sast

module StringMap = Map.Make(String)

exception ENOSYS of string
exception Bug of string

(* Translate an A.program to LLVM *)
let translate filename program =
  let context = L.global_context () in
  let sake = L.create_module context "sake"
    and i64_t  = L.i64_type  context
    and i32_t  = L.i32_type  context
    and i8_t   = L.i8_type   context
    and i1_t   = L.i1_type   context
    and void_t = L.void_type context in

  (* Helper functions *)
  let lltype = function
    | A.Int -> i32_t
    | A.Char -> i8_t
    | A.Bool -> i1_t
    | A.Enum _ -> i32_t
    | A.String -> L.pointer_type i8_t in
  let llop = function
    | A.Add -> L.build_add
    | A.Sub -> L.build_sub
    | A.Mul -> L.build_mul
    | A.Div -> L.build_sdiv
    | A.Eq  -> L.build_icmp L.Icmp.Eq
    | A.Neq -> L.build_icmp L.Icmp.Ne
    | A.Lt  -> L.build_icmp L.Icmp.Slt
    | A.Le  -> L.build_icmp L.Icmp.Sle
    | A.Gt  -> L.build_icmp L.Icmp.Sgt
    | A.Ge  -> L.build_icmp L.Icmp.Sge
    | A.And -> L.build_and
    | A.Or  -> L.build_or in
  let lluop = function
    | A.Neg -> L.build_neg
    | A.Not -> L.build_not in
  let lldtype (t, _) = lltype t in
  let bae = L.builder_at_end context in
```

```
let abc = L.append_block context in

let zero = L.const_int i32_t 0
and pos1 = L.const_int i32_t 1 in

(* New types *)
let input_t =
  let types = Array.of_list (List.map lldtype program.A.input) in
  L.struct_type context types in
let output_t =
  let types = Array.of_list (List.map lldtype program.A.output) in
  L.struct_type context types in
let state_t =
  let public =
    let fsms = List.map (fun _ -> lltype A.Int) program.A.fsms in
    let public = List.map (fun (t, _, _) -> lltype t) program.A.public in
    i32_t :: fsms @ public in (* _running variable *)
  let types = Array.of_list public in
  L.struct_type context types in

(* External functions *)
let printf =
  let ftype = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  L.declare_function "printf" ftype sake in
let memcpy =
  let state_t_ptr = L.pointer_type state_t in
  let args = [| state_t_ptr; state_t_ptr; i64_t |] in
  let ftype = L.function_type void_t args in
  L.declare_function "memcpy" ftype sake in

(* Debugging *)
let debug = false in
let init = L.const_int i32_t in
let gsp s b = if debug then L.build_global_stringptr s s b else zero in
let debug s l builder =
  if debug then
    let args = gsp ("\027[31m"^s^"\027[0m") builder :: l in
    List.iter L.dump_value args;
    ignore (L.build_call printf (Array.of_list args) "" builder)
  else () in

(* Variable maps *)
let imap =
  let rec imap i a = function [] -> a
    | (_, n) :: tail -> imap (i + 1) (StringMap.add n i a) tail in
  imap 0 StringMap.empty in
let public = (* int StringMap : names -> struct indices *)
  let fsms = List.map (fun f -> (A.Int, f.A.fsm_name)) program.A.fsms in
  let public = List.map (fun (t, n, _) -> (t, n)) program.A.public in
  imap ((A.Int, "*running") :: fsms @ public)
and input  = imap program.A.input
and output = imap program.A.output in

(* FSM-specific metadata *)
```

```
let locals = ref StringMap.empty and states = ref StringMap.empty in

(* Lookup function; search locals, local public, public, input/output *)
let lookup fn io name builder =
  try StringMap.find name !locals with
    Not_found ->
      let fa = L.params fn
      and io_if v1 v2 = if io == input then v1 else v2
      and local = (L.value_name fn) ^ "_" ^ name (* local name in FSM *) in
      if StringMap.mem local public || StringMap.mem name public then
        let pub_val, pub_ptr = try StringMap.find local public, fa.(0) with
          Not_found -> StringMap.find name public, io_if fa.(1) fa.(0) in
        debug "lookup %s: public[%d]\n"
        [gsp name builder; init pub_val] builder;
        L.build_struct_gep pub_ptr pub_val name builder
      else
        let io_ptr = io_if fa.(2) fa.(3)
        and io_val = try StringMap.find name io with
          Not_found -> raise (Bug (Printf.sprintf "No variable: %s" name)) in
        debug "lookup %s: %s[%d]\n" [gsp name builder;
        gsp (if io == input then "input" else "output") builder;
        init io_val] builder;
        L.build_struct_gep io_ptr io_val name builder in

(* Expression builder *)
let rec expr fn builder = function
  | A.IntLit i -> L.const_int i32_t i
  | A.BoolLit b -> L.const_int i1_t (if b then 1 else 0)
  | A.CharLit c -> L.const_int i8_t (int_of_char c)
  | A.StringLit s -> L.build_global_stringptr s "string" builder
  | A.Empty -> L.const_int i32_t 0
  | A.Variable s ->
      let value = L.build_load (lookup fn input s builder) s builder in
      debug "access %s: %d\n" [gsp s builder; value] builder;
      value
  | A.Printf (fmt, args) ->
      let args = (List.map (expr fn builder) args) in
      let args = (L.build_global_stringptr fmt "fmt" builder) :: args in
      let args = Array.of_list args in
      L.build_call printf args "printf" builder
  | A.Uop (uop, e) -> (lluop uop) (expr fn builder e) "tmp" builder
  | A.Binop (e1, op, e2) ->
      (llop op) (expr fn builder e1) (expr fn builder e2) "tmp" builder
  | A.Assign (s, e) ->
      let e = expr fn builder e in
      ignore (L.build_store e (lookup fn output s builder) builder);
      debug "assign %s: %d\n" [gsp s builder; e] builder; e in

let add_terminal builder f =
  match L.block_terminator (L.insertion_block builder) with
    | Some _ -> ()
    | None -> ignore (f builder) in

(* Statement builder *)
```

```
let rec stmt fn builder = function
  | A.Block body -> List.fold_left (stmt fn) builder body
  | A.Expr e -> ignore (expr fn builder e); builder
  | A.If (predicate, then_stmt, else_stmt) ->
      let merge_bb = abc "merge" fn in
      let then_bb = abc "then" fn in
      let else_bb = abc "else" fn in
      let cond = expr fn builder predicate in
      add_terminal builder (L.build_cond_br cond then_bb else_bb);
      add_terminal (stmt fn (bae then_bb) then_stmt) (L.build_br merge_bb);
      add_terminal (stmt fn (bae else_bb) else_stmt) (L.build_br merge_bb);
      bae merge_bb
  | A.While (predicate, body) ->
      let pred_bb = abc "while" fn in
      let body_bb = abc "while_body" fn in
      let merge_bb = abc "merge" fn in
      let value = expr fn (bae pred_bb) predicate in
      add_terminal builder (L.build_br pred_bb);
      add_terminal (bae pred_bb) (L.build_cond_br value body_bb merge_bb);
      add_terminal (stmt fn (bae body_bb) body) (L.build_br pred_bb);
      bae merge_bb
  | A.Switch (value, cases) ->
      let merge = abc "merge" fn in
      let switch =
        let value = expr fn builder value in
        L.build_switch value merge (List.length cases) builder in
      let build_case (onval, body) =
        let case = abc "case" fn in
        let body = A.Block body in
        L.add_case switch (expr fn builder onval) case;
        add_terminal (stmt fn (bae case) body) (L.build_br merge) in
      List.iter build_case cases;
      bae merge
  | A.For (name, (start, stop, step), body) ->
      let cond = A.Binop ((A.Variable name), A.Neq, (A.IntLit stop)) in
      let init = A.Expr (A.Assign (name, A.IntLit start)) in
      let increment =
        let value = A.Binop ((A.Variable name), A.Add, (A.IntLit step)) in
        A.Expr (A.Assign (name, value)) in
      let body = A.Block [body; increment] in
      let body = A.Block [init; A.While (cond, body)] in
      stmt fn builder body
  | A.State state ->
      let block, _ = try StringMap.find state !states with
        Not_found -> raise (Bug (Printf.sprintf "No state: %s" state)) in
      let term builder = stmt fn builder (A.Goto state) in
      add_terminal builder term;
      bae block
  | A.Goto state ->
      let _, value = try StringMap.find state !states with
        Not_found -> raise (Bug (Printf.sprintf "No state: %s" state)) in
      debug "goto %s: %d\n" [gsp state builder; L.const_int i32_t value] builder;
      ignore (expr fn builder (A.Assign (L.value_name fn, A.IntLit value)));
      add_terminal builder L.build_ret_void;
```

```
      builder
  | A.Halt ->
      ignore (expr fn builder (A.Assign ("*running", A.IntLit 0)));
      debug "halted %s\n" [gsp (L.value_name fn) builder] builder;
      add_terminal builder L.build_ret_void;
      builder in

(* FSM functions *)
let fsms =
  let build_fsm fsm =
    (* Function initialization *)
    let fn =
      let types = [state_t; state_t; input_t; output_t] in
      let pointers = Array.of_list (List.map L.pointer_type types) in
      let ftype = L.function_type void_t pointers in
      L.define_function fsm.A.fsm_name ftype sake in
    let init = abc "*init" fn and halt = abc "*halt" fn in

    (* Generate mapping of running & state name -> block / index *)
    let add_state m (n, i) = StringMap.add n (abc n fn, i) m in
    states := List.fold_left add_state StringMap.empty fsm.A.fsm_states;

    (* Halt if invalid state; use unique names for blocks *)
    let builder = bae halt in
    let ptr = L.build_struct_gep (L.params fn).(0) 0 "ptr" builder in
    debug "halting from %s\n" [gsp (L.value_name fn) builder] builder;
    ignore (L.build_store zero ptr builder);
    add_terminal builder (L.build_ret_void);

    (* Allocate locals and jump to the correct state *)
    let builder = bae (L.entry_block fn) in
    let add_local m (t, n, e) = (* Local variable allocation *)
      let local = L.build_alloca (lltype t) n builder
      and e = expr fn builder e in
      debug "local %s: %d" [gsp n builder; e] builder;
      ignore (L.build_store e local builder);
      StringMap.add n local m in
    locals := List.fold_left add_local StringMap.empty fsm.A.fsm_locals;
    let bindings = StringMap.bindings !states in
    let switch =
      let state = lookup fn input (L.value_name fn) builder in
      let value = L.build_load state (L.value_name fn) builder in
      debug "state %s: %d\n" [gsp (L.value_name fn) builder; value] builder;
      L.build_switch value halt (List.length bindings) builder in
    let build_case (_, (block, value)) =
      let value = L.const_int i32_t value in
      L.add_case switch value block in
    List.iter build_case (("", (init, 0)) :: bindings);

    (* Build the function body; start with dead, loop in last state *)
    let builder = bae init in
    let body = A.Block fsm.A.fsm_body in
    let builder =
      try
```

```
        let head = fst (List.hd fsm.A.fsm_states) in
        let first = fst (StringMap.find head !states) in
        L.builder_before context (L.build_br first builder) with
      Failure _ -> builder in
    add_terminal (stmt fn builder body) L.build_ret_void;
    fn in
  List.map build_fsm program.A.fsms in

(* Tick function definition *)
let tick =
  let types = [state_t; input_t; output_t] in
  let args = Array.of_list (List.map L.pointer_type types) in
  let ftype = L.function_type (L.pointer_type state_t) args in
  L.define_function (filename ^ "_tick") ftype sake in

(* Metadata, essentially *)
let ta = L.params tick and null = L.const_null (L.pointer_type state_t) in
let reset  = abc "reset"  tick and check  = abc "check"  tick
and update = abc "update" tick and halted = abc "halted" tick in

(* Reset if input is NULL; otherwise, proceed as normal *)
let builder = bae (L.entry_block tick) in
let is_null = L.build_is_null ta.(1) "null" builder in
debug "\ntick started\n" [] builder;
add_terminal builder (L.build_cond_br is_null reset check);

(* Reset *)
let builder = bae reset in
let store i v =
  let ptr = L.build_struct_gep ta.(0) i "reset" builder in
  debug "reset %d, %p: %d\n" [L.const_int i32_t i; ptr; v] builder;
  ignore (L.build_store v ptr builder) in
let l = List.length program.A.fsms + 1 in
let pub_iter i (_, _, e) = store (l + i) (expr tick builder e) in
store 0 pos1; (* the _running variable *)
List.iteri (fun i _ -> store (i + 1) zero) program.A.fsms; (* FSM states *)
List.iteri pub_iter program.A.public; (* public variables *)
add_terminal builder (L.build_ret null);

(* Check if halted *)
let builder = bae check in
let running =
  let ptr = L.build_struct_gep ta.(0) 0 "ptr" builder in
  let run = L.build_load ptr "running" builder in
  L.build_icmp L.Icmp.Ne run zero "run" builder in
debug "check _running: %d\n" [running] builder;
add_terminal builder (L.build_cond_br running update halted);

(* Allocate, initialize, modify, and update FSM state *)
let builder = bae update in
let state = L.build_alloca state_t "state" builder in
let fa = Array.of_list (state :: (Array.to_list ta)) in (* FSM args *)
let m1 = [| state; ta.(0); L.size_of state_t |]
and m2 = [| ta.(0); state; L.size_of state_t |] in
```

```
ignore (L.build_call memcpy m1 "" builder);
List.iter (fun fsm -> ignore (L.build_call fsm fa "" builder)) fsms;
ignore (L.build_call memcpy m2 "" builder);
add_terminal builder (L.build_ret ta.(0));

(* Halted: return 0 iff halted before tick was called *)
add_terminal (bae halted) (L.build_ret null);

(* Enjoy :) *)
sake
```

8.6 Header generator: header_generator.ml

```
(*
 * The function translate converts a Sast.program to a string.
 *
 * Authors: Kai-Zhan Lee, Emma Etherington
 *)

module A = Sast

exception Error_thing

(* generate macro declarations with newlines for all types *)
let macros_of_types name types =
  let rec macros_of_type result = function
    | [] -> result ^ "\n"
    | dtype :: types ->
        let i = ref 0 in
        let macro a v = (* accumulator and value *)
          i := !i + 1; a ^ (Printf.sprintf "#define %s_%s_%s %d\n"
          name dtype.A.type_name v !i) in
        let macro = List.fold_left macro "" dtype.A.type_values in
        macros_of_type (result ^ macro) types in
    macros_of_type "" types

(* generate string of macro declarations for all fsms' state variables *)
let macros_of_fsms name fsms =
  let macros a f =
    let macro a (v, i) =
      a ^ (Printf.sprintf "#define %s_%s_%s %d\n" name f.A.fsm_name v i) in
    a ^ "\n" ^ List.fold_left macro "" f.A.fsm_states in
  List.fold_left macros "" fsms

(* generate macro definitions from named AST *)
let macros_of_ast name ast =
  let types = macros_of_types name ast.A.types in
  let states = macros_of_fsms name ast.A.fsms in
    types ^ "\n" ^ states

let string_of_type = function
  | A.Int -> "int"
  | A.Char -> "char"
  | A.Bool -> "int"
  | A.String -> "char *"
  | A.Enum _ -> "int"

(* generate input struct declarations *)
let input_struct_of_ast name fsms =
  let var_of_tuple (t, n) = Printf.sprintf "\t%s %s;\n" (string_of_type t) n in
  let internals = String.concat "" (List.map var_of_tuple fsms.A.input) in
    Printf.sprintf "struct %s_input {\n%s};\n" name internals
```

```
(* generate output struct declations *)
let output_struct_of_ast name fsms =
  let var_of_tuple (t, n) = Printf.sprintf "\t%s %s;\n" (string_of_type t) n in
  let internals = String.concat "" (List.map var_of_tuple fsms.A.output) in
    Printf.sprintf "struct %s_output {\n%s};\n" name internals

(* generate state struct declarations *)
let state_struct_of_ast name program =
  let var_of_fsm fsm = Printf.sprintf "\tint %s;\n" fsm.A.fsm_name in
  let state_internals = List.map var_of_fsm program.A.fsms in
  let state_internals = String.concat "" state_internals in
  let var_of_public (t, n, _) = "\t" ^ (string_of_type t) ^ " " ^ n ^ ";\n" in
  let fsm_local_vars = List.map var_of_public program.A.public in
  let fsm_local_vars = String.concat ";\n\t" fsm_local_vars in
  Printf.sprintf "struct %s_state {\n\tint _running;\n%s%s};\n"
    name state_internals fsm_local_vars

    (* generate the struct declarations from fsms in ast *)
let structs_of_ast name ast =
  let input_struct = input_struct_of_ast name ast
  and output_struct = output_struct_of_ast name ast
  and state_struct = state_struct_of_ast name ast in
  input_struct ^ "\n" ^ output_struct ^ "\n" ^ state_struct

(* generate prototype of tick function, given a name *)
let tick_prototype name =
  Printf.sprintf "struct %s_state *%s_tick(struct %s_state *, struct %s_input *,
struct %s_output *);\n"
    name name name name name

(* the ifdef ... endif guard *)
let header_guard name macros structs tick =
  let upper = name in
  Printf.sprintf "#ifndef __%s_H__\n#define __%s_H__\n\n%s\n%s\n%s\n#endif\n"
    upper upper macros structs tick

(* convert named AST to header file *)
let translate name ast =
  let macros = macros_of_ast name ast
  and structs = structs_of_ast name ast
  and tick = tick_prototype name in
    header_guard name macros structs tick
```

8.7 AST-SAST restructurer: restruct.ml

```
(*
 * restruct.ml to restructure AST into SAST form
 * and do first round of semantic checking
 *
 * Author: Arunavha Chanda
 *)

module A = Ast
module S = Sast
open Printf

module StringMap = Map.Make(String)

exception SemanticError of string

let wrong_enum_error name =
  let msg = sprintf "undeclared enum value %s" name in
  raise (SemanticError msg)

let convert_type = function (* A.dtype *)
  | A.Bool -> S.Bool
  | A.Int -> S.Int
  | A.Char -> S.Char
  | A.String -> S.String
  | A.Enum(name) -> S.Enum(name)

let get_uop = function (* A.uop *)
  | A.Neg -> S.Neg
  | A.Not -> S.Not

let get_op = function (* A.op *)
  | A.Add -> S.Add
  | A.Sub -> S.Sub
  | A.Mul -> S.Mul
  | A.Div -> S.Div
  | A.Eq -> S.Eq
  | A.Neq -> S.Neq
  | A.Lt -> S.Lt
  | A.Le -> S.Le
  | A.Gt -> S.Gt
  | A.Ge -> S.Ge
  | A.And -> S.And
  | A.Or -> S.Or

let rec find_val vl ind = function (* start at 1 *)
  | [] -> (-1)
  | [x] -> if(x=vl) then ind else find_val vl (ind+1) []
  | x::tl -> if(x=vl) then ind else find_val vl (ind+1) tl

let look_for vl type_dec=
```

```
  find_val vl 1 type_dec.A.type_values

let rec is_there_res = function
  | [] -> (-1)
  | [x] -> if(x = (-1)) then is_there_res [] else x
  | x::tl -> if(x = (-1)) then is_there_res tl else x

let rec look_in_states vl = function
  | [] -> (-1)
  | [(name,num)] -> if (name=vl) then num else look_in_states vl []
  | (name,num)::tl -> if (name=vl) then num else look_in_states vl tl

let rec get_expr sts program = function (* A.expr *)
  | A.BoolLit(bl) -> S.BoolLit(bl)
  | A.CharLit(ch) -> S.CharLit(ch)
  | A.IntLit(num) -> S.IntLit(num)
  | A.StringLit(name) -> S.StringLit(name)
  | A.Variable(name) -> S.Variable(name)
  | A.EnumLit(vl) ->
    let result =
      let enum_search = List.map (look_for vl) program.A.types in
      is_there_res enum_search in
    if (result <> (-1))
      then S.IntLit(result)
      else
        let is_state = look_in_states vl sts in
        if (is_state <> (-1)) then S.IntLit(is_state) else (wrong_enum_error vl)
  | A.Access (outer,inner) -> S.Variable(outer ^ "_" ^ inner)
  | A.Uop(u,exp) -> S.Uop((get_uop u),(get_expr sts program exp))
  | A.Binop(e1,o,e2) -> S.Binop((get_expr sts program e1), (get_op o) ,(get_expr sts
program e2))
  | A.Assign(name,exp) -> S.Assign(name,(get_expr sts program exp))
  | A.Printf(fmt, lst) -> S.Printf(fmt, (get_e_list sts program lst))
  | A.Empty -> S.Empty
and get_e_list sts program = function (* expr list *)
  [] -> []
  | exp::tl -> (get_expr sts program exp)::(get_e_list sts program tl)

let rec do_stmt sts program = function (* stmts *)
  | A.Block(s_list) -> S.Block(take_stmts sts program s_list)
  | A.State(name) -> S.State(name)
  | A.If(pred,sta,stb) -> S.If((get_expr sts program pred),(do_stmt sts program
sta),(do_stmt sts program stb))
  | A.For(str,na,nb,nc,stm) -> S.For(str,(na,nb,nc),(do_stmt sts program stm))
  | A.While(pred,stm) -> S.While((get_expr sts program pred),(do_stmt sts program
stm))
  | A.Switch(exp, cases) -> S.Switch((get_expr sts program exp),(get_cases sts program
cases))
  | A.Expr(e) -> S.Expr(get_expr sts program e)
  | A.Goto(label) -> S.Goto(label)
  | A.Halt -> S.Halt
and take_stmts sts program = function (* stmt list *)
  [] -> []
  | stm::tl -> (do_stmt sts program stm)::(take_stmts sts program tl)
```

```
and get_cases sts program = function (* (expr * stmt) list *)
  [] -> []
  | (e,s_list)::tl -> ((get_expr sts program e),(take_stmts sts program
s_list))::(get_cases sts program tl)

let rec take_in = function
  [] -> []
  | (typ,name)::tl -> ((convert_type typ),name)::(take_in tl)

let rec take_out = function
  [] -> []
  | (typ,name)::tl -> ((convert_type typ),name)::(take_out tl)

let rec take_typ = function
  [] -> []
  | {A.type_name = name; A.type_values=vals}::tl -> {S.type_name = name; S.type_values
= vals}::(take_typ tl)

let rec copy_locals sts program = function (* (dtype * string * expr) list *)
  [] -> []
  | (typ,var_name,expr)::tl -> ((convert_type typ),var_name,(get_expr sts program
expr)):: (copy_locals sts program tl)

let rec get_states num = function (* body: stmt list *)
  [] -> []
  | A.State(name)::tl -> (name,num):: (get_states (num+1) tl)
  | _::tl -> get_states num tl

let rec take_fsm sts program = function
  [] -> []
  | {A.fsm_name = name; A.fsm_public = _ ; A.fsm_locals = local; A.fsm_body =
body}::tl
      -> { S.fsm_name = name; S.fsm_locals = (copy_locals sts program local);
S.fsm_states = (get_states 1 body); S.fsm_body = (take_stmts sts program
body)}::(take_fsm sts program tl)

let rec take_pubs sts program name = function (* (dtype * string * expr) list *)
  [] -> []
  | (typ,var_name,expr)::tl -> ((convert_type typ),name ^ "_" ^ var_name,(get_expr sts
program expr)):: (take_pubs sts program name tl)

let rec get_pubs sts program = function
  [] -> []
  | {A.fsm_name = name; A.fsm_public = pubs; A.fsm_locals = _ ; A.fsm_body =  _ }::tl
      -> (take_pubs sts program name pubs) @ (get_pubs sts program tl)

let rec muddle_it_all = function
  | [] -> []
  | [l1] -> l1
  | l1::tl -> l1 @ muddle_it_all tl

let yank_out_states fsm =
  get_states 1 fsm.A.fsm_body
```

```
let get_all_states fsms =
  let state_fam = List.map (yank_out_states) fsms in
  muddle_it_all state_fam

let convert i o typs fsms program = function
  | _ -> let all_sts = get_all_states fsms in {S.input = take_in i; S.output =
take_out o; S.public = get_pubs [("",0)] program fsms; S.types = take_typ typs; S.fsms
= take_fsm all_sts program fsms}

let transform program =
  convert program.A.input program.A.output program.A.types program.A.fsms program []
```

8.8 Semantic checker: semant.ml

```
(*
 * Main semantic checker
 *
 * Author: Arunavha Chanda
 *)

module A = Ast
module S = Sast
open Printf

module StringMap = Map.Make(String)

exception SemanticError of string

let string_of_type = function
  | S.Bool -> "Bool"
  | S.Int -> "Int"
  | S.Char -> "Char"
  | S.String -> "String"
  | _ -> "other"

let rec print_list = function
  [] -> ()
  | (s,t)::l -> print_string "(" ; print_string s ; print_string "," ; print_string
(string_of_type t) ; print_string ")" ; print_string " " ; print_list l

let rec find_variable scope name =
  try List.find (fun (s, _) -> s = name) scope.S.variables
  with Not_found ->
    match scope.parent with
      | Some(parent) -> find_variable parent name
      | _ -> raise Not_found

let require_integer e msg =
  if (e = S.Int) then () else raise (SemanticError msg)

let report_duplicate exceptf list =
  let rec helper = function
    | n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
    | _ :: t -> helper t
    | [] -> ()
  in helper (List.sort compare list)

let undeclared_identifier_error name =
    let msg = sprintf "undeclared identifier %s" name in
    raise (SemanticError msg)

let illegal_assignment_error = function
  | _ ->  let msg = sprintf "illegal assignment" in
    raise (SemanticError msg)
```

```
let illegal_unary_operation_error = function
  | _ ->  let msg = sprintf "illegal unary operator" in
    raise (SemanticError msg)

let illegal_binary_operation_error = function
  | _ ->  let msg = sprintf "illegal binary operator" in
    raise (SemanticError msg)

let check_assign lvaluet rvaluet = match lvaluet with
  | S.Bool when rvaluet = S.Int -> lvaluet
  | S.Enum(_) when  rvaluet = S.Int -> lvaluet
  | _ -> if lvaluet == rvaluet then lvaluet else illegal_assignment_error []

(* Checking Global Variables *)

let check_globals inp outp env =
  let globals = inp @ outp in
  report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd globals);
  List.fold_left (fun lst (typ,name) -> (name,typ)::lst) env.S.scope.variables globals

let check_fsm_decl fsms =
  report_duplicate (fun n -> "duplicate fsm " ^ n) (List.map (fun fd -> fd.S.fsm_name)
fsms)

let check_enums types =
  report_duplicate (fun n -> "duplicate type " ^ n )
    (List.map (fun t -> t.S.type_name) types);
  List.map (fun lst -> report_duplicate (fun n -> "duplicate type " ^ n ) lst)
    (List.map (fun t -> t.S.type_values) types)

let add_local_vars vars env =
  report_duplicate (fun n -> "duplicate local " ^ n )
    (List.map (fun (_,s,_) -> s) vars);
  List.fold_left (fun lst (typ,name,_) -> (name,typ)::lst) env.S.scope.variables vars

let check_pubs pubs env =
  report_duplicate (fun n -> "duplicate public " ^ n )
    (List.map (fun (_,s,_) -> s) pubs);
  List.fold_left (fun lst (typ,name,_) -> (name,typ)::lst) env.S.scope.variables pubs

let type_of_identifier fsm scope name =
  let vdecl = try find_variable scope name
  with Not_found ->
    try find_variable scope (fsm.S.fsm_name ^ "_" ^ name)
    with Not_found ->
      undeclared_identifier_error name
  in
  let (_,typ) = vdecl in
  typ

let rec get_expr fsm env = function (* A.expr *)
  | S.BoolLit(_) -> S.Bool
  | S.CharLit(_) -> S.Char
```

```
  | S.IntLit(_) -> S.Int
  | S.StringLit(_) -> S.String
  | S.Variable(name) ->
    let var = try find_variable env.S.scope name
    with Not_found ->
      try find_variable env.S.scope (fsm.S.fsm_name ^ "_" ^ name)
      with Not_found ->
          raise (SemanticError("undeclared identifier " ^ name))
    in
    let (_,vl) = var in
    vl
  | S.Uop(op, e) ->
      let t = get_expr fsm env e in
      (match op with
        | S.Neg when t = S.Int -> S.Int
        | S.Not when t = S.Bool -> S.Bool
        | _ -> illegal_unary_operation_error []
      )
  | S.Binop(e1,op,e2) ->
      let t1 = get_expr fsm env e1
      and t2 = get_expr fsm env e2 in
      ( match op with
        | S.Add | S.Sub | S.Mul | S.Div when t1 = S.Int && t2 = S.Int -> S.Int
        | S.Eq | S.Neq | S.Lt | S.Le | S.Gt | S.Ge when t1 = t2 -> S.Bool
        | S.Eq | S.Neq | S.Lt | S.Le | S.Gt | S.Ge when t1 = S.Int || t2 = S.Int ->
S.Bool
        | S.And | S.Or when t1 = S.Bool && t2 = S.Bool -> S.Bool
        | _  -> illegal_binary_operation_error []
      )
  | S.Assign(name,exp) ->
      let lt = type_of_identifier fsm env.scope name
      and rt = get_expr fsm env exp in
      check_assign lt rt
  | S.Printf(_, lst) -> ignore(List.map (get_expr fsm env) lst); S.Int
  | S.Empty -> S.Int

let rec check_stmt env fsm = function (* stmts *)
  | S.Block(s_list) ->
    let sl =
      let env' =
        let scope' = { S.parent = Some(env.S.scope); S.variables =
env.S.scope.variables } in
        { S.scope = scope' } in
      List.map (fun s -> check_stmt env' fsm s) s_list in
    ignore(sl);
  | S.State(_) -> ()
  | S.If(pred,sta,stb) ->
      let e = get_expr fsm env pred in
      ignore((match e with
        | S.Int | S.Bool -> ()
        | _ -> raise (SemanticError("Illegal predicate type"))));
      ignore(check_stmt env fsm sta); ignore(check_stmt env fsm stb)
  | S.For(str,(na,nb,nc),stm) ->
    ignore(
```

```
      try List.find (fun (s, _) -> s = str) env.S.scope.variables
      with Not_found ->
        ignore(env.S.scope.variables <- (str,Int)::env.S.scope.variables);
(str,S.Int));
      ignore(require_integer (get_expr fsm env (S.IntLit(na))) "Non-integer used in
for loop");
      ignore(require_integer (get_expr fsm env (S.IntLit(nb))) "Non-integer used in
for loop");
      ignore(require_integer (get_expr fsm env (S.IntLit(nc))) "Non-integer used in
for loop");
      (check_stmt env fsm stm)
  | S.While(pred,stm) ->
      let e = get_expr fsm env pred in
      ignore((match e with
        | S.Int | S.Bool -> ()
        | _ -> raise (SemanticError("Illegal predicate type"))));
      ignore(check_stmt env fsm stm)
  | S.Switch(exp, cases) ->
      ignore(get_expr fsm env exp);
      ignore(check_cases env fsm cases)
  | S.Expr(e) -> ignore (get_expr fsm env e)
  | S.Goto(label) ->
    ignore(
      try List.find (fun (s,_) -> s=label) fsm.S.fsm_states
      with Not_found -> raise (SemanticError "No such state exists"))
  | S.Halt -> ()

and check_cases env fsm = function (* (expr * stmt) list *)
  | [] -> ()
  | (e,s_list)::tl -> ignore(get_expr fsm env e);
    ignore(
    let sl =
      let env' =
        let scope' = { S.parent = Some(env.S.scope); S.variables =
env.S.scope.variables } in
        { S.scope = scope' } in
      List.map (fun s -> check_stmt env' fsm s) s_list in
    sl); ignore(check_cases env fsm tl)

let check_fsm_locals fsm env =
  (* Check FSM INSTANCE VARS: public and states *)
  report_duplicate (fun n -> "duplicate state " ^ n ^ " in " ^ fsm.S.fsm_name)
    (List.map fst fsm.S.fsm_states);
  report_duplicate (fun n -> "duplicate local " ^ n ^ " in " ^ fsm.S.fsm_name)
    (List.map (fun (_,s,_) -> s) fsm.S.fsm_locals);
  List.map (fun (typ,_,exp) -> check_assign (typ) (get_expr fsm env exp) )
fsm.S.fsm_locals

let check_body env fsm =
  check_stmt env fsm (S.Block(fsm.fsm_body))

let check_semant env fsm =
  let env' =
```

```
    let local_sym = { env.S.scope with variables = (add_local_vars fsm.S.fsm_locals
env) @ env.S.scope.variables} in
    { S.scope = local_sym } in
  ignore(check_fsm_locals fsm env); ignore(check_body env' fsm)

let check program =
  let all_fsm_names = List.map (fun fsm_dec -> (fsm_dec.S.fsm_name,S.Int) )
program.S.fsms in
  let sym_tab = {S.parent = None; S.variables = all_fsm_names } in
  let env = {S.scope=sym_tab} in
  let new_syms = {sym_tab with variables = check_globals program.S.input
program.S.output env} in
  let new_syms1 = {new_syms with variables = (check_pubs program.S.public env) @
(new_syms.S.variables)} in
  let env2 = { S.scope=new_syms1} in
  ignore(check_enums program.S.types);
  ignore(check_fsm_decl program.S.fsms);
  ignore(List.iter (check_semant env2) program.S.fsms)
```

8.9 Main controller: sake.ml

```
(* Generating the C header file and C++ LLVM code *)
(* Author: Emma Etherington *)
let name = Sys.argv.(2) in
  let header_name = name ^ ".h" and llvm_name = name ^ ".ll" in
  let in_channel = open_in Sys.argv.(1) in
  let lexbuf = Lexing.from_channel in_channel in
  let ast = Parser.program Scanner.token lexbuf in
  let sast = Restruct.transform ast in
  Semant.check sast;
  let header = Header_generator.translate name sast (* sast *)
     and llvm = Llvm_generator.translate name sast (* sast *) in
      Printf.fprintf (open_out header_name) "%s" header;
      Llvm_analysis.assert_valid_module llvm;
      Printf.fprintf (open_out llvm_name) "%s" (Llvm.string_of_llmodule llvm);
```

8.10 Main test suite script: testall.sh

```
# Main test script
# Author: Emma Etherington

#!/bin/sh

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="gcc"

# Path to sake compiler - usually just ./sake.native
#SAKE="./sake"
SAKE="_build/sake.native"

# Set time limit for all operations
ulimit -t 30

globallog=alltests.log
rm -f $globallog
error=0
globalerror=0
keep=0

# usage
Usage() {
    echo "Usage: tests.sh [.sk file]"
    echo "-k    Keep the intermediate files"
    exit 1
}

# SignalError()
SignalError() {
    if [ $error -eq 0 ] ; then
        echo "FAILED"
        error=1
    fi
    echo "  $1"
}

# Comparing the generated with expected
Compare() {
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 ">" $3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
        SignalError "$1 differs"
```

```
        echo "FAILED $1 differs from $2" 1>&2
    }
}

# Run functions -> how we want run it and then report errors
Run() {
    echo $* 1>&2
    #echo $*
    eval $* || {
        SignalError "$1 failed on $*"
        return 1
    }
}

RunFail() {
    echo $* 1>&2
    eval $* && {
        SignalError "Failed: $* did not report an error"
        return 1
    }
    return 0
}

# Check functions -> should be calling run() funcs and compare() funcs
Check() {

    error=0
    basename=`echo $1 | sed 's/.*\\///
                            s/.sk//'`
    reffile=`echo $1 | sed 's/.sk$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    #echo $wrapper
    #echo "../testing/$wrapper"

    echo -n "$basename..."
    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    if [ ! -f "../testing/${basename}.c" ]; then
        #error=2
        #echo "FAILED NO WRAPPER"

        echo "#include <stdio.h>" > ../testing/${basename}.c
        echo "#include <stdlib.h>" >> ../testing/${basename}.c
        echo "#include <unistd.h>" >> ../testing/${basename}.c
        echo "#include \"${basename}.h\"\n" >> ../testing/${basename}.c
        echo "int main() {\n\tstruct ${basename}_input i;" >> ../testing/${basename}.c
        echo "\tstruct ${basename}_state s;" >> ../testing/${basename}.c
        echo "\n\t${basename}_tick(&s, NULL, NULL); \n\t${basename}_tick(&s, &i,
NULL); \n\n\treturn 0;\n}" >> ../testing/${basename}.c
```

```
        generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe
${basename}.out ${basename}.o" &&
        Run "$SAKE" " " $1 ${basename} &&
        Run "mv ${basename}.h ../testing/" &&
        Run "$LLC" "${basename}.ll" ">" "${basename}.s" &&
        Run "$CC" "-c" "../testing/${basename}.c ../testing/${basename}.h" &&
        Run "$CC" "-o" "${basename}.exe" "${basename}.s" "${basename}.o" "print.o" &&
        Run "./${basename}.exe" > "${basename}.out" &&
        Compare ${basename}.out ${reffile}.out ${basename}.diff
    else
        generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe
${basename}.out ${basename}.o" &&
        Run "$SAKE" " " $1 ${basename} &&
        Run "mv ${basename}.h ../testing/" &&
        Run "$LLC" "${basename}.ll" ">" "${basename}.s" &&
        Run "$CC" "-c" "../testing/${basename}.c ../testing/${basename}.h" &&
        Run "$CC" "-o" "${basename}.exe" "${basename}.s" "${basename}.o" "print.o" &&
        Run "./${basename}.exe" > "${basename}.out" &&
        Compare ${basename}.out ${reffile}.out ${basename}.diff
    fi


    # Report the status and clean up the generated files
    if [ $error -eq 0 ] ; then
        if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
        fi
        echo "OK"
        echo "###### SUCCESS" 1>&2
    else
        echo "###### FAILED" 1>&2
        globalerror=$error
    fi
}

CheckFail() {

    error=0
    basename=`echo $1 | sed 's/.*\\///
                              s/.sk//'`
    reffile=`echo $1 | sed 's/.sk$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
    RunFail "$SAKE" " " $1 ${basename} "2>" "${basename}.err" ">>" $globallog &&
    Compare ${basename}.err ${reffile}.err ${basename}.diff
```

```
    # Report the status and clean up the generated files
    if [ $error -eq 0 ] ; then
        if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
        fi
        echo "OK"
        echo "###### SUCCESS" 1>&2
    else
        echo "###### FAILED" 1>&2
        globalerror=$error
    fi
}

# CHECK FOR FLAGS
while getopts kdpsh c; do
    case $c in
        k) # Keep intermediate files
            keep=1
            ;;
    esac
done

shift `expr $OPTIND - 1`

LLIFail() {
    echo "Could not find the LLVM interpreter \"$LLI\"."
    echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
    exit 1
}

which "$LLI" >> $globallog || LLIFail

if [ ! -f print.o ]
then
    echo "Could not find print.o"
    echo "Try \"make print.o\""
    exit 1
fi

# CODE TO GET THE TEST FILES

if [ $# -ge 1 ]
then
    files=$@
else
    #Check this path
    files="../testing/test_*.sk ../testing/fail_*.sk"
fi

# RUN CHECKS

for file in $files
do
    case $file in
```

```
        *test_*TL*)
            ;; # Don't run the traffic light programs becuase they have sleeps in
        *test_*)
            Check $file 2>> $globallog
            ;;
        *fail_*)
            CheckFail $file 2>> $globallog
            ;;
        *)
            echo "unkown file type $file"
            globalerror=1
            ;;
    esac
done


exit $globalerror
```

8.11 Traffic test script: traffic.sh

```
# Traffic test script
# Author: Emma Etherington

#!/bin/sh

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="gcc"

# Path to sake compiler - usually just ./sake.native
#SAKE="./sake"
SAKE="_build/sake.native"

# Set time limit for all operations
ulimit -t 30

globallog=trafficLights.log
rm -f $globallog
error=0
globalerror=0
keep=0

# usage
Usage() {
    echo "Usage: tests.sh [.sk file]"
    echo "-k    Keep the intermediate files"
    exit 1
}

# SignalError()
SignalError() {
    if [ $error -eq 0 ] ; then
        echo "FAILED"
        error=1
    fi
    echo "  $1"
}

# Comparing the generated with expected
Compare() {
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 ">" $3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
        SignalError "$1 differs"
```

```
        echo "FAILED $1 differs from $2" 1>&2
    }
}

# Run functions -> how we want run it and then report errors
Run() {
    echo $* 1>&2
    #echo $*
    eval $* || {
        SignalError "$1 failed on $*"
        return 1
    }
}

# Check functions -> should be calling run() funcs and compare() funcs
Check() {

    error=0
    basename=`echo $1 | sed 's/.*\\///
                            s/.sk//'`
    reffile=`echo $1 | sed 's/.sk$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo "$basename..."
    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    if [ ! -f "../testing/${basename}.c" ]; then
        error=2
        echo "FAILED - NO WRAPPER"
    else
        generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe
${basename}.out ${basename}.o" &&
        Run "$SAKE" " " $1 ${basename} &&
        Run "mv ${basename}.h ../testing/" &&
        Run "$LLC" "${basename}.ll" ">" "${basename}.s" &&
        Run "$CC" "-c" "../testing/${basename}.c ../testing/${basename}.h" &&
        Run "$CC" "-o" "${basename}.exe" "${basename}.s" "${basename}.o" "print.o" &&
        Run "./${basename}.exe" &&
        Run "./${basename}.exe" > "${basename}.out" &&
        Compare ${basename}.out ${reffile}.out ${basename}.diff
    fi


    # Report the status and clean up the generated files
    if [ $error -eq 0 ] ; then
        if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
        fi
        echo "OK\n"
        echo "###### SUCCESS" 1>&2
    else
```

```
        echo "###### FAILED" 1>&2
        globalerror=$error
    fi
}

# CHECK FOR FLAGS
while getopts kdpsh c; do
    case $c in
        k) # Keep intermediate files
            keep=1
            ;;
    esac
done

shift `expr $OPTIND - 1`

LLIFail() {
    echo "Could not find the LLVM interpreter \"$LLI\"."
    echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
    exit 1
}

which "$LLI" >> $globallog || LLIFail

if [ ! -f print.o ]
then
    echo "Could not find print.o"
    echo "Try \"make print.o\""
    exit 1
fi

# CODE TO GET THE TEST FILES

if [ $# -ge 1 ]
then
    files=$@
else
    #Check this path
    files="../testing/test_*TL.sk"
fi

# RUN CHECKS

for file in $files
do
    case $file in
        *test_*)
            Check $file 2>> $globallog
            ;;
        *)
            echo "unknown file type $file"
            globalerror=1
            ;;
    esac
```

```
done

exit $globalerror
```

8.12 Adventure test script: adventure.sh

```
# Adventure test script
# Author: Emma Etherington

#!/bin/sh

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="gcc"

# Path to sake compiler - usually just ./sake.native
#SAKE="./sake"
SAKE="_build/sake.native"

# Set time limit for all operations
ulimit -t 30

globallog=adventureStory.log
rm -f $globallog
error=0
globalerror=0
keep=0

# usage
Usage() {
    echo "Usage: tests.sh [.sk file]"
    echo "-k    Keep the intermediate files"
    exit 1
}

# SignalError()
SignalError() {
    if [ $error -eq 0 ] ; then
        echo "FAILED"
        error=1
    fi
    echo "  $1"
}

# Run functions -> how we want run it and then report errors
Run() {
    echo $* 1>&2
    #echo $*
    eval $* || {
        SignalError "$1 failed on $*"
```

```
        return 1
    }
}

# Check functions -> should be calling run() funcs and compare() funcs
Check() {

    error=0
    basename=`echo $1 | sed 's/.*\\///
                            s/.sk//'`
    reffile=`echo $1 | sed 's/.sk$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo "$basename..."
    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe
${basename}.out ${basename}.o" &&

    Run "$SAKE" " " $1 ${basename} &&
    Run "mv ${basename}.h ../testing/" &&
    Run "$LLC" "${basename}.ll" ">" "${basename}.s" &&
    Run "$CC" "-c" "../testing/${basename}.c ../testing/${basename}.h" &&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "${basename}.o" "print.o" &&
    Run "./${basename}.exe"

    # Report the status and clean up the generated files
    if [ $error -eq 0 ] ; then
        if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
        fi
             echo
        echo "The adventure is over :)"
        echo "###### SUCCESS" 1>&2
    else
             echo
        echo "The adventure is over :("
        echo "###### FAILED" 1>&2
        globalerror=$error
    fi
}

# CHECK FOR FLAGS
while getopts kdpsh c; do
    case $c in
        k) # Keep intermediate files
            keep=1
            ;;
    esac
done
```

```
shift `expr $OPTIND - 1`

LLIFail() {
    echo "Could not find the LLVM interpreter \"$LLI\"."
    echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
    exit 1
}

which "$LLI" >> $globallog || LLIFail

if [ ! -f print.o ]
then
    echo "Could not find print.o"
    echo "Try \"make print.o\""
    exit 1
fi

# CODE TO GET THE TEST FILES

if [ $# -ge 1 ]
then
    files=$@
else
    #Check this path
    files="../testing/adventure.sk"
fi

# RUN CHECKS

for file in $files
do
    case $file in
        *adventure*)
            Check $file 2>> $globallog
            ;;
        *)
            echo "unknown file type $file"
            globalerror=1
            ;;
    esac
done

exit $globalerror
```

8.13 TEST CASES: Source Files

./list_tests.sh:

```
    #!/bin/bash
    # GNU bash, version 4.4.12(1)-release (x86_64-apple-darwin15.6.0)
    #
    # A script for printing out code listings for the following files:
    # test_*.{c,sk,out} and fail_*.{sk,err}
    #
    # Author: Kai-Zhan Lee

    # Check if files exist
    function check {
       for file in $@; do
              [ ! -e "$file" ] && >&2 echo "Houston, we've got a problem" && exit
       done
    }

    # Print the code listing of a single file; exit if any are not present
    function list {
       check $@
       for file in $@; do
              echo "$file:"
              cat $file | sed 's/^/    /'
              echo
       done
    }

    # Print the script itself (it's in the testing directory, after all!)
    list $0

    # Print listings for adventure
    list adventure.{sk,c}

    # Print listings for traffic light FSMs
    for file in test_*TL.sk; do list "${file%.*}".{sk,c,out}; done

    # Print listings for positive test files
    GLOBIGNORE="*TL.sk"
    for file in test_*.sk; do list "${file%.*}".{sk,c,out}; done
    unset GLOBIGNORE

    # Print listings for negative fail files
    for file in fail_*.sk; do list "${file%.*}".{sk,err}; done
```

adventure.sk:

```
    input[char decision]
    output[int result]

    fsm ekas {

            state Sone
                    if (decision == 's') {
                            printf("%s\n", "Welcome to the story of the PLT
presentation :)")
                            printf("%s\n\n", "If at anytime you wish to leave, simply
press e instead of having the courage to make a choice!")
                            printf("%s\n%s\n\n", "It's May 9th, and your PLT
presentation is due tomorrow. Suddenly, you realize that your language", "sucks. What
do you do?")
                            printf("%s\n", "a. Go to Edwards and beg for an
extension.")
                            printf("%s\n\n", "b. Sell your soul to Satan in the hopes
he will help you pass.")
                            result = 1
                            goto Stwo
                    }
                    else if (decision == 'e') {
                            printf("\n%s\n", "Looks like you are leaving us. Guess you
just couldn't handle it.")
                            result = 2
                            halt
                    }
                    else {
                            printf("%s\n", "Don't just sit there. Get moving!")
                            result = 0
                            goto Sone
                    }
            state Stwo
                    if (decision == 'a') {
                            printf("%s\n", "Seriously? You're asking me for an
extension? Seriously?, Edwards says with a look of incredulity.")
                            printf("%s\n", "I, ummm, I, nope, jk. Bye!!! You run out
the room.")
                            printf("%s\n\n", "What do you do next?")
                            printf("%s\n", "a. Give up, go back to your room, and fall
asleep.")
                            printf("%s\n%s\n\n", "b. Kidnap all your groupmates, lock
them in room until the presentation, and only feed them if", "they code.")
                            result = 1
                            goto Sthree
                    }
                    else if (decision == 'b') {
```

```
                        printf("%s\n%s\n\n", "You go to Central Park Zoo to kidnap
the nearest goat to make your blood sacrifice. Unfortunately", "the gate is locked!
What should you do?")
                        printf("%s\n", "a. With your PLT skillz you think you can
hack into the security system and open the doors.")
                        printf("%s\n", "b. Give up and decide to beg Edwards for
an extension.")
                        printf("%s\n\n", "c. Decide to sacrifice one of your group
mates instead.")
                        result = 1
                        goto Sfour
                }
                else if (decision == 'e') {
                        printf("\n%s\n", "Looks like you are leaving us. Guess you
just could not handle it.")
                        halt
                }
                else if (decision == '(') {
                        goto Stwo
                }
                else {
                        printf("%s\n", "Don't just sit there. Get moving!")
                        result = 1
                        goto Stwo
                }
        state Sthree
                if (decision == 'a') {
                        printf("%s\n", "It turns out, your group mates also made
the same decision you did. In light of your stupidity in")
                        printf("%s\n", "asking for an extension and rubbish
project, Edwards gives you all a B-. He was not able to give")
                        printf("%s\n", "you a lower grade because you actually did
ok on the homeworks and exams.")
                        halt
                }
                else if (decision == 'b') {
                        printf("%s\n", "You've kidnapped all your groupmates and
locked them in your room. But for some reason they do not")
                        printf("%s\n\n", "seem to be getting much done! What do
you do?")
                        printf("%s\n", "a. Offer them Sake to get the creative
juices flowing.")
                        printf("%s\n%s\t\n", "b. Decide that it’s better to
sacrifice one of them to Satan. After all, what good are they if", "they cannot
work?")
                        printf("%s\n\n", "c. Give a motivational speech based off
the one in the movie Independence Day.")
                        result = 1
```

```
                        goto Sfive
                }
                else if (decision == 'e') {
                        printf("\n%s\n", "Looks like you are leaving us. Guess you
just couldn't handle it.")
                        halt
                }
                else if (decision == '(') {
                        goto Sthree
                }
                else {
                        printf("%s\n", "Don't just sit there. Get moving!")
                        result = 1
                        goto Sthree
                }
        state Sfour
                if (decision == 'a') {
                        printf("%s\n", "Realizing that PLT is not a security class
and that you actually suck at hacking into anything.")
                        printf("%s\n%s\n\n", "You get caught within thirty seconds
of trying to break in. As the police question you,", "what do you do?")
                        printf("%s\n", "a. Admit everything")
                        printf("%s\n\n", "b. Admit nothing")
                        result = 1
                        goto Ssix
                }
                else if (decision == 'b') {
                        printf("%s\n", "Seriously? You're asking me for an
extension? Seriously?, Edwards says with a look of incredulity.")
                        printf("%s\n\n", "I, ummm, I, nope, jk. Bye!!! You run out
the room. What do you do next?")
                        printf("%s\n", "a. Give up, go back to your room, and fall
asleep.")
                        printf("%s\n%s\n\n", "b. Kidnap all your groupmates, lock
them in room until the presentation, and only feed them if", "\tthey code.")
                        result = 1
                        goto Sthree
                }
                else if (decision == 'c') {
                        printf("%s\n\n", "But which group mate to sacrifice?")
                        printf("%s\n", "a. Shalva")
                        printf("%s\n", "b. Shalva")
                        printf("%s\n\n", "c. Shalva")
                        result = 1
                        goto Sseven
                }
                else if (decision == 'e') {
```

```
                        printf("\n%s\n", "Looks like you are leaving us. Guess you
just couldn't handle it.")
                        halt
                }
                else if (decision == '(') {
                        goto Sfour
                }
                else {
                        printf("%s\n", "Don't just sit there. Get moving!")
                        result = 1
                        goto Sfour
                }
        state Sfive
                if (decision == 'a') {
                        printf("%s\n", "One of your group mates refuses to drink
because he’s under age. You ignore his attempts to sway")
                        printf("%s\n\n", "the group against drinking too and go
ahead. Pleasantly drunk, your group decides to...")
                        printf("%s\n", "a. Spend the rest of the night watching
the anime Evangelion as homage to Japan (and Edwards).")
                        printf("%s\n", "b. Go to starbucks, waste all their
respective tuition payments on Venti Vanilla Lattes and in")
                        printf("%s\n\n", "\tthe midst of a caffeine buzz work on
the project all night.")
                        result = 1
                        goto Seight
                }
                else if (decision == 'b') {
                        printf("%s\n\n", "But which group mate to sacrifice?")
                        printf("%s\n", "a. Shalva")
                        printf("%s\n", "b. Shalva")
                        printf("%s\n\n", "c. Shalva")
                        result = 1
                        goto Sseven
                }
                else if (decision == 'c') {
                        printf("%s\n", "Your teammates are inspired. You work all
night. You get a ton done, and the next morning realise")
                        printf("%s\n", "your language does not suck after all. As
you present to Edwards, he is impressed and surprised" )
                        printf("%s\n", "you actually managed to survive the
semester. He had serious doubts. As he looks on, you feel proud")
                        printf("%s\n", "and happy it's all over. Then your face
drops as you all realise you have more finals. But on the")
                        printf("%s\n", "plus side you got a good grade.")
                        halt
                }
                else if (decision == 'e') {
```

```
                        printf("\n%s\n", "Looks like you are leaving us. Guess you
just couldn't handle it.")
                        halt
                }
                else if (decision == '(') {
                        goto Sfive
                }
                else {
                        printf("%s\n", "Don't just sit there. Get moving!")
                        result = 1
                        goto Sfive
                }
        state Ssix
                if (decision == 'a') {
                        printf("%s\n", "After looking at sorry looking college
student in front of them, the police decide to let you go.")
                        printf("%s\n\n", "As you are walking back to college you
debate your options. What do you do?")
                        printf("%s\n", "a. Decide to write a tell all book about
your PLT adventures.")
                        printf("%s\n", "b. Decide to sacrifice one of your group
mates instead.")
                        printf("%s\n%s\n", "c. Try to break into the Zoo again
because you think Satan will like a goat sacrifice more than","a human one.")
                        printf("%s\n%s\n\n", "d. Kidnap all your groupmates, lock
them in room until the presentation, and only feed them if", "they code.")
                        result = 1
                        goto Sten
                }
                else if (decision == 'b') {
                        printf("%s\n%s\n", "The police, taking none too kindly to
the silent college student in front of them, decide to lock", "you up for the night.")
                        printf("%s\n\n", "The next morning they let you out right
before your presentation. What do you do?")
                        printf("%s\n", "a. Ditch your group mates.")
                        printf("%s\n\n", "b. Decide to go to the presentation and
help them present your sucky language.")
                        result = 1
                        goto Seleven
                }
                else if (decision == 'e') {
                        printf("\n%s\n", "Looks like you are leaving us. Guess you
just couldn't handle it.")
                        halt
                }
                else if (decision == '(') {
                        goto Ssix
                }
```

```
                else {
                        printf("%s\n", "Don't just sit there. Get moving!")
                        result = 1
                        goto Ssix
                }
        state Sseven
                if (decision == 'a' || decision == 'b' || decision == 'c') {
                        printf("%s\n", "Satan looks at Shalva and denies to take
the offering. He says that she is already spreading evil in the world and ")
                        printf("%s\n", "that there is no need to sacrifice her.
But he commends your ability to sacrifice your loved ones. He ")
                        printf("%s\n\n", "gives you the choice of two gifts in
acknowledgement of your heartlessness. ")
                        printf("%s\n\n", "Press a or b to recieve a surprise
gift")
                        result = 1
                        goto Snine
                }
                else if (decision == 'e') {
                        printf("\n%s\n", "Looks like you are leaving us. Guess you
just couldn't handle it.")
                        halt
                }
                else if (decision == '(') {
                        goto Sseven
                }
                else {
                        printf("%s\n", "Don't just sit there. Get moving!")
                        result = 1
                        goto Sseven
                }
        state Seight
                if (decision == 'a') {
                        printf("%s\n", "You get little done on the project. But as
you present Edwards is impressed by your vast knowledge ")
                        printf("%s\n", "of anime and gives everyone in your group
a B+.")
                        halt
                }
                else if (decision == 'b') {
                        printf("%s\n", "It's time to present! You realize your
language still sucks but there is little to be done now, you ")
                        printf("%s\n", "did your best. You survived the passive
aggressive facebook messages you and your groupmates sent")
                        printf("%s\n", "to each other, the unproductive meetings,
and emotional/psychological scarring. As you start talking")
                        printf("%s\n", "to Edwards it surprises you that you
actually know what you're talking about. Satisfied, Edwards")
```

```
                        printf("%s\n", "awards everyone in your group with an A.
")
                        halt
                }
                else if (decision == 'e') {
                        printf("\n%s\n", "Looks like you are leaving us. Guess you
just couldn't handle it.")
                        halt
                }
                else if (decision == '(') {
                        goto Seight
                }
                else {
                        printf("%s\n", "Don't just sit there. Get moving!")
                        result = 1
                        goto Seight
                }
        state Snine
                if (decision == 'a') {
                        printf("%s\n", "As the mighty dragon, Fluffles descends
you look on in shock. He bellows in an awesome voice GREETINGS YOUNG TRAVELER.")
                        printf("%s\n", "YOU ARE THE CHOSEN ONE WHO WILL HELP ME
RID THIS WORLD OF EVIL. JOIN ME ON MY QUEST TO FIGHT THE")
                        printf("%s\n\n", "EVIL WIZARD, KERFKLIPKATSTEIN. What do
you do?")
                        printf("%s\n", "a. Decide not to join FLuffles on his
quest.")
                        printf("%s\n\n", "b. Decide to join Fluffles on his
quest.")
                        result = 1
                        goto Stwelve
                }
                else if (decision == 'b') {
                        printf("%s\n", "You are given a seemingly perfectly
working project. What do you do?")
                        printf("%s\n", "a. Yay, you're done! Time to sleep for the
presentation tomorrow.")
                        printf("%s\n%s\n\n", "b. You decide to not trust Satan,
because duh Satan, and run the project one time to make sure", "it works.")
                        result = 1
                        goto Sthirteen
                }
                else if (decision == 'e') {
                        printf("\n%s\n", "Looks like you are leaving us. Guess you
just couldn't handle it.")
                        halt
                }
                else if (decision == '(') {
```

```
                        goto Snine
                }
                else {
                        printf("%s\n", "Don't just sit there. Get moving!")
                        result = 1
                        goto Snine
                }
        state Sten
                if (decision == 'a') {
                        printf("%s\n", "You spend the next five years of your life
writing your memoir of PLT. Unfortunately you failed to")
                        printf("%s\n", "realize that in addition to being a bad
programmer you are terrible writer. Offended by your")
                        printf("%s\n", "writing you get an FF in PLT.")
                        halt
                }
                else if (decision == 'b') {
                        printf("%s\n\n", "But which group mate to sacrifice?")
                        printf("%s\n", "a. Shalva")
                        printf("%s\n", "b. Shalva")
                        printf("%s\n\n", "c. Shalva")
                        result = 1
                        goto Sseven
                }
                else if (decision == 'c') {
                        printf("%s\n", "That failed. Did you learn nothing? You
get caught again, locked up, and miss the final")
                        printf("%s\n", "presentation. It turns out, your group
mates stayed up all night finishing the project.")
                        printf("%s\n%s\n", "In light of their achievements,
Edwards gave them all A+. In light of your absence you", "received a T.")
                        halt
                }
                else if (decision == 'd') {
                        printf("%s\n%s", "You've kidnapped all your groupmates and
locked them in your room. But for some","reason they")
                        printf("%s\n\n", "don’t seem to be getting much done! What
do you do?")
                        printf("%s\n", "a. Offer them Sake to get the creative
juices flowing.")
                        printf("%s\n%s\n", "b. Decide that it’s better to
sacrifice one of them to Satan. After all, what good are they if they", "can’t work?")
                        printf("%s\n\n", "c. Give a motivational speech based off
the one in the movie Independence Day.")
                        result = 1
                        goto Sfive
                }
                else if (decision == 'e') {
```

```
                        printf("\n%s\n", "Looks like you are leaving us. Guess you
just couldn't handle it.")
                        halt
                }
                else if (decision == '(') {
                        goto Sten
                }
                else {
                        printf("%s\n", "Don't just sit there. Get moving!")
                        result = 1
                        goto Sten
                }
        state Seleven
                if (decision == 'a') {
                        printf("%s\n", "It turns out, your group mates stayed up
all night finishing the project. In light of their")
                        printf("%s\n", "achievements, Edwards gave them all A+. In
light of your absence you received a T.")
                        halt
                }
                else if (decision == 'b') {
                        printf("%s\n", "You reach the presentation just in time.
It turns out, that your group mates had done a bit of work")
                        printf("%s\n", "on the project. It was still not fully
functional, but it didn't completely suck. In light of your")
                        printf("%s\n", "hard work, your entire group received a
B-.")
                        halt
                }
                else if (decision == 'e') {
                        printf("%s\n", "Looks like you are leaving us. Guess you
just couldn't handle it.")
                        halt
                }
                else if (decision == '(') {
                        goto Seleven
                }
                else {
                        printf("\n%s\n", "Don't just sit there. Get moving!")
                        result = 1
                        goto Seleven
                }
        state Stwelve
                if (decision == 'a') {
                        printf("%s\n", "I am sorry to say that Fluffles doesn't
take too kindly to the rejection. He eats you. Your group")
                        printf("%s\n", "mates go on to successfully present the
project without you and receive As.")
```

```
                        halt
                }
                else if (decision == 'b') {
                        printf("%s\n", "As you ride off into the sunset, you feel
the stress of PLT falling away. Your group mates are left")
                        printf("%s\n%s\n", "to solve the problem alone. Despairing
at the loss of their teammate they fail the course out of","solidarity of your
absence")
                        halt
                }
                else if (decision == 'e') {
                        printf("%s\n", "Looks like you are leaving us. Guess you
just couldn't handle it.")
                        halt
                }
                else if (decision == '(') {
                        goto Stwelve
                }
                else {
                        printf("\n%s\n", "Don't just sit there. Get moving!")
                        result = 1
                        goto Stwelve
                }
        state Sthirteen
                if (decision == 'a') {
                        printf("%s\n", "It is presentation time. Your project ends
up ripping a hole in the space time continuum which")
                        printf("%s\n%s\n", "releases a slew of demons from the
underworld. Before Edwards gets eaten by a demonic unicorn","he yells You've failed.
Fs all around!")
                        halt
                }
                else if (decision == 'b') {
                        printf("%s\n", "As you delve into your code, your
‘amazing’ tester discovers millions of hidden bugs Satan has")
                        printf("%s\n\n", "hidden in your code. What do you do?")
                        printf("%s\n", "a. Go to Edwards and beg for an
extension.")
                        printf("%s\n%s\n", "b. Kidnap all your groupmates, lock
them in room until the presentation, and only feed them if", "they code.")
                        printf("%s\n%s\n\n", "c. Realize the numbers of cares you
give is smaller than your IQ and decide to present it anyway.","It cannot be too bad,
right?")
                        result = 1
                        goto Sfourteen
                }
                else if (decision == 'e') {
```

```
                        printf("\n%s\n", "Looks like you are leaving us. Guess you
just couldn't handle it.")
                        halt
                }
                else if (decision == '(') {
                        goto Sthirteen
                }
                else {
                        printf("%s\n", "Don't just sit there. Get moving!")
                        result = 1
                        goto Sthirteen
                }
        state Sfourteen
                if (decision == 'a') {
                        printf("%s\n", "Seriously? You're asking me for an
extension? Seriously?, Edwards says with a look of incredulity.")
                        printf("%s\n\n", "I, ummm, I, nope, jk. Bye!!! You run out
the room. What do you do next?")
                        printf("%s\n", "a. Give up, go back to your room, and fall
asleep.")
                        printf("%s\n%s\n\n", "b. Kidnap all your groupmates, lock
them in room until the presentation, and only feed them if", "\tthey code.")
                        result = 1
                        goto Sthree
                }
                else if (decision == 'b') {
                        printf("%s\n", "You've kidnapped all your groupmates and
locked them in your room. But for some reason they")
                        printf("%s\n\n", "don't seem to be getting much done! What
do you do?")
                        printf("%s\n", "a. Offer them Sake to get the creative
juices flowing.")
                        printf("%s\n%s\n", "b. Decide that it's better to
sacrifice one of them to Satan. After all, what good are they if","they can't work?")
                        printf("%s\n\n", "c. Give a motivational speech based off
the one in the movie Independence Day.")
                        result = 1
                        goto Sfive
                }
                else if (decision == 'c') {
                        printf("%s\n", "It is presentation time. Your project ends
up ripping a hole in the space time continuum which ")
                        printf("%s\n", "releases a slew of demons from the
underworld. Before Edwards gets eaten by a demonic unicorn ")
                printf("%s\n", "he yells You've failed. Fs all around!")
                        halt
                }
                else if (decision == 'e') {
```

```
                        printf("\n%s\n", "Looks like you are leaving us. Guess you
just couldn't handle it.")
                        halt
                }
                else if (decision == '(') {
                        goto Sfourteen
                }
                else {
                        printf("%s\n", "Don't just sit there. Get moving!")
                        result = 1
                        goto Sfourteen
                }
    }

adventure.c:
    #include <stdio.h>
    #include "adventure.h"

    int main() {

        struct adventure_input input;
        struct adventure_output output;
        struct adventure_state states;

        adventure_tick(&states, NULL, NULL);
        output.result = 0;

        char user_input[50];
        printf("\033[H\033[J");

        while (output.result == 0) {
            printf("Press s to start the adventure: ");
            scanf("%s", (user_input));
            input.decision = *user_input;
            printf("\n");
            printf("\033[H\033[J");
            adventure_tick(&states, &input, &output);
        }

        while (output.result == 1) {
            input.decision = '(';
            if (adventure_tick(&states, &input, &output) == NULL) {
                break;
            }
            printf("Press the corresponding letter on your keyboard to make your
choice: ");
            scanf("%s", (user_input));
            input.decision = *user_input;
```

```
            printf("\n");
            printf("\033[H\033[J");
            adventure_tick(&states, &input, &output);
        }


        printf("\nThanks for playing :) This story was brought to you by SAKE: Don't
drink if you're underage.");
    }

test_brokenTL.sk:
    input[int inOne, int inTwo]
    output[char outOne, char outTwo]

    fsm tlOne {

       state R1
                    if (inOne == 1) {
                            outOne = 'g'
                            goto G1
                    }
                    else {
                            outOne = 'r'
                            goto R1
                    }
            state Y1
                    outOne = 'r'
                    goto R1
            state G1
                    if (inOne == 0) {
                            outOne = 'y'
                            goto Y1
                    }
                    else {
                            outOne = 'g'
                            goto G1
                    }
    }
    fsm tlTwo {

           state R2
                    if (inTwo == 1) {
                            outTwo = 'g'
                            goto G2
                    }
                    else {
                            outTwo = 'r'
                            goto R2
```

```
                }
        state Y2
                outTwo = 'r'
                goto R2
        state G2
                if (inTwo == 0) {
                        outTwo = 'y'
                        goto Y2
                }
                else {
                        outTwo = 'g'
                        goto G2
                }
    }

test_brokenTL.c:
    #include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>
    #include "test_brokenTL.h"

    int main() {
       struct test_brokenTL_input i;
       struct test_brokenTL_state s;
       struct test_brokenTL_output o;

       test_brokenTL_tick(&s, NULL, NULL);

            char *inputOne = "1111001101";
            char *inputTwo = "1110011011";

            char temp[1];
            int count = 0;

            while (*inputOne) {
                if (count != 0) {
                    sleep(1);
                }

                temp[0] = inputOne[0];
                i.inOne = atoi(temp);
                temp[0] = inputTwo[0];
                i.inTwo = atoi(temp);

                test_brokenTL_tick(&s, &i, &o);

                printf("TL 1: %c\t\t", o.outOne);
                printf("TL 2: %c\n", o.outTwo);
```

```
                inputOne++;
                inputTwo++;

                count = 1;
            }

       return 0;
    }
```

```
test_brokenTL.out:
    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
```

```
test_hogTL.sk:
    input[int inOne, int inTwo]
    output[char outOne, char outTwo]

    fsm tlOne {

            state R1
                    if (inOne == 1 && inTwo == 0 && tlTwo == R2) {
                            outOne = 'g'
                            goto G1
                    }
                    else {
                            outOne = 'r'
                            goto R1
                    }
            state Y1
                    outOne = 'r'
                    goto R1
            state G1
                    if (inOne == 0 || inTwo == 1) {
                            outOne = 'y'
                            goto Y1
                    }
                    else {
                            outOne = 'g'
                            goto G1
```

```
                }
    }
    fsm tlTwo {

            state R2
                    if (inTwo == 1 && inOne == 0 && tlOne == R1) {
                            outTwo = 'g'
                            goto G2
                    }
                    else {
                            outTwo = 'r'
                            goto R2
                    }
            state Y2
                    outTwo = 'r'
                    goto R2
            state G2
                    if (inTwo == 0 || inOne == 1) {
                            outTwo = 'y'
                            goto Y2
                    }
                    else {
                            outTwo = 'g'
                            goto G2
                    }
    }

test_hogTL.c:
    #include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>
    #include "test_hogTL.h"

    int main() {
       struct test_hogTL_input i;
       struct test_hogTL_state s;
       struct test_hogTL_output o;

       test_hogTL_tick(&s, NULL, NULL);

            char *inputOne = "11110000000011111000";
            char *inputTwo = "10000011110000011100";

            char temp[1];
            int count = 0;

            while (*inputOne) {
```

```
                if (count != 0) {
                    sleep(1);
                }

                temp[0] = inputOne[0];
                i.inOne = atoi(temp);
                temp[0] = inputTwo[0];
                i.inTwo = atoi(temp);

                test_hogTL_tick(&s, &i, &o);

                printf("TL 1: %c\t\t", o.outOne);
                printf("TL 2: %c\n", o.outTwo);

                inputOne++;
                inputTwo++;

                count = 1;
            }

       return 0;
    }
```

```
test_hogTL.out:
    TL 1: r         TL 2: r
    TL 1: g         TL 2: r
    TL 1: g         TL 2: r
    TL 1: g         TL 2: r
    TL 1: y         TL 2: r
    TL 1: r         TL 2: r
    TL 1: r         TL 2: g
    TL 1: r         TL 2: g
    TL 1: r         TL 2: g
    TL 1: r         TL 2: g
    TL 1: r         TL 2: y
    TL 1: r         TL 2: r
    TL 1: g         TL 2: r
    TL 1: g         TL 2: r
    TL 1: g         TL 2: r
    TL 1: y         TL 2: r
    TL 1: r         TL 2: r
    TL 1: r         TL 2: g
    TL 1: r         TL 2: y
    TL 1: r         TL 2: r
```

```
test_loopingTL.sk:
    input[int inTwo]
    output[char outOne, char outTwo]
```

```
fsm tlOne {
        public int turnOne = 1
        public int count = 0

        state R1
                if (tlTwo.turnTwo == 1) {
                        outOne = 'r'
                        count = 0
                        goto R1
                }
                else {
                        if (count < 1) {
                                count = count + 1
                                outOne = 'r'
                                goto R1
                        }
                        count = 0
                        outOne = 'g'
                        goto G1
                }
        state Y1
                outOne = 'r'
                count = 0
                goto R1
        state G1
                if (tlTwo.turnTwo == 0) {
                        outOne = 'g'
                        goto G1
                }
                outOne = 'y'
                goto Y1
}
fsm tlTwo {
        public int turnTwo = 0
        public int count = 0

        state R2
                if (inTwo == 1) {
                        turnTwo = 1
                }
                if (turnTwo == 1 && tlOne == R1) {
                        outTwo = 'g'
                        count = 0
                        goto G2
                }
                else {
                        outTwo = 'r'
```

```
                        goto R2
                }
        state Y2
                outTwo = 'r'
                goto R2
        state G2
                while (count < 10) {
                        count = count + 1
                        outTwo = 'g'
                        goto G2
                }
                outTwo = 'y'
                turnTwo = 0
                goto Y2
    }

test_loopingTL.c:
    #include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>
    #include "test_loopingTL.h"

    int main() {
       struct test_loopingTL_input i;
       struct test_loopingTL_state s;
       struct test_loopingTL_output o;

       test_loopingTL_tick(&s, NULL, NULL);

            char *inputTwo = "00001001110000000000000000111100";

            char temp[1];
            int count = 0;

            while (*inputTwo) {
                if (count != 0) {
                    sleep(1);
                }

                temp[0] = inputTwo[0];
                i.inTwo = atoi(temp);

                test_loopingTL_tick(&s, &i, &o);

                printf("TL 1: %c\t\t", o.outOne);
                printf("TL 2: %c\n", o.outTwo);

                inputTwo++;
```

```
                count = 1;
            }

       return 0;
    }
```

```
test_loopingTL.out:
    TL 1: r         TL 2: r
    TL 1: g         TL 2: r
    TL 1: g         TL 2: r
    TL 1: g         TL 2: r
    TL 1: g         TL 2: r
    TL 1: y         TL 2: r
    TL 1: r         TL 2: r
    TL 1: r         TL 2: g
    TL 1: r         TL 2: g
    TL 1: r         TL 2: g
    TL 1: r         TL 2: g
    TL 1: r         TL 2: g
    TL 1: r         TL 2: g
    TL 1: r         TL 2: g
    TL 1: r         TL 2: g
    TL 1: r         TL 2: g
    TL 1: r         TL 2: g
    TL 1: r         TL 2: g
    TL 1: r         TL 2: y
    TL 1: r         TL 2: r
    TL 1: g         TL 2: r
    TL 1: g         TL 2: r
    TL 1: g         TL 2: r
    TL 1: g         TL 2: r
    TL 1: g         TL 2: r
    TL 1: g         TL 2: r
    TL 1: g         TL 2: r
    TL 1: y         TL 2: r
    TL 1: r         TL 2: r
    TL 1: r         TL 2: g
    TL 1: r         TL 2: g
    TL 1: r         TL 2: g
```

```
test_trafficLightEndTL.sk:
    input[int i]
    output[char out]

    fsm trafficLight {

        state Red
                    switch(i) {
```

```
                        case 1: out = 'g'
                                goto Green
                        case 0: out = 'r'
                                goto Red
                }
        state Green
                switch(i) {
                        case 1: out = 'g'
                                goto Green
                        case 0: out = 'y'
                                goto Yellow
                }
        state Yellow
                out = 'r'
    }

test_trafficLightEndTL.c:
    #include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>
    #include "test_trafficLightEndTL.h"

    int main() {
       struct test_trafficLightEndTL_input in;
       struct test_trafficLightEndTL_state s;
       struct test_trafficLightEndTL_output o;

       test_trafficLightEndTL_tick(&s, NULL, NULL);

            char *input = "00011111000";

            char temp[1];
            int count = 0;

            while (*input) {

                if (count != 0) {
                    sleep(1);
                }

                temp[0] = input[0];
                in.i = atoi(temp);

                if (test_trafficLightEndTL_tick(&s, &in, &o) == NULL) {
                     printf("Ended traffic light sequence");
                     break;
                }
```

```
                printf("Light color: %c\n", o.out);

                input++;
                count = 1;
            }

       return 0;
    }
```

```
test_trafficLightEndTL.out:
    Light color: r
    Light color: r
    Light color: r
    Light color: g
    Light color: g
    Light color: g
    Light color: g
    Light color: g
    Light color: y
    Light color: r
    Light color: r
```

```
test_trafficLightTL.sk:
    input[int i]
    output[char out]

    fsm trafficLight {

        state Red
                    switch(i) {
                            case 1: out = 'g'
                                    goto Green
                            case 0: out = 'r'
                                    goto Red
                    }
            state Yellow
                    out = 'r'
                    goto Red
            state Green
                    switch(i) {
                            case 1: out = 'g'
                                    goto Green
                            case 0: out = 'y'
                                    goto Yellow
                    }
    }
```

```
test_trafficLightTL.c:
```

```
    #include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>
    #include "test_trafficLightTL.h"

    int main() {
       struct test_trafficLightTL_input in;
       struct test_trafficLightTL_state s;
       struct test_trafficLightTL_output o;

       test_trafficLightTL_tick(&s, NULL, NULL);

            char *input = "00011111000";

            char temp[1];
            int count = 0;

            while (*input) {

                if (count != 0) {
                    sleep(1);
                }

                temp[0] = input[0];
                in.i = atoi(temp);

                test_trafficLightTL_tick(&s, &in, &o);

                printf("Light color: %c\n", o.out);

                input++;
                count = 1;
            }

       return 0;
    }
```

```
test_trafficLightTL.out:
    Light color: r
    Light color: r
    Light color: r
    Light color: g
    Light color: g
    Light color: g
    Light color: g
    Light color: g
    Light color: y
    Light color: r
```

```
    Light color: r

test_unreachableTL.sk:
    input[int inOne]
    output[char outOne]

    fsm tlOne {

            state R1
                    if (inOne == 1) {
                            outOne = 'g'
                            goto G1
                    }
                    else {
                            outOne = 'r'
                            goto R1
                    }
            state Y1
                    outOne = 'y'
                    goto Y1
            state G1
                    if (inOne == 0) {
                            outOne = 'y'
                            goto Y1
                    }
                    else {
                            outOne = 'g'
                            goto G1
                    }
    }

test_unreachableTL.c:
    #include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>
    #include "test_unreachableTL.h"

    int main() {
       struct test_unreachableTL_input in;
       struct test_unreachableTL_state s;
       struct test_unreachableTL_output o;

       test_unreachableTL_tick(&s, NULL, NULL);

            char *input = "11101010101";
            char temp[1];
            int count = 0;
```

```
            while(*input) {

                if (count != 0) {
                    sleep(1);
                }

                temp[0] = input[0];
                in.inOne = atoi(temp);

                test_unreachableTL_tick(&s, &in, &o);
                printf("Light color: %c\n", o.outOne);

                input++;
                count = 1;
            }

       return 0;
    }

test_unreachableTL.out:
    Light color: g
    Light color: g
    Light color: g
    Light color: y
    Light color: y
    Light color: y
    Light color: y
    Light color: y
    Light color: y
    Light color: y
    Light color: y

test_42.sk:
    fsm fortytwo {

            printf("%d\n", 42)
    }

test_42.c:
    #include <stdio.h>
    #include "test_42.h"

    int main() {
       struct test_42_input i;
       struct test_42_state s;

       test_42_tick(&s, NULL, NULL);
       test_42_tick(&s, &i, NULL);
```

```
       return 0;
    }

test_42.out:
    42

test_block.sk:
    fsm bc {
            /~ block comment ~/
            /~ block
             ~ comments
             ~ on
             ~ multiple
             ~ lines
             ~/

       printf("%s", "I successfully navigated a block comment!")
            /~ block
               comment
             ~/
    }

test_block.c:
    #include <stdio.h>
    #include "test_block.h"

    int main() {
       struct test_block_input i;
       struct test_block_state s;

       test_block_tick(&s, NULL, NULL);
       test_block_tick(&s, &i, NULL);

       return 0;
    }

test_block.out:
    I successfully navigated a block comment!

test_bool.sk:
    input [bool i]
    output [bool o]

    fsm test {
            ~ publics
            public bool l = false
```

```
            if (l) {
                    printf("%s\n", "L is true")
            }
            else {
                    printf("%s\n", "L is false")
            }
            if (i) {
                    printf("%s\n", "i is true")
            }
            else {
                    printf("%s\n", "i is false")
            }
    }

test_bool.c:
    #include <stdio.h>
    #include "test_bool.h"

    int main() {
       struct test_bool_input i;
       struct test_bool_state s;

       test_bool_tick(&s, NULL, NULL);

            i.i = 1;

            test_bool_tick(&s, &i, NULL);

       return 0;
    }

test_bool.out:
    L is false
    i is true

test_char.sk:
    fsm ekas {
            char decision = 's'

            if (decision == 's') {
                    printf("%c\n", decision)
            }
    }

test_char.c:
    #include <stdio.h>
    #include "test_char.h"
```

```
    int main() {
       struct test_char_input i;
       struct test_char_state s;

       test_char_tick(&s, NULL, NULL);
       test_char_tick(&s, &i, NULL);

       return 0;
    }

test_char.out:
    s

test_comment.sk:
    fsm comments {
            ~ int i = 4
            ~ public int k = 4

       printf("%s", "Line comment sandwich")
            ~ hello there
    }

test_comment.c:
    #include <stdio.h>
    #include "test_comment.h"

    int main() {
       struct test_comment_input i;
       struct test_comment_state s;

       test_comment_tick(&s, NULL, NULL);
       test_comment_tick(&s, &i, NULL);

       return 0;
    }

test_comment.out:
    Line comment sandwich

test_concurrent.sk:
    input[int i]
    output[int o]

    fsm one {
            public int j = 4
            int r = two.k

            state DependK
```

```
                    r = two.k
                    printf("%d\n", r)
                    goto DependK
    }
    fsm two {
            public int k = 3

       state SetK
                    if (i == 2) {
                            k = 45
                    }
                    printf("%d\n", k)
                    goto SetK
    }

test_concurrent.c:
    #include <stdio.h>
    #include "test_concurrent.h"

    int main() {
       struct test_concurrent_input i;
       struct test_concurrent_state s;

       test_concurrent_tick(&s, NULL, NULL);

            i.i = 42;
            test_concurrent_tick(&s, &i, NULL);

            i.i = 2;
            test_concurrent_tick(&s, &i, NULL);

            i.i = 42;
            test_concurrent_tick(&s, &i, NULL);

       return 0;
    }

test_concurrent.out:
    3
    3
    3
    45
    45
    45

test_conjunctions.sk:
    input [int v]
    output [int p]
```

```
    fsm conjunctions {
            int q = 4

            if (v == 5) {
                    p = 6
            }
            else if (v == 7 || q == 4) {
                    p = 89
                    q = 10
            }
            else if (v == 123) {
                    p = 21
            }
            else {
                    p = 42
            }
    }

test_conjunctions.c:
    #include <stdio.h>
    #include "test_conjunctions.h"

    int main() {

        struct test_conjunctions_input i;
        struct test_conjunctions_state s;
        struct test_conjunctions_output o;

        test_conjunctions_tick(&s, NULL, NULL);

        i.v = 5;
        test_conjunctions_tick(&s, &i, &o);
        printf("%d\n", o.p);

        i.v = 1233;
        test_conjunctions_tick(&s, &i, &o);
        printf("%d\n", o.p);

        i.v = 123;
        test_conjunctions_tick(&s, &i, &o);
        printf("%d\n", o.p);

        i.v = 7;
        test_conjunctions_tick(&s, &i, &o);
        printf("%d\n", o.p);

        i.v = 56;
```

```
        test_conjunctions_tick(&s, &i, &o);
        printf("%d\n", o.p);

        return 0;
    }

test_conjunctions.out:
    6
    89
    89
    89
    89

test_emptyfsm.sk:
    input [int i]
    output [bool k]

    fsm empty {

    }

test_emptyfsm.c:
    #include <stdio.h>
    #include "test_emptyfsm.h"

    int main() {
       struct test_emptyfsm_input i;
       struct test_emptyfsm_state s;

       test_emptyfsm_tick(&s, NULL, NULL);
       test_emptyfsm_tick(&s, &i, NULL);

       return 0;
    }

test_emptyfsm.out:

test_enums.sk:
    input[int p, string q]
    output[int k]

    type Emma = Shalva | Ac | Kc

    fsm hello {
            Emma x = Shalva

       state Hello
                    if (p == 1) {
```

```
                        x = Ac
                }
                if (x == Ac) {
                        printf("%s", "Hello ")
                        goto World
                }
                else {
                        goto Hello
                }
        state World
                if (p == 1) {
                        printf("%s%s\n", q, "'s World")
                        goto Hello
                }
                else {
                        goto World
                }
    }

test_enums.c:
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include "test_enums.h"

    int main(){

        struct test_enums_state s;
        struct test_enums_input i;

        test_enums_tick(&s, NULL, NULL);

        i.p = 0; // print nothing
        test_enums_tick(&s, &i, NULL);

        i.p = 1; // print hello
        test_enums_tick(&s, &i, NULL);

        i.p = 0;
        i.q = "Emma"; //print nothing
        test_enums_tick(&s, &i, NULL);

        i.p = 1;
        i.q = "Emma"; //print Emma's world
        test_enums_tick(&s, &i, NULL);

        // print hello b/c p is still 1
        test_enums_tick(&s, &i, NULL);
```

```
        return 0;
    }

test_enums.out:
    Hello Emma's World
    Hello

test_for.sk:
    fsm test {
            int i = 2

       for i in (0:4:1) {
                i = i + 1
            }
            printf("%d", i)
    }

test_for.c:
    #include <stdio.h>
    #include "test_for.h"

    int main() {
       struct test_for_input i;
       struct test_for_state s;

       test_for_tick(&s, NULL, NULL);
       test_for_tick(&s, &i, NULL);

       return 0;
    }

test_for.out:
    4

test_fsmhello.sk:
    input[int p, string q]
    output[int k]

    fsm hello {

       state Hello
                if (p == 1) {
                        printf("%s", "Hello ")
                        goto World
                }
                else {
                        goto Hello
```

```
                }
        state World
                if (p == 1) {
                        printf("%s%s\n", q, "'s World")
                        goto Hello
                }
                else {
                        goto World
                }
    }

test_fsmhello.c:
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include "test_fsmhello.h"

    int main(){

        struct test_fsmhello_state s;
        struct test_fsmhello_input i;

        test_fsmhello_tick(&s, NULL, NULL);

        i.p = 0; // print nothing
        test_fsmhello_tick(&s, &i, NULL);

        i.p = 1; // print hello
        test_fsmhello_tick(&s, &i, NULL);

        i.p = 0;
        i.q = "Emma"; //print nothing
        test_fsmhello_tick(&s, &i, NULL);

        i.p = 1;
        i.q = "Emma"; //print Emma's world
        test_fsmhello_tick(&s, &i, NULL);

        // print hello b/c p is still 1
        test_fsmhello_tick(&s, &i, NULL);

        return 0;
    }

test_fsmhello.out:
    Hello Emma's World
    Hello
```

```
test_halt.sk:
    input[int p, string q]
    output[int k]

    fsm halting {

       state One
                    if (p == 1) {
                            printf("%s", "Hello ")
                            goto Two
                    }
                    else {
                            goto One
                    }
             state Two
                    if (p == 1) {
                            printf("%s%s\n", q, "'s World")
                            halt
                    }
                    else {
                            goto Two
                    }
    }

test_halt.c:
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include "test_halt.h"

    int main(){

        struct test_halt_state s;
        struct test_halt_input i;

        test_halt_tick(&s, NULL, NULL);

        i.p = 0; // print nothing
        test_halt_tick(&s, &i, NULL);

        i.p = 1; // print hello
        test_halt_tick(&s, &i, NULL);

        i.p = 0;
        i.q = "Emma"; //print nothing
        test_halt_tick(&s, &i, NULL);

        i.p = 1;
```

```
        i.q = "Emma"; //print Emma's world
        test_halt_tick(&s, &i, NULL);

        if (test_halt_tick(&s, &i, NULL) == NULL) {
           printf("And we halted");
        }

        return 0;
    }

test_halt.out:
    Hello Emma's World
    And we halted

test_header.sk:
    input[int i, int o]
    output[int k, int l]

    type Test = One | Two | Three

    fsm one {
            public int j = 4
            ~ int r = two.k

            state Emma
                    goto Shalva
            state Shalva
                    goto Emma
    }
    fsm two {
            public int f = 3

       state Kz
                    f = 45
    }
    fsm three {
            public int k = 4
            int f = 24
            int z = 2

            state Ac
                   z = 35
    }
    fsm four {
          int t = 132124
          int s = 123123

    }
```

```
    fsm five {

    }
    fsm six {

            state Yay
                    goto Yay
    }

test_header.c:
    #include <stdio.h>
    #include "test_header.h"

    int main() {
       struct test_header_input i;
       struct test_header_state s;

       test_header_tick(&s, NULL, NULL);
       test_header_tick(&s, &i, NULL);

       return 0;
    }

test_header.out:

test_hello.sk:
    fsm hello {

       printf("%d\n", 1*1)
            printf("%d\n", 2 * 3 - 4)
            printf("%d\n", 3)
            printf("%d\n", 5*4-16)
            printf("%d\n", 4+1)
            printf("%d\n", 6-0)
            printf("%d\n", 1+1+1+1+1+1+1)
            printf("%d\n", 1+2-3+4+4)
    }

test_hello.c:
    #include <stdio.h>
    #include "test_hello.h"

    int main() {
       struct test_hello_input i;
       struct test_hello_state s;

       test_hello_tick(&s, NULL, NULL);
       test_hello_tick(&s, &i, NULL);
```

```
       return 0;
    }

test_hello.out:
    1
    2
    3
    4
    5
    6
    7
    8

test_ifelse.sk:
    input [int v]
    output [int p]

    fsm headerTest {

            if (v == 5) {
                    p = 6
            }
            else if (v == 7) {
                    p = 89
            }
            else if (v == 123) {
                    p = 21
            }
            else {
                    p = 42
            }
    }

test_ifelse.c:
    #include <stdio.h>
    #include "test_ifelse.h"

    int main() {

        struct test_ifelse_input i;
        struct test_ifelse_state s;
        struct test_ifelse_output o;

        test_ifelse_tick(&s, NULL, NULL);

        i.v = 5;
        test_ifelse_tick(&s, &i, &o);
```

```
        printf("%d\n", o.p);

        i.v = 1233;
        test_ifelse_tick(&s, &i, &o);
        printf("%d\n", o.p);

        i.v = 123;
        test_ifelse_tick(&s, &i, &o);
        printf("%d\n", o.p);

        i.v = 7;
        test_ifelse_tick(&s, &i, &o);
        printf("%d\n", o.p);

        i.v = 56;
        test_ifelse_tick(&s, &i, &o);
        printf("%d\n", o.p);

        return 0;
    }

test_ifelse.out:
    6
    42
    21
    89
    42

test_multiSwitch.sk:
    input[int i]
    output[int out]

    fsm switchCase {
            int p = 0

            switch(i) {
                    case 0: out = 42
                            p = 24
                            printf("%d\n", p)
             case 1: out = 24
                            p = 42
                            printf("%d\n", p)
            }
    }

test_multiSwitch.c:
    #include <stdio.h>
    #include "test_multiSwitch.h"
```

```
    int main() {
       struct test_multiSwitch_input i;
       struct test_multiSwitch_state s;
            struct test_multiSwitch_output o;

       test_multiSwitch_tick(&s, NULL, NULL);

            i.i = 0;
            test_multiSwitch_tick(&s, &i, &o);
            printf("%d\n", o.out);

            i.i = 1;
            test_multiSwitch_tick(&s, &i, &o);
            printf("%d\n", o.out);

            i.i = 0;
            test_multiSwitch_tick(&s, &i, &o);
            printf("%d\n", o.out);

       return 0;
    }

test_multiSwitch.out:
    24
    42
    42
    24
    24
    42

test_multiSwitch2.sk:
    input[int i]
    output[int out]

    fsm switchCase {
            int p = 4

            switch(i) {
                    case 0: out = 42
                            printf("%d\n", 42)
             case 1: out = 24
                            printf("%d\n", 24)
            }
    }

test_multiSwitch2.c:
    #include <stdio.h>
```

```
    #include "test_multiSwitch2.h"

    int main() {
       struct test_multiSwitch2_input i;
       struct test_multiSwitch2_state s;
            struct test_multiSwitch2_output o;

       test_multiSwitch2_tick(&s, NULL, NULL);

            i.i = 0;
            test_multiSwitch2_tick(&s, &i, &o);
            printf("%d\n", o.out);

            i.i = 1;
            test_multiSwitch2_tick(&s, &i, &o);
            printf("%d\n", o.out);

       return 0;
    }

test_multiSwitch2.out:
    42
    42
    24
    24

test_nestedFor.sk:
    input [int p]
    output [int q]

    fsm headerTest {
            int k = 1
       int i
       int j

            for i in (0:2:1) {
                    for j in (0:5:1) {
                            k = k + 1
                    }
            }
            q = k
    }

test_nestedFor.c:
    #include <stdio.h>
    #include "test_nestedFor.h"

    int main() {
```

```
       struct test_nestedFor_input i;
       struct test_nestedFor_state s;
            struct test_nestedFor_output o;

       test_nestedFor_tick(&s, NULL, NULL);

       test_nestedFor_tick(&s, &i, &o);

            printf("%d", o.q);

       return 0;
    }

test_nestedFor.out:
    11

test_nestedIf.sk:
    fsm ifTest {

            if (true) {
                    if (true) {
                            printf("%s\n", "True Loop")
                    }
            }
            if (true) {
                    if (false) {
                            printf("%s\n", "False Loop")
                    }
            }
    }

test_nestedIf.c:
    #include <stdio.h>
    #include "test_nestedIf.h"

    int main() {
       struct test_nestedIf_input i;
       struct test_nestedIf_state s;

       test_nestedIf_tick(&s, NULL, NULL);
       test_nestedIf_tick(&s, &i, NULL);

       return 0;
    }

test_nestedIf.out:
    True Loop
```

```
test_nestedIf2.sk:
    fsm ifTest {
            int n = 4
            int m = 5

            if (n == 4) {
                    n = n + 5
                    if (m == 5) {
                            m = m + 4
                    }
            }
            printf("%s %d\n%s %d", "n:", n, "m:", m)
    }

test_nestedIf2.c:
    #include <stdio.h>
    #include "test_nestedIf2.h"

    int main() {
       struct test_nestedIf2_input i;
       struct test_nestedIf2_state s;

       test_nestedIf2_tick(&s, NULL, NULL);
       test_nestedIf2_tick(&s, &i, NULL);

       return 0;
    }

test_nestedIf2.out:
    n: 9
    m: 9

test_nestedWhile.sk:
    fsm nestWhile {
       int i = 0
       int k = 0
       int out = 0

       while(i != 2) {
              while(k != 2){
                  out = out + 1
                  k = k + 1
              }
              k = 0
              i = i + 1
       }
       printf("%d", out)
    }
```

```
test_nestedWhile.c:
    #include <stdio.h>
    #include "test_nestedWhile.h"
    int main() {
    struct test_nestedWhile_input i;
    struct test_nestedWhile_state s;
    test_nestedWhile_tick(&s, NULL, NULL);
    test_nestedWhile_tick(&s, &i, NULL);
    return 0;}

test_nestedWhile.out:
    4

test_onePub.sk:
    fsm onePub {
        public bool fixed = false

        ~fix = 42
    }

test_onePub.c:
    #include <stdio.h>
    #include "test_onePub.h"

    int main() {
       struct test_onePub_input i;
       struct test_onePub_state s;

       test_onePub_tick(&s, NULL, NULL);
       test_onePub_tick(&s, &i, NULL);

       return 0;
    }

test_onePub.out:

test_printing.sk:
    fsm printing {
              public int i = 378
             int n = 34
             string w = "hello"
             char c = 'e'

              i = 0
             printf("%d\n", 1*1)
             printf("%d %s\n", 42, "is a cool number!")
             printf("%d %s\n", n, "is also a cool number!")
```

```
            printf("%d %s\n", n+1, "is also a cool number!")
            n = 54
            printf("%d %s\n", n, "is also a cool number!")
            n = n + 1
            printf("%d %s\n", n, "is also a cool number!")
            printf("%s %s\n", w, " world!")
            printf("%c %s\n", c, " is the coolest letter!")
            printf("%s %s %s %s\n", "This", "is", "multiple", "strings")
            printf("\t%s\n", "This is tabbed in!")
            printf("%s\r\n", "Carriage return")
            printf("\\%s\n", " is a backslash")
            printf("\'%s\n", " is a single quote")
    }

test_printing.c:
    #include <stdio.h>
    #include "test_printing.h"

    int main() {
       struct test_printing_input i;
       struct test_printing_state s;

       test_printing_tick(&s, NULL, NULL);
       test_printing_tick(&s, &i, NULL);

       return 0;
    }

test_printing.out:
    1
    42 is a cool number!
    34 is also a cool number!
    35 is also a cool number!
    54 is also a cool number!
    55 is also a cool number!
    hello  world!
    e  is the coolest letter!
    This is multiple strings
            This is tabbed in!
    Carriage return
    \ is a backslash
    ' is a single quote

test_string.sk:
    fsm stringlit {

            printf("%s\n", "Hello World")
            printf("\t%s\n", "Hello World")
```

```
            printf("\t\t%s", "Hello World")
    }

test_string.c:
    #include <stdio.h>
    #include "test_string.h"

    int main() {
       struct test_string_input i;
       struct test_string_state s;

       test_string_tick(&s, NULL, NULL);
       test_string_tick(&s, &i, NULL);

       return 0;
    }

test_string.out:
    Hello World
            Hello World
                    Hello World

test_switch.sk:
    input[int i]
    output[int o]

    fsm switchCase {

            switch(i) {
                    case 0: o = 42
             case 1: o = 24
            }
    }

test_switch.c:
    #include <stdio.h>
    #include "test_switch.h"

    int main() {
       struct test_switch_input i;
       struct test_switch_state s;
            struct test_switch_output o;

       test_switch_tick(&s, NULL, NULL);

            i.i = 0;
            test_switch_tick(&s, &i, &o);
            printf("%d\n", o.o);
```

```
            i.i = 1;
            test_switch_tick(&s, &i, &o);
            printf("%d\n", o.o);

       return 0;
    }

test_switch.out:
    42
    24

test_switch1.sk:
    fsm test {
            public int y = 4

             switch(y) {
                    case 4: printf("%d", y)
            }
    }

test_switch1.c:
    #include <stdio.h>
    #include "test_switch1.h"

    int main() {
       struct test_switch1_input i;
       struct test_switch1_state s;

       test_switch1_tick(&s, NULL, NULL);
       test_switch1_tick(&s, &i, NULL);

       return 0;
    }

test_switch1.out:
    4

test_variables.sk:
    type Emma = Shalva | Ac | Kc

    fsm variables {
            public int j = 4
            public int x = 7
            public int q = 190
            bool p = false
            int l = 0
            bool r = true
```

```
            string w = "Hello"
            char n = 'c'

    }

test_variables.c:
    #include <stdio.h>
    #include "test_variables.h"

    int main() {
       struct test_variables_input i;
       struct test_variables_state s;

       test_variables_tick(&s, NULL, NULL);
       test_variables_tick(&s, &i, NULL);

       return 0;
    }

test_variables.out:

test_while.sk:
    input [int i]
    output [int o]

    fsm headerTest {
            int v = 5

            while (v < 15) {
                    v = v + 1
            }
            o = v
    }

test_while.c:
    #include <stdio.h>
    #include "test_while.h"

    int main() {
       struct test_while_input i;
       struct test_while_state s;
       struct test_while_output o;

       test_while_tick(&s, NULL, NULL);
       test_while_tick(&s, &i, &o);
            printf("%d", o.o);
       return 0;
    }
```

```
test_while.out:
    15

fail_assign.sk:
    fsm printing {
            int p = "hello"

    }

fail_assign.err:
    Fatal error: exception Semant.SemanticError("illegal assignment")

fail_comments.sk:
    fsm printing {
            /~ This hopefully will break :)
    }

fail_comments.err:
    Fatal error: exception Failure("lexing: empty token")

fail_dupEnums.sk:
    input[int p, string q]
    output[int k]

    type Emma = Shalva | Ac | Ac

    fsm hello {

       state Hello
                    if (p == 1) {
                            printf("%s", "Hello ")
                            goto World
                    }
                    else {
                            goto Hello
                    }
            state World
                    if (p == 1) {
                            printf("%s%s\n", q, "'s World")
                            goto Hello
                    }
                    else {
                            goto World
                    }
    }

fail_dupEnums.err:
```

```
    Fatal error: exception Failure("duplicate type Ac")

fail_dupFsm.sk:
    input[int p, string q]
    output[int k]

    fsm hello {

       state Hello
                    if (p == 1) {
                            printf("%s", "Hello ")
                            goto World
                    }
                    else {
                            goto Hello
                    }
             state World
                    if (p == 1) {
                            printf("%s%s\n", q, "'s World")
                            goto Hello
                    }
                    else {
                            goto World
                    }
    }
    fsm hello {

       state Hello
                    if (p == 1) {
                            printf("%s", "Hello ")
                            goto World
                    }
                    else {
                            goto Hello
                    }
             state World
                    if (p == 1) {
                            printf("%s%s\n", q, "'s World")
                            goto Hello
                    }
                    else {
                            goto World
                    }
    }

fail_dupFsm.err:
    Fatal error: exception Failure("duplicate fsm hello")
```

```
fail_dupGlobal.sk:
    input[int p, string q]
    output[int k, char k]

    fsm hello {

       state Hello
                    if (p == 1) {
                            printf("%s", "Hello ")
                            goto World
                    }
                    else {
                            goto Hello
                    }
             state World
                    if (p == 1) {
                            printf("%s%s\n", q, "'s World")
                            goto Hello
                    }
                    else {
                            goto World
                    }
    }

fail_dupGlobal.err:
    Fatal error: exception Failure("duplicate global k")

fail_dupLocal.sk:
    input[int p, string q]
    output[int k]

    fsm hello {
            int x = 42
            int x = 43

       state Hello
                    if (p == 1) {
                            printf("%s", "Hello ")
                            goto World
                    }
                    else {
                            goto Hello
                    }
             state World
                    if (p == 1) {
                            printf("%s%s\n", q, "'s World")
                            goto Hello
                    }
```

```
                    else {
                            goto World
                    }
    }

fail_dupLocal.err:
    Fatal error: exception Failure("duplicate local x")

fail_dupPublic.sk:
    input[int p, string q]
    output[int k]

    fsm hello {
            public int g = 42
            public int g = 24

       state Hello
                    if (p == 1) {
                            printf("%s", "Hello ")
                            goto World
                    }
                    else {
                            goto Hello
                    }
            state World
                    if (p == 1) {
                            printf("%s%s\n", q, "'s World")
                            goto Hello
                    }
                    else {
                            goto World
                    }
    }

fail_dupPublic.err:
    Fatal error: exception Failure("duplicate public hello_g")

fail_dupStates.sk:
    input[int p, string q]
    output[int k]

    fsm hello {

       state Hello
                    if (p == 1) {
                            printf("%s", "Hello ")
                            goto Hello
                    }
```

```
                else {
                        goto Hello
                }
        state Hello
                if (p == 1) {
                        printf("%s%s\n", q, "'s World")
                        goto Hello
                }
                else {
                        goto Hello
                }
    }

fail_dupStates.err:
    Fatal error: exception Failure("duplicate state Hello in hello")

fail_dupVars.sk:
    input[int p, string p]
    output[int k]

    fsm hello {

       state Hello
                if (p == 1) {
                        printf("%s", "Hello ")
                        goto World
                }
                else {
                        goto Hello
                }
        state World
                if (p == 1) {
                        printf("%s%s\n", q, "'s World")
                        goto Hello
                }
                else {
                        goto World
                }
    }

fail_dupVars.err:
    Fatal error: exception Failure("duplicate global p")

fail_empty.sk:



fail_empty.err:
```

```
    Fatal error: exception Parsing.Parse_error

fail_fsmhello.sk:
    input[int p, string q]
    output[int k]

    fsm hello {

       state Hello
                    if (i == 1) {
                            printf("%s", "Hello")
                            goto World
                    }
                    else {
                            goto Hello
                    }
             state World
                    if (i == 1) {
                            printf("%s%s\n", q, "'s World")
                            goto Hello
                    }
                    else {
                            goto World
                    }
    }

fail_fsmhello.err:
    Fatal error: exception Semant.SemanticError("undeclared identifier i")

fail_hello.sk:
    fsm hello {
             printf (1*1)
             printf (2 * 3 - 4)
             printf (3)
             printf (5*4-16)
             printf (4+1)
             printf (6-0)
             printf (1+1+1+1+1+1+1)
             printf (1+2-3+4+4)
    }

fail_hello.err:
    Fatal error: exception Parsing.Parse_error

fail_illegalAssign.sk:
    input[int p, string q]
    output[int k]
```

```
    fsm hello {
            int x = 4

       state Hello
                    if (p == 1) {
                            printf("%s", "Hello ")
                            x = 'e'
                            goto World
                    }
                    else {
                            goto Hello
                    }
            state World
                    if (p == 1) {
                            printf("%s%s\n", q, "'s World")
                            goto Hello
                    }
                    else {
                            goto World
                    }
    }

fail_illegalAssign.err:
    Fatal error: exception Semant.SemanticError("illegal assignment")

fail_illegalBin.sk:
    input[int p, string q]
    output[int k]

    fsm hello {
            int x = 6
            char y = 'a'

       state Hello
                    if (p == 1) {
                            printf("%s", "Hello ")
                            x = x / y
                            goto World
                    }
                    else {
                            goto Hello
                    }
            state World
                    if (p == 1) {
                            printf("%s%s\n", q, "'s World")
                            goto Hello
                    }
                    else {
```

```
                        goto World
                }
    }

fail_illegalBin.err:
    Fatal error: exception Semant.SemanticError("illegal binary operator")

fail_illegalOp.sk:
    input[int p, string q]
    output[int k]

    fsm hello {
            int x = 4

       state Hello
                if (!x) {
                        printf("%s", "Hello ")
                        goto World
                }
                else {
                        goto Hello
                }
            state World
                if (p == 1) {
                        printf("%s%s\n", q, "'s World")
                        goto Hello
                }
                else {
                        goto World
                }
    }

fail_illegalOp.err:
    Fatal error: exception Semant.SemanticError("illegal unary operator")

fail_illegalPredicate.sk:
    input[int p, string q]
    output[int k]

    fsm hello {
            char e = 'e'

       state Hello
                if (e) {
                        printf("%s", "Hello ")
                        goto World
                }
                else {
```

```
                        goto Hello
                }
        state World
                if (p == 1) {
                        printf("%s%s\n", q, "'s World")
                        goto Hello
                }
                else {
                        goto World
                }
    }

fail_illegalPredicate.err:
    Fatal error: exception Semant.SemanticError("Illegal predicate type")

fail_noState.sk:
    input[int p, string q]
    output[int k]

    fsm hello {

       state Hello
                if (p == 1) {
                        printf("%s", "Hello ")
                        goto World
                }
                else {
                        goto Jupiter
                }
        state World
                if (p == 1) {
                        printf("%s%s\n", q, "'s World")
                        goto Hello
                }
                else {
                        goto Mars
                }
    }

fail_noState.err:
    Fatal error: exception Semant.SemanticError("No such state exists")

fail_printing.sk:
    fsm printing {
            char c = 'e'

            printf("%c %s\n", c+1, " is the coolest letter!")
    }
```

```
fail_printing.err:
    Fatal error: exception Semant.SemanticError("illegal binary operator")

fail_typeMismatch.sk:
    input[char decision]
    output[char result]

    fsm ekas {

            state Sone
                    result = 1
    }

fail_typeMismatch.err:
    Fatal error: exception Semant.SemanticError("illegal assignment")

fail_undeclared.sk:
    input[int p, string q]
    output[int k]

    fsm hello {

       state Hello
                    if (p == 1) {
                            printf("%s", "Hello ")
                            undec = 5
                            goto World
                    }
                    else {
                            goto Hello
                    }
            state World
                    if (p == 1) {
                            printf("%s%s\n", q, "'s World")
                            goto Hello
                    }
                    else {
                            goto World
                    }
    }

fail_undeclared.err:
    Fatal error: exception Semant.SemanticError("undeclared identifier undec")

fail_underscore.sk:
    input[int p, string q]
    output[int k]
```

```
    fsm hello {
            int _underscore = 4

       state Hello
                    if (p == 1) {
                            printf("%s", "Hello ")
                            goto World
                    }
                    else {
                            goto Hello
                    }
            state World
                    if (p == 1) {
                            printf("%s%s\n", q, "'s World")
                            goto Hello
                    }
                    else {
                            goto World
                    }
    }

fail_underscore.err:
    Fatal error: exception Parsing.Parse_error
```