SNL Final Report

by
James Lin
Alex Liu
Andre Paiva
Daniel Maxson

December 17 2014
Contents

1 Introduction .......................... 1
   1.1 Introduction .......................... 1
   1.2 Motivation .......................... 1
   1.3 Why call it Stage (null) language? 1

2 Language Tutorial .................. 2
   2.1 Compilation Walkthrough ........... 2
       2.1.1 Compiler Setup .................. 2
       2.1.2 Compile SNL to Java ............ 2
       2.1.3 Compile SNL to Java Executable 2
   2.2 Stages ............................ 3
   2.3 Recipes ............................ 3
   2.4 Variables .......................... 3
   2.5 Lists .............................. 3
   2.6 Control Flow ........................ 4
   2.7 Input/Output ........................ 4

3 Language Reference Manual ........ 5
   3.1 Lexical Elements .................. 5
       3.1.1 Comments ....................... 5
       3.1.2 Identifiers ..................... 5
       3.1.3 Keywords ....................... 5
       3.1.4 Literals ......................... 5
       3.1.5 Operators ....................... 6
       3.1.6 Seperators ...................... 6
       3.1.7 White Space ..................... 6
   3.2 Data Types ........................ 7
       3.2.1 Typing .......................... 7
       3.2.2 Built-in Data Types ............ 7
   3.3 Expressions and Operators ....... 7
   3.4 Recipes ............................ 8
3.4.1 Recipe Definitions .............................................. 8
3.4.2 Calling Recipes ............................................... 8
3.5 Program Structure and Scope ........................................... 9
  3.5.1 Program Structure ......................................... 9
  3.5.2 Stages .......................................................... 9
  3.5.3 Scope .......................................................... 9

4 Project Plan .................................................................. 10
  4.1 Project Processes .................................................. 10
    4.1.1 Planning ....................................................... 10
    4.1.2 Specification ............................................... 10
    4.1.3 Development ............................................... 10
    4.1.4 Testing ......................................................... 10
  4.2 Style Guide ........................................................ 11
  4.3 Team Responsibilities ............................................ 11
  4.4 Project Timeline .................................................. 12
  4.5 Development Environment ....................................... 12
  4.6 Project Log ........................................................ 12

5 Architectural Design .................................................. 13
  5.1 Overview ................................................................ 13
  5.2 Scanning ................................................................ 13
  5.3 Parsing .................................................................. 14
  5.4 Semantic Analysis ................................................ 14
    5.4.1 Java Generation .............................................. 14

6 Testing ..................................................................... 15
  6.1 Overview ............................................................. 15
  6.2 Test Suite File Listings ........................................... 16
    6.2.1 Expression Tests ............................................ 16
    6.2.2 Java Tests ....................................................... 16
    6.2.3 Program Tests ............................................... 17
    6.2.4 Statement Tests .............................................. 17
    6.2.5 Failing Tests ................................................... 17
  6.3 Example Tests ....................................................... 17
    6.3.1 Factorial ........................................................ 17
    6.3.2 Fibonacci ........................................................ 18

7 Lessons Learned ........................................................ 20
  7.1 James Lin ............................................................. 20
  7.2 Alex Liu ............................................................... 20
A Project Log

B Full Source Code

B.1 Parser ................................................................. 27
B.2 Scanner ............................................................... 30
B.3 AST ................................................................. 32
B.4 SAST ............................................................... 34
B.5 Semantic Analyzer .................................................. 35
B.6 Code Generator ..................................................... 41
B.7 Compiler ............................................................. 45
B.8 SNL Script .......................................................... 46
B.9 Java Backend ....................................................... 46
B.10 Test Script ......................................................... 54
B.11 Makefile ........................................................... 57
B.12 Tests ............................................................... 58
Chapter 1

Introduction

1.1 Introduction

SNL is a language designed to model role-playing game scenarios based on the structure of state machines. It allows users to programmatically re-create sequential scenarios as “stories.” Its syntax aims to be simple so that adolescents can easily understand how to use it. Where possible, SNL uses intuitive keywords in place of symbols, since symbols commonly known in the CS community can be confusing to those who are unfamiliar with programming. This will encourage children to learn how to write code in a controlled, fun environment while allowing them to exercise their creativity.

1.2 Motivation

Since our program will essentially model state diagrams, a “story” can be expressed in terms of a series of stages with transitions in between them in SNL. Recipes, the SNL version of functions, help with modularization. Despite its simplification of syntax and features, SNL is still robust enough to write programs at the same level as other common programming languages. These programs can range from algorithms like calculating GCD or factorial to more creative outlets like multi-ending chapter books or RPGs. A further application of SNL could be the generation of computerized surveys and studies within the social sciences.

1.3 Why call it Stage (null) language?

The core structure of the language of movement through a sequence of stages, which pretty obviously explains the first word. This is a programming language, which pretty obviously explains the third word. So why null? Well, the word “null” gets hated on so much in the computer science community that we decided we wanted to give it a positive spin. Plus, it allows us to have a funny acronym that reminds people of a funny TV show.
Chapter 2

Language Tutorial

2.1 Compilation Walkthrough

2.1.1 Compiler Setup

To compile the compiler, utilize the “make” command. This will create the SNL compiler “snlc”.

$ make

2.1.2 Compile SNL to Java

To compile SNL code to Java code, run the SNL compiler with the “-j” flag on the SNL file and the Java files will appear in the same level directory. Use the flag “-output_path <path>” to designate a specific path for the Java files. This can then be manually compiled with “javac” given that the file SNLObject.java is in the same level as “snlc”.

$ ./snlc -j program_name.snl

2.1.3 Compile SNL to Java Executable

To compile SNL code to a Java executable, run the SNL compilation script on the SNL file. You can then run this executable using Java.

$ ./snl program_name.snl

To run program, enter: java program_name

$ java program_name
2.2 Stages

The starting stage is the point at which an SNL program begins running. Subsequent stages are accessed by the call “next <stage name>”. Stages can be called any number of times and are terminated with the keyword “done”.

```
start stage1:
  next stage2
done
stage2:
done
```

2.3 Recipes

Recipes are essentially functions in our language. They can be defined and called by the user, and have the same stage flow as the top level of the program. Recipes are called with the format “do <recipe> to <arg1>, <arg2>, … <argn>”. Recipes can also return a value using the “return” keyword. Recipes can be called any number of times and are terminated with the keyword “done”.

```
recipe inc to number:
  start recipe_stage:
    return number + 1
  done
done
```

2.4 Variables

Variables do not have an explicitly assigned type in SNL, which utilizes dynamic typing similar to Python. Below is an example of a variable assignment using the “is” assignment keyword:

```
number is 24
str is “this is a string variable”
```

2.5 Lists

A list can also be stored as a variable. Lists are not restricted to a single type and are mutable. Lists have several additional operations and features outlined in the reference manual. These are all performed in the same way that recipes are called. Below is an example of a list being assigned to a variable:
lst is [1,2,4.5,’element’, 5>=2]

2.6 Control Flow

In order to keep the language intuitive for non-programmers, control flow has been reduced to if- and if-else statements. The conditionals can be modified using logical operators “=”, “and”, “or”, and “not”. All subsequent statements in the if- or if-else statement must be placed inside the parentheses.

if num < 0 or str = ‘hello’
(return 42)

2.7 Input/Output

Receiving input in the form of strings is done using the keyword “input”. Printing to standard output is done through the library function “show”.

do show to “hello world!”
person_to_greet is input
do show to “hello ” + person_to_greet
Chapter 3

Language Reference Manual

3.1 Lexical Elements

3.1.1 Comments

All comments are single-line and denoted by the # character. Any content to the right of the # will be ignored.

3.1.2 Identifiers

Identifiers are sequences of characters used for naming variables, functions, and stages. All characters must be alphanumeric or the underscore character. The first character must be an alphabetic character.

3.1.3 Keywords

```
if     else     not
and    or      do
to     start   next
is     local   true
false  return  recipe
done   input   of
```

3.1.4 Literals

There are several literals in SNL. These include integer literals, float literals, boolean literals, string literals, and list literals.

Integer Literals

An integer literal is a sequence of digits. All digits are taken to be decimal. 12 is an example of an integer constant.

Float Literals

A float literal consists of a decimal point, and either an integer part or a fraction part or both. 5.0
and 5. and .5 are all valid floating constants.

**Boolean Literals**
A boolean literals is either ‘true’ or ‘false’.

**String Literals**
A string literal is a sequence of chars. These are sequences of characters surrounded by double quotes. Two examples of string literals are “hello” and “world”.

**List Literals**
A list literal is a sequence of literals that have been placed between square brackets ‘[ ]’ and separated by commas ‘,’. Lists can contain one or more types and are mutable. [1,2,3,4] and [1,2,true,” peggy”] are both examples of lists.

### 3.1.5 Operators
An operator is a special token that performs an operation on two operands. More information about these are provided in the Expressions and Operations section (4).

### 3.1.6 Separators
A separator separates tokens. These are not seen as tokens themselves, but rather break our language into discrete pieces.

**White Space**
White space is the general purpose separator in SNL. More information is provided in the White Space section (2.7).

**Comma**
The comma is a separator, specifically in the context of creating lists (and their elements) and also for parameters passed to a function which is being called.

**Colon**
The colon is a separator in the context of starting a new stage or recipe. The separator will be placed right after the name of the stage or after the recipe declaration.

### 3.1.7 White Space
**Spaces**
Spaces are used to separate tokens within a single line outside of the creation of list and the first line of a stage.
Newline
Newlines are used to separate statements from one another. There is only one

## 3.2 Data Types

### 3.2.1 Typing

Variables in SNL are dynamically typed, similar to those in Python or Perl. Variables are implicitly assigned a type depending on the value assigned to it. You can find more information about these constants in the section about Literals (2.4).

### 3.2.2 Built-in Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>A series of digits</td>
</tr>
<tr>
<td>float</td>
<td>A series of digits with a single '.'</td>
</tr>
<tr>
<td>bool</td>
<td>Boolean values of true or false</td>
</tr>
<tr>
<td>string</td>
<td>A sequence of characters within &quot; &quot;</td>
</tr>
<tr>
<td>list</td>
<td>A sequence of items enclosed by [ ]</td>
</tr>
</tbody>
</table>

---

## 3.3 Expressions and Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition, string concatenation</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td></td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>!=</td>
<td>Not equals</td>
</tr>
<tr>
<td>and</td>
<td>Conjunction</td>
</tr>
<tr>
<td>or</td>
<td>Disjunction</td>
</tr>
<tr>
<td>not</td>
<td>Negation</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equals</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>=&gt;</td>
<td>Greater than or equals</td>
</tr>
<tr>
<td>()</td>
<td>Grouping expressions/statements</td>
</tr>
<tr>
<td>is</td>
<td>Assignment</td>
</tr>
<tr>
<td>of</td>
<td>Access element from list</td>
</tr>
</tbody>
</table>
3.4 Recipes

3.4.1 Recipe Definitions

A recipe is set of stages with a separate global scope. If there are any arguments passed into the recipe, the ‘to’ keyword must come before the comma-separated list of arguments. The ‘return’ keyword will return at most one item back to the stage from which it was initially called.

An example of a recipe built using multiple stages:

```plaintext
start example_program:
  lst is [3, 4, 5, 6]
  do inc_list to lst
  show lst
done

recipe inc_list to my_list: # declaration of recipe

  start start_inc_list:
    length is do get_len to my_list # calling a recipe
    index is 0
    next loop_start
  done

  loop_start:
    if index < length
      (next s_list_modifier)
    else (return my_list) # returning out of our recipe
  done

  s_list_modifier:
    index of my_list is index of my_list + 1
    index is index + 1
    next loop_start
  done
done
```

Listing 3.1: Recipe Example

3.4.2 Calling Recipes

The keywords ‘do’ and ‘to’ mark recipe calls, and the comma is used to separate function arguments. For example:

do foo to bar, baz

When there are no arguments to a recipe, ‘to’ must be omitted such as:

do foo
3.5 Program Structure and Scope

3.5.1 Program Structure

Each program must be written within one source file and are a combination of a single Universe along with Stages and Recipes. These can each be defined anywhere within the file.

3.5.2 Stages

A Stage will consist of a series of statements. The starting Stage for each recipe or program will be specified by the the ‘start’ keyword. Next will come the name of the Stage followed by a colon. For all Stages outside of the starting Stage of a recipe or program, only the name of the Stage and the colon should be used.

Within a Stage, the ‘next’ keyword will designate the following Stage to jump to. These will control the movement of the Character between different Stages, particularly by utilizing conditional statements to vary between different next Stages.

3.5.3 Scope

Global Scope
All variables defined either in the Universe or a Stage are by default part of the global scope and can be accessed and modified from any of the other stages within the program.

Scope within a Stage
To declare a variable at a Stage scope you will use the reserved keyword ‘local’ followed by the variable name. For example:

local colour_of_ball is "blue"

Scope within a Recipe
A recipe does not have any access to the Universe scope but will only have access to any items passed in or declared within this recipe. Users must be careful to remember which recipe they are declaring variables in at each stage.
Chapter 4

Project Plan

4.1 Project Processes

4.1.1 Planning

We decided to meet every Tuesday night at 11 pm in order to plan and work on the SNL compiler. This was a good time for all of us, but also proved to be one of the few times during the week our team could set aside to work on the compiler.

4.1.2 Specification

Our proposal and LRM were key for identifying the required tools and features of SNL. These were particularly helpful because we had a strong vision for our language from the beginning. The LRM was particularly helpful and provided the baseline for all of the different components that ended up comprising our language. There were obviously on-the-fly additions to the language as we saw fit in the late nights making our compiler.

4.1.3 Development

Our general process for implementing features was to plan out the feature, then write a test in SNL for the feature (using a .snl file and a corresponding .out file), and finally implement the feature. This was incredibly useful and a process that we largely picked up from our awesome Language Guru.

4.1.4 Testing

At the early stages of development, we tested our AST for the correctness of its expressions and statements as well as overall program structure. These were done by printing out the AST and comparing it with expected output. When code generation and compilation were implemented, we wrote many tests to match output of executables to ensure correctness. Through this process, we were able to catch several major bugs and apply fixes. We implemented a script to perform
large scale testing and facilitate easy addition of new tests to be performed each time changes were made to ensure changes would not break existing code.

4.2 Style Guide

We followed rules for style:

- No more than 80 characters per line of code
- Automatic Tuareg and Omlet style formatting in Emacs and Vim, respectively
- SNL is written with standard Python formatting, using lowercase letters and underscores for names when possible
- Java syntax was used for backend Java whenever possible
- Modularization was prioritized through intensive re-factoring of source code

4.3 Team Responsibilities

Our team chemistry took a bit of time to fully take form, but came together nicely in the end. The parts that each team member worked on are listed below. We used Github for version control and sharing the same source code. The project log will be included in the appendix.

**Language Guru: Alex Liu**
scanner, parser, AST, analyzer, testing suite (OCaml, Python)

**System Architect: Daniel Maxson**
SAST, analyzer, codegen (OCaml)

**Testing and Validation: Andre Paiva**
tests (SNL, Python)

**Project Manager: James Lin**
Java backend, tests, codegen (Java, SNL, OCaml)
4.4  Project Timeline

<table>
<thead>
<tr>
<th>Date</th>
<th>Task Finished</th>
</tr>
</thead>
<tbody>
<tr>
<td>9/24</td>
<td>Project Proposal Done</td>
</tr>
<tr>
<td>10/27</td>
<td>LRM Done</td>
</tr>
<tr>
<td>11/1</td>
<td>Scanner and Parser Done</td>
</tr>
<tr>
<td>11/17</td>
<td>Testing Suite Constructed</td>
</tr>
<tr>
<td>12/10</td>
<td>SAST / Analyzer Done</td>
</tr>
<tr>
<td>12/12</td>
<td>Java Object Abstraction Done</td>
</tr>
<tr>
<td>12/14</td>
<td>Core Compiler and Code Generation Done</td>
</tr>
<tr>
<td>12/15</td>
<td>Library Functions Completed</td>
</tr>
<tr>
<td>12/16</td>
<td>Presentation and Final Report</td>
</tr>
</tbody>
</table>

4.5  Development Environment

Our entire team was effectively using Mac OS X due to the Windows 7 teammate’s computer not functioning correctly. Our OCaml was version 4.02.1; our Python was version 2.7.6; our Java was Java 6. We used git on Github for version control. For text editing, we used a combination of Vim, Emacs, and Sublime Text. We used Bash and Python scripts along with Makefiles for automation.

4.6  Project Log

Located in Appendix A
Chapter 5

Architectural Design

5.1 Overview

The SNL compiler consists of the major components seen in most compiler designs: scanner, parser, semantic analyzer and code generator. From the diagram below you can see the order of these processes.

Figure 5.1: Process Flow of SNL Compiler

The scanning and parsing was primarily done by Alex, the semantic analysis done by Alex and Daniel, the Java code generation done by Daniel and James while the Java implementation was done by James. Andre was responsible for testing all components of the compiler.

5.2 Scanning

The SNL scanner, written using ocamllex, takes all the input from the a supplied .snl file and tokenizes it. In this stage we remove all whitespace (except for newlines) and comments and throw exceptions for any illegal characters. The scanner also removes certain newlines to produce a token stream that can be described by an unambiguous grammar in the parsing phase.
5.3 Parsing

The parser, written using ocamlyacc, takes in the tokens generated during the scanning stage and produces an abstract syntax tree (AST) from an unambiguous grammar. This process catches any syntax errors and populates the tree with stages, recipes, and their respective components.

5.4 Semantic Analysis

After the AST is produced the semantic analyzer walks through the tree annotating each stage, recipe, expression and statement creating the SAST. Errors in variable scope are checked but there is minimal type checking since our language is dynamically typed.

After building the SAST, the analyzer runs through recipe and stage names to determine if any of them are duplicated. The analyzer then checks to make sure there is exactly one starting stage. It also builds and traverses a directed graph of the stages, as linked by ‘next’ statements, so it can warn if there are unreachable stages or return errors if ‘next’ calls a stage that does not exist. Recipe calls are checked to ensure that the recipe is either a valid library recipe or defined in the program and that the appropriate number of arguments is passed in.

5.4.1 Java Generation

The final part of our compiler runs through the checked SAST and prints out the appropriate Java code. There is one Java file for the main set of stages and an additional Java file for each of the recipes. The library functions are built directly into the code generation. This Java code is then compiled using the snl compile script.
Chapter 6

Testing

6.1 Overview

Each test was stored as a combination of two files in the tests directory: the SNL program and the expected result. Our team wrote a Python script that would attempt to reproduce the expected output based on the .snl file fed in to it and compare this output with the .out file.

We separated these tests into five different folders. For the files in expr, program, and stmt, our script would produce the AST output. The failing folder contained some tests with simple errors to make sure that our compiler failed properly on certain syntax errors. Finally, the java directory was reserved for testing the correctness of our compiled Java code. Here, we sought to validate every feature of our language (operators, library functions, etc.) while also trying our best to catch subtler loose ends (processing comments interspersed with newlines, naming stages and recipes after Java keywords).

The AST testing was done by Alex, and the Java tests were written mainly by Andre.
6.2 Test Suite File Listings

6.2.1 Expression Tests

<table>
<thead>
<tr>
<th>add.snl</th>
<th>geq.snl</th>
<th>neq.snl</th>
</tr>
</thead>
<tbody>
<tr>
<td>and.snl</td>
<td>gt.snl</td>
<td>next.snl</td>
</tr>
<tr>
<td>assign1.snl</td>
<td>id1.snl</td>
<td>not.snl</td>
</tr>
<tr>
<td>assign2.snl</td>
<td>id2.snl</td>
<td>or.snl</td>
</tr>
<tr>
<td>assign3.snl</td>
<td>id3.snl</td>
<td>paren-int.snl</td>
</tr>
<tr>
<td>bool1.snl</td>
<td>id4.snl</td>
<td>return.snl</td>
</tr>
<tr>
<td>bool2.snl</td>
<td>input.snl</td>
<td>seq1.snl</td>
</tr>
<tr>
<td>call1.snl</td>
<td>int.snl</td>
<td>seq2.snl</td>
</tr>
<tr>
<td>call2.snl</td>
<td>leq.snl</td>
<td>seq3.snl</td>
</tr>
<tr>
<td>call3.snl</td>
<td>lt.snl</td>
<td>seq4.snl</td>
</tr>
<tr>
<td>call4.snl</td>
<td>math1.snl</td>
<td>string1.snl</td>
</tr>
<tr>
<td>div.snl</td>
<td>math2.snl</td>
<td>string2.snl</td>
</tr>
<tr>
<td>equal.snl</td>
<td>math3.snl</td>
<td>string3.snl</td>
</tr>
<tr>
<td>float1.snl</td>
<td>math4.snl</td>
<td>string4.snl</td>
</tr>
<tr>
<td>float2.snl</td>
<td>mult.snl</td>
<td>sub.snl</td>
</tr>
<tr>
<td>float3.snl</td>
<td>negate.snl</td>
<td></td>
</tr>
</tbody>
</table>

6.2.2 Java Tests

<table>
<thead>
<tr>
<th>add1.snl</th>
<th>func3.snl</th>
<th>neg1.snl</th>
</tr>
</thead>
<tbody>
<tr>
<td>add2.snl</td>
<td>gcd.snl</td>
<td>number_to_word2.snl</td>
</tr>
<tr>
<td>add3.snl</td>
<td>global1.snl</td>
<td>number_to_word.snl</td>
</tr>
<tr>
<td>add4.snl</td>
<td>global2.snl</td>
<td>ops2.snl</td>
</tr>
<tr>
<td>add5.snl</td>
<td>hello_world.snl</td>
<td>ops.snl</td>
</tr>
<tr>
<td>add6.snl</td>
<td>id1.snl</td>
<td>perform1.snl</td>
</tr>
<tr>
<td>add7.snl</td>
<td>if1.snl</td>
<td>remove.snl</td>
</tr>
<tr>
<td>append2.snl</td>
<td>if2.snl</td>
<td>stmt1.snl</td>
</tr>
<tr>
<td>append.snl</td>
<td>if3.snl</td>
<td>string1.snl</td>
</tr>
<tr>
<td>bool1.snl</td>
<td>if4.snl</td>
<td>sub1.snl</td>
</tr>
<tr>
<td>bool2.snl</td>
<td>if5.snl</td>
<td>sub2.snl</td>
</tr>
<tr>
<td>comment1.snl</td>
<td>insert.snl</td>
<td>sub3.snl</td>
</tr>
<tr>
<td>factorial2.snl</td>
<td>int1.snl</td>
<td>sub4.snl</td>
</tr>
<tr>
<td>factorial.snl</td>
<td>length.snl</td>
<td>var2.snl</td>
</tr>
<tr>
<td>fib2.snl</td>
<td>list1.snl</td>
<td>var3.snl</td>
</tr>
<tr>
<td>fib.snl</td>
<td>list2.snl</td>
<td>var4.snl</td>
</tr>
<tr>
<td>float1.snl</td>
<td>list3.snl</td>
<td>while1.snl</td>
</tr>
<tr>
<td>func1.snl</td>
<td>list4.snl</td>
<td>whitespace.snl</td>
</tr>
<tr>
<td>func2.snl</td>
<td>name.snl</td>
<td></td>
</tr>
</tbody>
</table>
6.2.3 Program Tests

program1.snl  stage1.snl
recipe1.snl  stage2.snl

6.2.4 Statement Tests

if1.snl  if3.snl
if2.snl  newline1.snl
newline2.snl

6.2.5 Failing Tests

invalid_next.snl
missing_done.snl
missing_endquote.snl

6.3 Example Tests

6.3.1 Factorial

Listing 6.1: Factorial SNL Source Code: factorial.snl

import java.util.Scanner;
public class factorial2{
  private static Scanner input = new Scanner(System.in);
  public static void main(String args[]){
    s_init();
  }
  private static void s_init{
num = new SNOBject(5, "int");
total = new SNOBject(1, "int");
s_stage1();
return;
}

private static void s_stage1(){
if(num.eq( new SNOBject(0, "int")).getBool())
{s_last();
   return;
}
else()
    total= total.mult(num);
num= num.sub( new SNOBject(1, "int"));
s_stage1();
return;
}

private static void s_last(){
    System.out.println(total);
}

private static SNOBject total;
private static SNOBject num;

Listing 6.2: Factorial Compiled Java Code: factorial.java

6.3.2 Fibonacci

recipe fib to x:
  start fib:
    if x < 2
      (return 1)
    else
      (return (do fib to x − 1) + (do fib to x − 2))
  done

start main:
  do show to do fib to 0
do show to ""
do show to do fib to 1
do show to ""
do show to do fib to 2
do show to ""
do show to do fib to 3
do show to ""
do show to do fib to 4
do show to ""
do show to do fib to 5
do show to ""
do show to do fib to 10
do show to ""
  done

Listing 6.3: Fibonacci SNL Source Code: fib.snl
import java.util.Scanner;

public class fib{
    private Scanner input = new Scanner(System.in);

    public static void main(String args[]){
        s_main();
    }

    private static void s_main(){
        System.out.println(new Recipe_fib().perform( new SNObject(0)));
        System.out.println(new Recipe_fib().perform( new SNObject(1)));
        System.out.println(new Recipe_fib().perform( new SNObject(2)));
        System.out.println(new Recipe_fib().perform( new SNObject(3)));
        System.out.println(new Recipe_fib().perform( new SNObject(4)));
        System.out.println(new Recipe_fib().perform( new SNObject(5)));
        System.out.println(new Recipe_fib().perform( new SNObject(10)));
    }

    private void s_fib(){
        if(x.lt( new SNObject(2)).getBool())
            {ret = new SNObject(1);
            return;
        }
        else{ret = new Recipe_fib().perform(x.sub( new SNObject(1))).add(new Recipe_fib().
            perform(x.sub( new SNObject(2))));
            return;
        }
    }

    private SNObject x;
}

Listing 6.4: Fibonacci Compiled Java Code: fib.java

import java.util.Scanner;

public class Recipe_fib{
    private SNObject ret;
    private Scanner input = new Scanner(System.in);

    public Recipe_fib(){
    }

    public SNObject perform(SNObject x_arg){
        x = new SNObject(x_arg);
        s_fib();
        return ret;
    }

    private void s_fib(){
        if(x.lt( new SNObject(2)).getBool())
            {ret = new SNObject(1);
            return;
        }
        else{ret = new Recipe_fib().perform(x.sub( new SNObject(1))).add(new Recipe_fib().
            perform(x.sub( new SNObject(2))));
            return;
        }
    }

    private SNObject x;
}

Listing 6.5: Recipe For Fibonacci Compiled Java Code: Recipe_fib.java
Chapter 7

Lessons Learned

7.1 James Lin

Undertaking this project was incredibly fruitful for me on many levels. Though I have experience with project management, I have never done this with a programming project of this scale. I was very humbled by this experience, because I realized how familiarity with the project and its components is key to proper management. At the beginning of the course and the project, I did not really understand the technical aspects of the project. As a result, my leadership ability was severely crippled and I was unable to make wise decisions. I am determined not to be this kind of manager in the future and do my best to invest in understanding the ins and outs of my future projects.

Two other important areas of growth were the utilization of git and a testing-first mentality. Again, I only had minimal exposure to both of these in my software engineering career thus far. These projects pushed me and challenged me to use these tools to their maximum potential. I was also able to pick up handy additions to my shell and other development environments.

My suggestion for future groups would be to pair program as early as possible in the development process, especially if there are disparities in technical understanding. Our productivity increased greatly when we had two sets of eyes and two brains tackling a single problem at a time. It also made merge conflicts much easier to deal with. I will be sure to use this in my future career. Thank you to Professor Edwards for this excellent class!

7.2 Alex Liu

If you're the language guru, debugging the scanner and parser is a lot easier when you understand how shift-reduce parsing works. As the internet helpfully suggests, use the parser output (run verb | ocamllyacc -v filename.mly) and set the OCamlRUNPARAM environment variable to p (run export OCamlRUNPARAM=’p’ in bash) so you can see all the stages the parser is passing through on what input and figure out why the dreaded exception Parsing.Parse_error pops up.
Also, the language your group envisions may actually be ambiguous if you can only look one character ahead, and there may be no way to easily and crisply describe it for your parser. One way to address this problem is to come up with a standardized form of the language that is more easily described with an unambiguous context-free grammar. Then, use the scanner to strip away all the characters or sequences that are nice to allow in the language but difficult to parse, and run the parser on this transformed language.

OCaml is actually a nice language and the compiler is very helpful in figuring out what's wrong with your code. What’s even better is if you use an editor with automatic indentation and syntax highlighting, like tuareg-mode in Emacs.

Having lots of regression tests is extremely important because if you're constantly refactoring code, especially if it's code mainly authored by your teammates, you need to know when you've broken something. Knowing that you can check if you've safely refactored code should encourage you to do so more often, which is great because it promotes clarity.

Pair programming is really fun! It also produces fewer bugs and makes quashing them faster, especially if no member in your group is very familiar with OCaml. Chances are that no member remembers all of the OCaml syntax and also that each remembers different things. When programming together, you get a greater coverage of the language.

### 7.3 Daniel Maxson

This was probably one of the most complex systems I have been a part of building. My role as the software architect placed me in a position where understanding the big picture and how all the components tie together was crucial. Through this project I learned the importance of building and using a regression test suite from the earliest stages of development – it allowed us to be confident that the changes we had made didn't break other components of the whole system. I also became a lot more comfortable with git, particularly tricky git merges which I had always tried to avoid before this project.

Although I never thought I would be saying this I actually came to enjoy programming in OCaml, there are a lot of things which initially don't seem intuitive but after practice it slowly starts to become more familiar. In general it allows you to write a pretty complex system in a very concise manner.

Probably in terms of advice for future teams I would suggest you to find a time to meet to allow you to do most of your programming in the same room as your other team-mates. This allows you to quickly bounce questions off of each other without wasting time in an email exchange. Also I can not stress enough the advantage of pair programming. Having an extra mind and set of eyes
looking at the code helps with catching bugs and makes solving problems a lot faster – it’s also more fun than coding by yourself. Also become comfortable with your coding environment – you will be spending endless hours in front of your terminal or text editor so set it up in a way that allows you to focus and be the most efficient. If you are a Vim user download and install Omlet for automatic OCaml styling. Also do your eyes a favor and install flux.

All the advice we got told us to start early, this probably would have made sense but if you find yourself like we did with almost nothing to show and one or two weeks left in the semester don’t lose heart. Just realize that you’ll be seeing an awful lot of your teammates in the days leading up to the deadline.

7.4 Andre Paiva

Testing was perhaps both the least and most stressful part of the project - it's fun to do your best to break things, and unsettling to realize that you've effectively broken a program you've been obsessing over for the past two or three weeks (the night before said program is due). Do your best to write the most outlandish scenarios possible (pay attention to one lecture where Prof. Edwards will point out strange undefined behaviors in C!)

Weekly meetings are a must, to make sure everyone reasons through broad decisions about the language and to simply force the group to work on something for the week (thanks to this we never discovered the limitations of email for project collaboration). This is much easier in September - when crunch time comes, your group might want to allow for some degree of flexibility in the precise schedule. Other than that, make sure to learn git, don’t set up grand expectations for your language, don’t fret if it doesn’t meet your lower expectations either.

This project is quite a departure from anything else undergraduates have seen in an academic setting. Don’t be daunted: as long as you follow the advice given by Prof. Edwards and read about the projects of those who came before, you should be more than well prepared to cook up something respectable. (though if you're reading this, you either don't need this advice or are five minutes from giving our group a grade [comment this part out?]).
Appendix A

Project Log

Note: Andre Paiva has minimal commits because his computer had problems during the time of development. As such, he was mainly using James Lin's and Daniel Maxson's computers in order to commit.

1. [2014-12-16, A. Liu] Refactor SNLObject.java, checked style formatting
3. [2014-12-16, A. Liu] scanner lint
5. [2014-12-16, James Lin] presentation examples
6. [2014-12-16, A. Liu] Hopefully made scanner strip comments better
7. [2014-12-16, tinyvm] Probation
8. [2014-12-16, tinyvm] Couple of more cases
9. [2014-12-16, A. Liu] codegen working
10. [2014-12-16, A. Liu] Continuing edits to fix block generation
12. [2014-12-16, A. Liu] Changes to language to support stmts inside stmts
13. [2014-12-16, James Lin] fixed tests
14. [2014-12-16, tinyvm] work test work
16. [2014-12-16, Daniel Maxson] added number to string functionality
18. [2014-12-16, tinyvm] fixed append test
20. [2014-12-16, James Lin] fixed snlobjct error
23. [2014-12-16, James Lin] adding number_to_word feature
24. [2014-12-16, tinyvm] append test
25. [2014-12-16, A. Liu] More changes to parser concerning newlines, fix string test out
27. [2014-12-16, James Lin] fixed string bug
28. [2014-12-16, tinyvm] Ok, no input tests then.
31. [2014-12-16, A. Liu] Allow for actual multiple newlines before program
32. [2014-12-16, Andre Paiva] Input tests anyone?
33. [2014-12-16, A. Liu] add diagnostics for recipe internals
34. [2014-12-16, A. Liu] Add diagnostics checking to recipe internals
35. [2014-12-16, Daniel Maxson] merging
36. [2014-12-16, Daniel Maxson] added word_to_number library function
37. [2014-12-16, James Lin] fixed word_to_number library function
38. [2014-12-16, James Lin] added word_to_number library function
39. [2014-12-16, James Lin] factorial tests
[2014–12–16, Daniel Maxson] fixed a bug with the Next statement
[2014–12–16, James Lin] fixed bug in snlobject
[2014–12–15, Daniel Maxson] added list length
[2014–12–15, Daniel Maxson] merge’
[2014–12–15, Daniel Maxson] added list library functions
[2014–12–15, James Lin] added remove_back functionality
[2014–12–15, James Lin] adding list feature tests
[2014–12–15, James Lin] re-organizing executables and scripts
[2014–12–15, Daniel M.] Update TODO
[2014–12–15, A. Liu] finish analysis error generation, updating tests
[2014–12–15, A. Liu] Recoding analyzer’s error finding
[2014–12–15, James Lin] fixed while result
[2014–12–15, Daniel Maxson] add s_to all stage names
[2014–12–15, James Lin] fib results fixed
[2014–12–15, Daniel Maxson] fixed problem with static variables messing up recursion
[2014–12–15, A. Liu] Fix a test, allow for NEWLINE before program starts
[2014–12–15, Daniel Maxson] re-added code to clear the global hashtbl for each recipe and fixed formal argument bug
[2014–12–15, A. Liu] Continuing to refactor codegen
[2014–12–15, A. Liu] Starting to refactor codegen
[2014–12–14, James Lin] housekeeping
[2014–12–14, James Lin] house cleaning and added list features
[2014–12–14, Daniel Maxson] merging
[2014–12–14, Daniel Maxson] fixed a bug with block statements
[2014–12–14, A. Liu] modifications to analyzer to allow global var access
[2014–12–14, James Lin] fixed append
[2014–12–14, James Lin] fixed block statements
[2014–12–14, Daniel Maxson] added local — Analyzer isn’t working though.. see testing.snl
[2014–12–14, James Lin] Added five tests
[2014–12–14, Daniel Maxson] fixed some parser bugs — only local needs to work
[2014–12–14, James Lin] revised list example

24
[2014−12−14, James Lin] switched to array implementation
[2014−12−14, Daniel Maxson] removed extra checks from analyzer and finished recipes
[2014−12−14, Daniel Maxson] working on recipes
[2014−12−14, A. Liu] Add formals to recipe global scope
[2014−12−14, A. Liu] Remove unnecessary array of types from TList
[2014−12−14, Daniel Maxson] almost working recipes
[2014−12−14, A. Liu] lint run_tests.py
[2014−12−14, A. Liu] Change list access from int * expr to expr * expr
[2014−12−14, Daniel Maxson] started work on recipes
[2014−12−14, Daniel Maxson] merge conflict resolved
[2014−12−14, Daniel Maxson] working ID's and assignments
[2014−12−14, James Lin] Andre doing that testing business
[2014−12−14, James Lin] fixed file path name stuff
[2014−12−14, Daniel Maxson] just in case
[2014−12−13, Daniel Maxson] Added some tests
[2014−12−13, Daniel Maxson] adding support for lists
[2014−12−13, Daniel Maxson] fixing a merge conflict
[2014−12−13, Daniel Maxson] forced git commit
[2014−12−13, Daniel Maxson] refactored code
[2014−12−13, James Lin] merge conflict
[2014−12−13, James Lin] making changes
[2014−12−13, A. Liu] editing run_tests script and snl.ml
[2014−12−13, James Lin] tests for java gen
[2014−12−13, James Lin] changes to list structure, codegen, naming
[2014−12−13, James Lin] resolved merge conflict
[2014−12−13, James Lin] partial work for merge
[2014−12−13, Daniel Maxson] added return
[2014−12−13, James Lin] next stage and remove java files before codegen
[2014−12−13, James Lin] working on adding tests
[2014−12−13, James Lin] rm
[2014−12−13, Daniel Maxson] added input and printing
[2014−12−13, Daniel Maxson] working print function
[2014−12−13, Daniel Maxson] merge
[2014−12−13, Daniel Maxson] added unary and binary operations
[2014−12−12, A. Liu] Modified run_tests.py to run failing tests
[2014−12−12, Daniel Maxson] merge conflict
[2014−12−12, James Lin] fixed merge conflict
[2014−12−12, Daniel Maxson] working constants
[2014−12−12, James Lin] snl can read from file
[2014−12−12, A. Liu] Adding tests to make sure syntax errors cause failure and for compiler system
[2014−12−12, James Lin] got list example working, fixed list operations in snlobject
[2014−12−12, Daniel Maxson] merging
[2014−12−12, Daniel Maxson] starting to work on code generation
[2014−12−12, James Lin] changed source copy as well
[2014−12−12, James Lin] product java complete and edits to snl
[2014−12−12, Daniel Maxson] merging
[2014−12−12, James Lin] small test file
[2014−12−12, Daniel Maxson] merging
[2014−12−12, Daniel Maxson] started to work on code gen
[2014−12−12, James Lin] snlobject for generating java
[2014−12−12, A. Liu] Add more newlines in the parser and add tests
[2014–12–11, James Lin] fixed automated test break
[2014–12–10, James Lin] work on code gen
[2014–12–10, James Lin] remove class files accidentally committed
[2014–12–10, A. Liu] space
[2014–12–10, A. Liu] Finished with sast creation
[2014–12–10, A. Liu] JK sast not done but annotate_expr should be
[2014–12–09, A. Liu] Sast is built, working on finishing type checking
[2014–12–09, James Lin] updated compilation examples
[2014–12–03, Daniel Maxson] Merging
[2014–12–03, Daniel Maxson] Working on analyzer, got through compiler with TODOs left
[2014–12–03, James Lin] answered some questions
[2014–12–03, James Lin] humanly generated java with commentary
[2014–11–26, Daniel Maxson] added a todo list
[2014–11–26, Daniel Maxson] fixed a merge conflict
[2014–11–26, Daniel Maxson] started working on SAST and semantic analyzer
[2014–11–25, James Lin] starting code generation
[2014–11–17, A. Liu] Start adding program tests, fix a bug in parser
[2014–11–13, A. Liu] Add more tests, fix some parser stuff, lint files
[2014–11–03, James Lin] python script for outputting AST
[2014–11–03, A. Liu] Add more expr tests, refactor test script
[2014–11–03, A. Liu] Writing tests for the language
[2014–11–03, A. Liu] Starting to test everything by printing out ASTs
[2014–11–01, James Lin] added input in scanner and parser
[2014–10–27, A. Liu] Parser and scanner theoretically in line with current LRM but untested
[2014–10–25, Daniel M.] Update README.md
[2014–10–25, Daniel M.] Update README.md
[2014–10–25, Daniel M.] Update README.md
[2014–10–25, Daniel M.] Update README.md
[2014–10–25, Daniel M.] Update README.md
[2014–10–25, Daniel M.] Update README.md
[2014–10–25, Daniel M.] Update README.md
[2014–10–25, Daniel M.] Update README.md
[2014–10–25, Daniel M.] Update README.md
[2014–10–24, A. Liu] Add statements to parser, move some files around
[2014–10–22, A. Liu] fix list construction
[2014–10–22, A. Liu] add lists to parser, rename some vars
[2014–10–22, A. Liu] Begin to work on parser, only very basics down
[2014–10–15, James Lin] edwards code micro c as testing example
[2014–10–15, A. Liu] change double to single quotes for \n
[2014–10–14, A. Liu] add binary search example
[2014–09–24, James Lin] sample code for proposal
[2014–09–24, James Lin] first commit
Appendix B

Full Source Code

B.1 Parser

```plaintext
%%

%( open Ast %)
%token COMMENT COLON LPAREN RPAREN LBRACKET RBRACKET COMMA
%token PLUS MINUS TIMES DIVIDE ASSIGN
%token EQ NEQ LT LEQ GT GEQ
%token IF ELSE AND OR NOT TRUE FALSE
%token RECIPE DONE START NEXT RETURN DO TO OF LOCAL INPUT
%token <int> INT
%token <string> ID STRING
%token NEWLINE EOF

%nonassoc NOCOMMA
%nonassoc COMMA
%nonassoc NOELSE
%nonassoc ELSE
%nonassoc RETURN
%nonassoc DO TO
%right ASSIGN
%left AND OR
%right NOT
%left EQ NEQ LT LEQ GT GEQ
%right OF
%left PLUS MINUS
%left TIMES DIVIDE
%nonassoc UMINUS
%nonassoc LPAREN RPAREN

%start expr
%type <Ast.expr> expr

%start stmt
%type <Ast.stmt> stmt

%start program
%type <Ast.program> program

%%

/* Matches NEWLINE* */
```
opt_nln:
    /* nothing */ %prec NOCOMMA { }
| multi_nln %prec NOCOMMA { }

/* Matches NEWLINE */
multi_nln:
    NEWLINE %prec COMMA { }
| multi_nln NEWLINE %prec COMMA { }
/* int, float, bool, string literals. */
constant:
    INT   { Int($1) }
| FLOAT  { Float($1) }
| TRUE   { Bool(true) }
| FALSE  { Bool(false) }
| STRING { String($1) }
/* Ids may be local or global. */
ids:
    ID     { Id($1, Global) }
| LOCAL ID { Id($2, Local) }
/* Exprs are the basic building blocks of programs. */
* No newlines are allowed inside. */
expr:
    constant         { Constant($1) }
| LPAREN expr RPAREN { $2 }
| ids              { $1 }
| LBRAKCET expr_seq RBracket { List($2) }
| expr ASSIGN expr  { Assign($1, $3) }
| math             { $1 }
| logic            { $1 }
| recipe_app       { $1 }
| RETURN expr      { Return($2) }
| NEXT ID          { Next($2) }
| INPUT            { Input } 
| expr OF ids      { Access($1, $3) }
/* Mathematical expressions. */
math:
    expr PLUS expr   { Binop($1, Add, $3) }
| expr MINUS expr  { Binop($1, Sub, $3) }
| expr TIMES expr  { Binop($1, Mul, $3) }
| expr DIVIDE expr { Binop($1, Div, $3) }
| MINUS expr %prec UMINUS { Unop(Negate, $2) }
/* Boolean expressions. */
logic:
    expr EQ expr    { Binop($1, Equal, $3) }
| expr NEQ expr   { Binop($1, Neq, $3) }
| expr LT expr    { Binop($1, Lt, $3) }
| expr LEQ expr   { Binop($1, Leq, $3) }
| expr GT expr    { Binop($1, Gt, $3) }
| expr GEQ expr   { Binop($1, Geq, $3) }
/* A sequence is a comma-separated succession of exprs. It can be used inside * brackets to define a list or when defining or applying recipes. */
expr_seq:
  /* nothing */ %prec NOCOMMA { [] }
  | expr_seq_builder %prec NOCOMMA { List.rev $1 }

expr_seq_builder:
  expr %prec NOCOMMA { [$1] }
  | expr_seq_builder COMMA expr { $3 :: $1 }

/* Applying recipes. */
recipe_app:
  DO ID TO expr_seq_builder %prec NOCOMMA { Call($2, List.rev $4) }
  | DO ID { Call($2, []) }

/* A statement is either an expression or an if–else construct. */
stmt:
  expr %prec NOELSE ( Expr($1) )
  | IF expr multi_nl LPAREN block_builder RPAREN
    ELSE opt_nl LPAREN block_builder RPAREN
    { If($2, Block(List.rev $5), Block(List.rev $10)) }
  | IF expr multi_nl LPAREN block_builder RPAREN %prec NOELSE
    { If($2, Block(List.rev $5), Block([])) }

/* A block is a sequence of expr separated by newlines that appears in an if statement. */
block_builder:
  stmt { [$1] }
  | block_builder multi_nl stmt { $3 :: $1 }

stage_body:
  stmt { [$1] }
  | stage_body opt_nl stmt { $3 :: $1 }

stage:
  ID COLON multi_nl stage_body opt_nl DONE { { sname = $1;
    body = List.rev $4;
    is_start = false } }
  | START ID COLON multi_nl stage_body opt_nl DONE { { sname = $2;
    body = List.rev $5;
    is_start = true } }

formal_list:
  ID { [$1] }
  | formal_list COMMA ID { $3 :: $1 }

stage_seq:
```
stage   { [1] }
| stage_seq opt_nl stage { $3 :: $1 }

recipe:
  RECIPE ID COLON multi_nl
  stage_seq opt_nl DONE
  { { name = $2;
      formals = [];
      body = List.rev $5; } }
| RECIPE ID TO formal_list COLON multi_nl
  stage_seq opt_nl DONE
  { { name = $2;
      formals = List.rev $4;
      body = List.rev $7; } }

program_body:
  stage opt_nl { { recipes = [];
      stages = [$1]; } }
| recipe opt_nl { { recipes = [$1];
      stages = []; } }
| stage opt_nl program_body { { recipes = $3.recipes;
      stages = $1 :: $3.stages; } }
| recipe opt_nl program_body { { recipes = $1 :: $3.recipes;
      stages = $3.stages; } }

program:
  /* nothing */ { { recipes = [];
      stages = []; } }
| program_body { $1 }
| multi_nl program_body { $2 }
```

Listing B.1: parser.mly

### B.2 Scanner

```
{ open Parser }

let digit = ['0'..'9']
let whitespace = [' ' '	' '
']
let comment = "#" [^ "\n"]+ "\n"
let ws_strip = (whitespace|comment|'\n')*

rule tokenize = parse
  (* Whitespace we split on. *)
  whitespace { tokenize lexbuf }
| comment { tokenize lexbuf }

  (* Comments. *)
  | "#" { tokenize lexbuf }

  (* Binary operators: math, comparison, and logic. *)
  | "+" { PLUS }
  | "-" { MINUS }
  | "+" { TIMES }
  | "/" { DIVIDE }
  | "=" { EQ }
  | "+" { NEQ }
```
"<" { LT } | "<=" { LEQ } | ">

"=" { EQ } | ">=" { GEQ } | ">

"and" { AND } | ">" { GT } | ">=" { GEQ } | ">

"or" { OR } | ">" { GT } | ">=" { GEQ } | ">

"not" { NOT } | ">" { GT } | ">=" { GEQ } | ">

(* Control flow. *) | ">" { GT } | ">=" { GEQ } | ">

"if" { IF } | ">" { GT } | ">=" { GEQ } | ">

ws_strip "else" { ELSE } | ">" { GT } | ">=" { GEQ } | ">

(* Function calls. *) | ">" { GT } | ">=" { GEQ } | ">

"do" { DO } | ">" { GT } | ">=" { GEQ } | ">

"to" { TO } | ">" { GT } | ">=" { GEQ } | ">

(* Used for grouping things and creating lists. *) | ">" { GT } | ">=" { GEQ } | ">

(" ws_strip { LPAREN } | ">" { GT } | ">=" { GEQ } | ">

ws_strip ")" { RPAREN } | ">" { GT } | ">=" { GEQ } | ">

["\n"] { LBRACKET } | ">" { GT } | ">=" { GEQ } | ">

["\"] { RBRACKET } | ">" { GT } | ">=" { GEQ } | ">

,

(* Recipe− and stage−related terms. *) | ">" { GT } | ">=" { GEQ } | ">

; 

"=" { COLON } | ">" { GT } | ">=" { GEQ } | ">

"recipe" { RECIPE } | ">" { GT } | ">=" { GEQ } | ">

"done" { DONE } | ">" { GT } | ">=" { GEQ } | ">

"start" { START } | ">" { GT } | ">=" { GEQ } | ">

"next" { NEXT } | ">" { GT } | ">=" { GEQ } | ">

"return" { RETURN } | ">" { GT } | ">=" { GEQ } | ">

(* Other operators used with variables. *) | ">" { GT } | ">=" { GEQ } | ">

"is" { ASSIGN } | ">" { GT } | ">=" { GEQ } | ">

"of" { OF } | ">" { GT } | ">=" { GEQ } | ">

"local" { LOCAL } | ">" { GT } | ">=" { GEQ } | ">

(* I/O *) | ">" { GT } | ">=" { GEQ } | ">

"input" { INPUT } | ">" { GT } | ">=" { GEQ } | ">

(* Identifiers and literals (int, float, bool, string). *) | ">" { GT } | ">=" { GEQ } | ">

"true" { TRUE } | ">" { GT } | ">=" { GEQ } | ">

"false" { FALSE } | ">" { GT } | ">=" { GEQ } | ">

digit+ as lxm { INT(int_of_string lxm) } | ">" { GT } | ">=" { GEQ } | ">

["a'−z' 'A'−'Z'] ["a'−z' 'A'−'Z' '0'−'9' '_']* as lxm { ID(lxm) } | ">" { GT } | ">=" { GEQ } | ">

(digits+.'digits+) | (digit+.'digits+) as lxm { FLOAT(float_of_string lxm) } | ">" { GT } | ">=" { GEQ } | ">

"" { read_string (Buffer.create 17) lexbuf } | ">" { GT } | ">=" { GEQ } | ">

(* Special characters we use to mark end of programs/statements. *) | ">" { GT } | ">=" { GEQ } | ">

| eof { EOF } | ">" { GT } | ">=" { GEQ } | ">

| "\n"+ ws_strip { NEWLINE } (* Empty lines are collapsed. *) | ">" { GT } | ">=" { GEQ } | ">

| _ as char { raise (Failure("illegal character " ^ Char.escape char)) } | ">" { GT } | ">=" { GEQ } | ">

(* Anything else is an illegal character. *) | ">" { GT } | ">=" { GEQ } | ">

| _ as char { raise (Failure("illegal character " ^ Char.escape char)) } | ">" { GT } | ">=" { GEQ } | ">

(* Read in string literals. The code is from | ">" { GT } | ">=" { GEQ } | ">

https://realworldocaml.org/v1/en/html/parsing−with−ocamllex−and−menhir.html *) | ">" { GT } | ">=" { GEQ } | ">

and read_string buf = parse | ">" { GT } | ">=" { GEQ } | ">

| "" "{ STRING(Buffer.contents buf) } | ">" { GT } | ">=" { GEQ } | ">

| "\" "| Buffer.add_char buf "\"; read_string buf lexbuf } | ">" { GT } | ">=" { GEQ } | ">


type op =
| Add | Sub | Mult | Div | Negate |
| Equal | Neq | Lt | Leq | Gt | Geq |
| And | Or | Not |

type scope = Local | Global

type constant =
| Int of int |
| Float of float |
| Bool of bool |
| String of string |

type expr =
| Constant of constant |
| Id of string * scope |
| Unop of op * expr |
| Binop of expr * op * expr |
| Assign of expr * expr |
| Call of string * expr list |
| List of expr list |
| Return of expr |
| Next of string |
| Input |
| Access of expr * expr |

type stmt =
| Expr of expr |
| Block of stmt list |
| If of expr * stmt * stmt |

type stage = {
  sname: string; (* Name of the stage. *)
  body: stmt list; (* The statements that comprise the stage. *)
  is_start: bool; (* Whether the stage is a start stage. *)
}

type recipe = {
  name: string; (* Name of the recipe. *)
  formals: string list; (* Formal argument names. *)
  body: stage list; (* Stages in the recipe’s scope. *)
}
```plaintext
type program = {
    recipes: recipe list;
    stages: stage list;
}

(* Low-level AST printing, to help debug the structure. *)

let op_s = function
    Add -> "Add"
| Sub -> "Sub"
| Mult -> "Mult"
| Div -> "Div"
| Negate -> "Negate"
| Equal -> "Equal"
| Neq -> "Neq"
| Lt -> "Lt"
| Leq -> "Leq"
| Gt -> "Gt"
| Geq -> "Geq"
| And -> "And"
| Or -> "Or"
| Not -> "Not"

let constant_s = function
    Int(i) -> "Int " ^ string_of_int i
| Float(f) -> "Float " ^ string_of_float f
| Bool(b) -> "Bool " ^ string_of_bool b
| String(s) -> "String " ^ s

let rec expr_s = function
    Constant(c) -> constant_s c
| Id(str, scope) -> "Id " ^
        (match scope with Local -> "Local "
           | Global -> "Global ") ^
    str
| Unop(o, e) -> "Unop " ^ (op_s o) ^ " (" ^ expr_s e ^ ")"
| Binop(e1, o, e2) -> "Binop (" ^ expr_s e1 ^ ") " ^
        (op_s o) ^
        " (" ^ expr_s e2 ^ ")"
| Assign(v, e) -> "Assign (" ^ expr_s v ^ ") (" ^ expr_s e ^ ")"
| Call(f, es) -> "Call " ^ f ^ " [" ^
            String.concat ", " (List.map
                (fun e -> "(" ^ expr_s e ^ ")")
            es) ^
            "]"
| List(es) -> "List [" ^
            String.concat ", " (List.map
                (fun e -> "(" ^ expr_s e ^ ")")
            es) ^
            "]"
| Return(e) -> "Return (" ^ expr_s e ^ ")"
| Next(s) -> "Next " ^ s
| Input -> "input"
| Access(i, l) -> "Access " ^ (expr_s i) ^ " of " ^ (expr_s l)

let rec stmt_s = function
    Expr(e) -> "Expr (" ^ expr_s e ^ ")"
| Block(ss) -> "Block [" ^
```

let stage_s s = 
"{ s.name = "" ^ s.sname ^ "\n" ^
  " is_start = " ^ string_of_bool s.is_start ^ "\n" ^
  " body = [" ^ String.concat ",\n" (List.map stmt_s s.body) ^
  "}"\n"

let recipe_s r = 
"{ r.name = "" ^ r.rname ^ "\n" ^
  " formals = [" ^ String.concat ", " r.formals ^ "]\n" ^
  " body = [" ^ String.concat ",\n" (List.map stage_s r.body) ^
  "}"\n"

let program_s prog = 
"recipes = [" ^ String.concat ",\n" (List.map recipe_s prog.recipes) ^
  "]\n" ^
"stages = [" ^ String.concat ",\n" (List.map stage_s prog.stages) ^ ""]

Listing B.3: ast.ml

B.4 SAST

(* The basic types used in annotation *)
type t =
  TInt
| TFloat
| TBool
| TString
| TList
| TOCamlString
| TUnknown

type a_constant =
  AInt of int * t
| AFloat of float * t
| ABool of bool * t
| AString of string * t

type a_expr =
  AConstant of a_constant
| AId of string * Ast.scope * t
| AUnop of Ast.op * a_expr * t
| ABinop of a_expr * Ast.op * a_expr * t
| AAssign of a_expr * a_expr
| ANext of string * t
| AReturn of a_expr * t
| AList of a_expr list * t
| AInput of t
| ACall of string * a_expr list * t
| AAccess of a_expr * a_expr * t

type a_stmt =
  AExpr of a_expr
| ABlock of a_stmt list
### B.5 Semantic Analyzer

```ml
open Ast
open Sast

module StringMap = Map.Make(String);;
module StringSet = Set.Make(String);;

let lib_funcs = ["show", 1]; ("remove", 2); ("insert", 3); ("append", 2); ("length", 1); ("word_to_number", 1); ("number_to_word", 1)];;

(* A symbol table which includes a parent symbol table and variables which are tuples of strings and Sast types *)
type symbol_table = {
    mutable variables : (string * Sast.t) list;
} type environment = {
    global_scope : symbol_table;
    local_scope : symbol_table;
} type_of_constraint (ac : Sast.a_constant) : Sast.t =
match ac with
    AInt(_, t) -> t
| AFloat(_, t) -> t
| ABool(_, t) -> t
| AString(_, t) -> t

let rec type_of_expr (ae : Sast.a_expr) : Sast.t =
match ae with
    AConstant(const) -> type_of_constraint const
```

Listing B.4: sast.ml
let find_variable_type (env : environment) (id : Ast.expr) : 
  Sast.t option =
try
  let (_, typ) = match id with
  | Id(name, Local) -> List.find
    (fun (s, _) -> s = name)
    env.local_scope.variables
  | Id(name, Global) -> List.find
    (fun (s, _) -> s = name)
    env.global_scope.variables
  | _ -> failwith "Error using find_variable_type"
  in
  Some typ
with Not_found -> match id with
  | Id(_, Global) -> Some(TUnknown)
  | _ -> None

(* Check to see if param is important or not *)
let mutate_or_add (env : environment) (id : Ast.expr) (new_type : Sast.t) =
  let typ = find_variable_type env id in
  let name, scope = match id with
  | Id(i, Local) -> i, env.local_scope
  | Id(i, Global) -> i, env.global_scope
  | _ -> failwith "Error using mutate_or_add"
  in
  match typ with
  | Some(t) ->
    (* filter name, t out of symbol_table.variables *)
    scope.variables <-
      (name, new_type) :: (List.filter (fun (s, _) -> s <> name)
        scope.variables)
  | None ->
    scope.variables <- (name, new_type) :: scope.variables

let annotate_const (c : Ast.constant) : Sast.a_expr =
  match c with
  | Int(n) -> AConstant(AInt(n, TInt))
  | Float(f) -> AConstant(AFloat(f, TFloat))
  | Bool(b) -> AConstant(ABool(b, TBool))
  | String(s) -> AConstant(AString(s, TString))

let rec annotate_expr (e : Ast.expr) (env : environment) : Sast.a_expr =
  match e with
  | Constant(c) -> annotate_const c
  | Id(i, s) ->
(match find_variable_type env e with
  | Some(x) -> Ald(i, s, x)
  | None -> failwith ("unrecognized identifier " ^ i ^ ".")
| Unop(op, e1) ->
  let ae1 = annotate_expr e1 env in
  AUnop(op, ae1, type_of ae1)
| Binop(e1, op, e2) ->
  let ae1 = annotate_expr e1 env
  and ae2 = annotate_expr e2 env in
  ABinop(ae1, op, ae2, TUnknown)
| Assign(e1, e2) ->
  (match e1 with
    | Id(str, scope) ->
      let ae2 = annotate_expr e2 env in
      mutate_or_add env e1 (type_of ae2);
      let ae1 = annotate_expr e1 env in
      AAssign(ae1, ae2)
    | Access(e, id) ->
      let ae2 = annotate_expr e2 env in
      let ae1 = annotate_expr e1 env in
      (match find_variable_type env id with
        | Some(TList) -> AAssign(ae1, ae2)
        | _ -> failwith "Variable not found"
      )
    | _ -> failwith "Invalid assignment operation"
  | Next(s) -> ANext(s, TOCamlString)
  | Return(e) -> let ae = annotate_expr e env in
  | Return(e) -> AReturn(ae, type_of ae)
  | List(e_list) -> let ae_list = List.map
    (fun e -> annotate_expr e env)
    e_list in
  | List(e_list) -> AList(ae_list, TList)
  | Input -> AInput(TString)
  | Call(s, e_list) -> let ae_list = List.map
    (fun e -> annotate_expr e env)
    e_list in
  | Call(s, e_list) -> ACall(ae_list, TUnknown)
  | Access(e, id) ->
      let l = find_variable_type env id in
      let ind_expr = annotate_expr e env in
      match l with
      | Some(TList) ->
        AAccess(ind_expr,
        (annotate_expr id env),
        TUnknown)
      | _ -> failwith "Bad list access"
  | _ -> failwith "Bad list access"

let rec annotate_stmt (s : Ast.stmt) (env : environment) : Sast.a_stmt =
  match s with
  | Expr(e) -> AExpr(annotate_expr e env)
  | Block(s_list) -> ABlock(List.map (fun s -> annotate_stmt s env) s_list)
  | If(e, s1, s2) -> let ae = annotate_expr e env in
    AIf(ae,
    annotate_stmt s1 env,
    annotate_stmt s2 env)

let annotate_stage (s : Ast.stage) (env : environment) : Sast.a_stage =
  let new_env = (global_scope = env.global_scope;
  local_scope = { variables = []; }); in
  { sname = s.sname;
  body = List.map (fun stmt -> annotate_stmt stmt new_env) s.body;
  is_start = s.is_start; }
let annotate_recipe (r : Ast.recipe) : Sast.a_recipe = 
let new_env = { global_scope = {
    variables = List.map (fun s -> (s, TUnknown)) r.formals;
};
local_scope = { variables = []; } in 
{ rname = r.rname;
formals = r.formals;
body = List.map (fun stage -> annotate_stage stage new_env) r.body; }

let annotate_program (p : Ast.program) : Sast.a_program = 
let new_env = { global_scope = { variables = []; };
local_scope = { variables = []; } in 
{ recipes = List.map annotate_recipe p.recipes;
stages = List.map (fun stage -> annotate_stage stage new_env) p.stages; }

let rec collect_outs (s : Sast.a_stage) : string list = 
List.fold_left collect_nexts_stmt [] s.body 
and collect_nexts_stmt (l : string list) (s : Sast.a_stmt) : string list = 
match s with 
  AExpr(ae) -> collect_nexts_expr l ae 
| ABlock(s_l) -> List.fold_left collect_nexts_stmt l s_l 
| Alf(_, s1, s2) -> collect_nexts_stmt (collect_nexts_stmt l s1) s2 
and collect_nexts_expr (l : string list) (e : Sast.a_expr) : string list = 
match e with 
  ANext(s, _) -> if List.exists (fun name -> name = s) l 
    then l 
    else s :: l 
| _ -> l

(* Returns a set of the names of reachable stages and a list of errors with the 
names of invalid stages attempted to visit. *)
let rec visit_stages (queue : string list) 
  (visited : StringSet.t)
  (stages)
  (errors : string list) : StringSet.t * string list = 
if List.length queue = 0 
then visited, errors 
else let current = List.hd queue in 
let nexts = StringMap.find current stages in 
visit_stages 
((List.tl queue) @ 
(List.filter 
  (fun name -> not(StringMap.mem name queue) &&
   not(StringSet.mem name visited) &&
   StringMap.mem name stages)
nexts)) 
(StringSet.add current visited) 
stages 
(Errors @ 
List.map 
  (fun inval -> "Error in stage " ^ current ^ " : next " ^
   inval ^ " calls an invalid stage.")
(List.filter 
  (fun name -> not(StringMap.mem name stages))
nexts))
let generate_stage_flow_diagnostics (stages : Sast.a_stage list) :
  string list
  string list =
  let start = List.find (fun s -> s.is_start) stages in
  let visited, errors =
    visit_stages
    [start.sname]
    StringSet.empty
    (List.fold_left (fun map stage ->
      StringMap.add stage.sname (collect_outs stage) map)
      StringMap.empty
      stages)
    []
  and stage_set = StringSet.of_list (List.map (fun s -> s.sname) stages) in
  let unreachable = StringSet.diff stage_set visited in
  StringSet.fold
  (fun name warnings --->
    ("Warning: stage " ^ name ^ " is unreachable.") :: warnings)
  unreachable
  []), errors

let dup_string_check (names : string list) : string list =
  StringMap.fold
  (fun name count dups --->
    if count > 1
    then name :: dups
    else dups)
  (List.fold_left
    (fun map name --->
      if StringMap.mem name map
      then StringMap.add name ((StringMap.find name map) + 1) map
      else StringMap.add name 1 map)
    StringMap.empty
    names)
  []

let generate_stage_diagnostics (stages : Sast.a_stage list) :
  string list
  string list =
  let snames = List.map (fun s -> s.sname) stages in
  let dup_name_errors =
    List.map
    (fun name ---> "Error: multiple stages named " ^ name ^ ".")
    (dup_string_check snames)
  and num_starts = List.length (List.filter (fun s -> s.is_start) stages) in
  let errors =
    if num_starts > 1
    then ["Error: more than one stage is marked start."] @ dup_name_errors
    else if num_starts < 1
then ["Error: no stages marked start."] @ dup_name_errors
else dup_name_errors in
if List.length errors > 0
then [], errors
else generate_stage_flow_diagnostics stages

(* Check if multiple recipes have the same name. Returns a list of errors. *)
let generate_recipe_diagnostics (recipes : Sast.a_recipe list) =
let rnames = List.map (fun r -> r.rname) recipes in
List.map
  (fun name -> "Error: multiple recipes named " ^ name ^ ".")
(dup_string_check rnames)

let rec collect_calls (s : Sast.a_stage) : (string * int) list =
  List.fold_left collect_calls_stmt [] s.body
and collect_calls_stmt (l : (string * int) list) (s : Sast.a_stmt) :
  (string * int) list =
match s with
  AExpr(ae) -> collect_calls_expr l ae
| ABlock(s_l) -> List.fold_left collect_calls_stmt l s_l
| ALet(_, s1, s2) -> collect_calls_stmt (collect_calls_stmt l s1) s2
and collect_calls_expr (l : (string * int) list) (e : Sast.a_expr) :
  (string * int) list =
match e with
  ACall(name, formals, _) -> (name, List.length formals) :: l
| _ -> l

(* Check if all recipe calls are calls to library functions or user-defined functions. Also checks if the number of arguments is correct. Args:
  recipes: a list of recipes, assumed to have unique names
  stages: a list of stages *)
let generate_call_diagnostics (recipes : Sast.a_recipe list)
  (stages : Sast.a_stage list) : string list =
let rformals = List.fold_left
  (fun l r -> (r.rname, List.length r.formals) :: l)
[[]
recipes @ lib funcs in
List.fold_left
  (fun list stage ->
    (List.fold_left
      (fun l name_formals ->
        let name = fst name_formals in
        let count = snd name_formals in
        if not(List.mem_assoc name rformals)
        then ("Error in stage " ^ stage.sname ^ ": call to " ^ name ^ 
          " does not refer to a defined recipe."") :: l
        else let ecount = List.assoc name rformals in
        if ecount != count
        then ("Error in stage " ^ stage.sname ^ ": call to " ^ name ^ 
          " expects " ^ (string_of_int ecount) ^ " arguments but " ^
          (string_of_int count) ^ " provided."") :: l
        else l)
    []
  )
(collect_calls stage) @ list)
[]
stages

(* Returns a list of diagnostics (warnings and errors) and whether any of the
diagnostics are fatal errors. *)

let generate_diagnostics (p : Sast.a_program) : string list * bool =
  let r_format name str = "In recipe " ^ name ^ ": " ^ str in
  let r_internal_call_errors =
    List.concat (List.map
      (fun r -> List.map
        (fun str -> r_format r.name str)
        (generate_call_diagnostics p.recipes r.body))
      p.recipes)

  and r_internal_diagnostics, has_r_internal_errors =
    List.fold_left
      (fun pair r ->
         let r_internal_s_warnings, r_internal_s_errors =
           generate_stage_diagnostics r.body in
         ((fst pair) @
          (List.map
            (fun str -> r_format r.name str)
            r_internal_s_warnings) @
          (List.map
            (fun str -> r_format r.name str)
            r_internal_s_errors)),
          snd pair || List.length r_internal_s_errors > 0)
      p.recipes

  and r_errors = generate_recipe_diagnostics p.recipes

  and s_warnings, s_errors = generate_stage_diagnostics p.stages

  and c_errors = generate_call_diagnostics p.recipes p.stages in

  let all_diagnostics =
    (r_errors @ s_warnings @ s_errors @
     r_internal_call_errors @ r_internal_diagnostics) in

  all_diagnostics, (has_r_internal_errors ||
    List.length all_diagnostics = List.length s_warnings > 0)

Listing B.5: analyzer.ml

B.6 Code Generator

open Ast
open Printf
open Sast

let global_scope = Hashtbl.create 1000;;
let local_scope = Hashtbl.create 1000;;

let get_initial_stage_header (start_stage_name : string)
  (is_recipe : bool)
  (formals : string list) =

  if is_recipe
  then let initial = "\n \tpublic SNLObject perform(" in
    let list_of_args = List.map
      (fun name -> "SNLObject " ^ name ^ ": _arg")
      formals in

    let perform_args = (String.concat ", " list_of_args) ^ ")\n\n" in

    let args_in_body = List.map
      (fun name -> name ^ " = new SNLObject(" ^
        name ^ ": _arg);\n\n") formals in

open !ST
open ORINTF
open 3AST
```
let constructs = (String.concat "" args_in_body) in
initial ^ perform_args ^ constructs ^"s_" ^ start_stage_name ^
"()
\return ret;
\n"
else "\n public static void main(String args[])\ns_" ^ start_stage_name ^
"()
\n"

let to_string_const (const : a_constant) : string =
match const with
  | Int(num, _) -> " new SNLObject(" ^ (string_of_int num) ^ ")"
  | AFloat(fl, _) -> " new SNLObject(" ^ (string_of_float fl) ^ ")"
  | ABool(b, _) -> " new SNLObject(" ^ (string_of_bool b) ^ ")"
  | AString(s, _) -> " new SNLObject(" ^ s ^ ")"

let to_string_id (name : string) (scope : Ast.scope) : string =
match scope with
  | Local -> (match Hashtbl.mem local_scope name with
           | true -> name
           | false -> Hashtbl.add local_scope name name;
                        "SNLObject " ^ name)
  | Global -> (match Hashtbl.mem global_scope name with
              | true -> name
              | false -> Hashtbl.add global_scope name name; name)

let rec to_string_expr (expr : a_expr) : string =
match expr with
  | AConstant(const) -> to_string_const const
  | Ald(name, scope, _) -> to_string_id name scope
  | AUnop(op, e, _) -> to_string_unop e op
  | ABinop(e1, op, e2, _) -> to_string_binop e1 e2 op
  | AAssign(e1, e2) -> to_string_expr e1 ^ "=" ^ to_string_expr e2
  | ANext(s, _) -> "s_" ^ s ^ ")\nreturn"
  | AReturn(e, _) -> "ret = " ^ (to_string_expr e) ^ ";\n" ^ ";return"
  | AList(e_list, _) -> to_string_list e_list
  | AInput(t) -> "new SNLObject(input.nextLine())"
  | ACall(s, e_list, _) -> to_string_call s e_list
  | AAccess(index_e, e, _) -> (to_string_expr e) ^
              ".getArr()[" ^
              (to_string_expr index_e) ^
              ".getInt()"

and to_string_unop (e : a_expr) (op : Ast.op) : string =
let string_op =
match op with
  | Negate -> "neg"
  | _ -> "Error" in
(to_string_expr e) ^ "." ^ string_op ^ "()

and to_string_binop (e1 : a_expr) (e2 : a_expr) (op : Ast.op) =
let string_op =
match op with
  | Add -> "add"
  | Sub -> "sub"
  | Mult -> "mul"
  | Div -> "div"
```
match name with
  "show" -> let list_e_strings = List.rev (List.fold_left (fun list e -> (to_string_expr e) :: list) [] e_list) in
    "System.out.println(" ^ (String.concat " + " list_e_strings) ^")"
  | "remove" -> let lst = to_string_expr (List.nth e_list 0) in
    let index = to_string_expr (List.nth e_list 1) in
    lst ^ ".remove(" ^ index ^ ")"
  | "insert" -> let lst = to_string_expr (List.nth e_list 0) in
    let item_to_add = to_string_expr (List.nth e_list 1) in
    let index = to_string_expr (List.nth e_list 2) in
    lst ^ ".insert(" ^ index ^ ", " ^ item_to_add ^ ")"
  | "append" -> let lst = to_string_expr (List.nth e_list 0) in
    let item_to_add = to_string_expr (List.nth e_list 1) in
    lst ^ ".append(" ^ item_to_add ^ ")"
  | "length" -> let lst = to_string_expr (List.nth e_list 0) in
    lst ^ ".length()"
  | "word_to_number" -> let word = to_string_expr (List.nth e_list 0) in
    word ^ ".word_to_number()"
  | "number_to_word" -> let word = to_string_expr (List.nth e_list 0) in
    word ^ ".number_to_word()"
  | _ -> let list_e_strings = List.rev (List.fold_left (fun list e -> (to_string_expr e) :: list) [] e_list) in
    "new Recipe." ^ name ^ "().perform(" ^ (String.concat ", " list_e_strings) ^ ")"

and to_string_list (e_list : a_expr list) : string =
let list_e_strings = List.rev (List.fold_left (fun list e -> (to_string_expr e) :: list) [] e_list) in
  "new SNLObject(" ^ (String.concat ", " list_e_strings) ^ ")"

let rec to_string_stmt (statement : a_stmt) =
match statement with
| AExpr(e) -> (to_string_expr e) ^ ";\n"
| ABlock(s_list) ->
  let list_of_strings = List.rev (List.fold_left (fun list e ->
    ...
(to_string_stmt e) :: list

[]
s_list) in

String.concat "" list_of_strings
| Alf(e, first, second) -> let expr_str = (to_string_expr e) in
let first_str = to_string_stmt first in
let second_str = to_string_stmt second in
"if(" ^ expr_str ^ ".getBool()" ^ first_str ^ ")" ^ "else(" ^ second_str ^ ")" ^ "n"

let to_string_stage (stage : a_stage)
(is_recipe : bool)
(formals : string list) : string =
Hashtbl.clear local_scope;
let header =
if is_recipe then "private void s_" ^ stage.sname ^ "()\n"
else "private static void s_" ^ stage.sname ^ "()\n"
in
let initial_header =
if stage.is_start
then get_initial_stage_header stage.sname is_recipe formals
else ""

let list_of_strings = List.rev (List.fold_left
(fun list s ->
(to_string_stmt s) :: list)
[]
stage.body) in
initial_header ^ header ^ (String.concat "\n" list_of_strings) ^ ")"

let to_string_stages (stages : a_stage list)
(is_recipe : bool)
(formals : string list) =
let list_of_strings = List.rev
(List.fold_left
(fun list s ->
(to_string_stage s is_recipe formals) :: list)
[]
stages) in

let global_vars =
if is_recipe then Hashtbl.fold (fun k v acc ->
"private SNLObject 
^ k ^ ";\n" ^ acc) global_scope "
else Hashtbl.fold (fun k v acc ->
"private static SNLObject 
^ k ^ ";\n" ^ acc) global_scope "
in
(String.concat "" list_of_strings) ^ global_vars ^ ")"

(* name should be the file name of the snl file or recipe
without any extensions. *)
let make_header (name : string) (is_recipe : bool) : string =
let scanner = "import java.util.Scanner;\n" in
if is_recipe
then let scanner2 = "\private Scanner input = new Scanner(System.in);" in
  scanner ^ "public class " ^ "Recipe_" ^ name ^ "{\n" ^ 
    "\private SNLObject ret;\n" ^ scanner2 ^ "\npublic Recipe_" ^ name ^ 
  "();\n" ^ "n"

44
else let scanner2 = "\tpublic static Scanner input = " ^
          "new Scanner(System.in);" in
      scanner ^ "public class " ^ name ^ "\n" ^ scanner2

let gen_main (stages : a_stage list) (name : string) : string =
  make_header name false ^ to_string_stages stages false []

let gen_recipe (recipe : a_recipe) : string =
  Hashtbl.clear global_scope;
  List.iter (fun formal -> Hashtbl.add global_scope formal formal)
      recipe.formals;
  make_header recipe.name true ^
      to_string_stages recipe.body true recipe.formals

Listing B.6: codegen.ml

B.7 Compiler

(* Usage: ./snlc [-e | -s | -p | -j] file [-o output_file] *)

open Analyzer
open Printf
open Sast

type action = Expr | Stmt | Program | Java

let write_out (filename : string) (buffer : string) =
  if Sys.file_exists filename then Sys.remove(filename);
  let file = (open_out_gen
         [Open_creat; Open_wonly; Open_text]
         00666
         filename) in
  fprintf file "%s" buffer;
  close_out file

let _ =
  let action = List.assoc Sys.argv.(1) [("-e", Expr);
            ("-s", Stmt);
            ("-p", Program);
            ("-j", Java)] in

  let lexbuf = Lexing.from_channel (open_in Sys.argv.(2)) in
  match action with
    (* expr, stmt, and program are for testing the AST, java is code gen *)
    Expr -> print_string (Ast.expr_s (Parser.expr Scanner.tokenize lexbuf))
  | Stmt -> print_string (Ast.stmt_s (Parser.stmt Scanner.tokenize lexbuf))
  | Program -> print_string (Ast.program_s
          (Parser.program Scanner.tokenize lexbuf))
  | Java ->
    (* see if file exists and remove if it is already there *)
    let strlst = Str.split (Str.regexp "/") Sys.argv.(2) in
    let tail = List.hd (List.rev strlst) in
    let name = String.sub tail 0 ((String.length tail) - 4)
    and path = if Array.length Sys.argv > 3 && Sys.argv.(3) = "--output_path"
       then Sys.argv.(4) ^ "/"
```plaintext
else "./" in
let ast = Parser.program Scanner.tokenize lexbuf in
let sast = Analyzer.annotate_program ast in
let diagnostics, any_error = Analyzer.generate_diagnostics sast in
List.iter print_endline diagnostics;
if any_error then failwith "Errors in program."
else write_out (path ^ name ^ ".java") (CodeGen.gen_main sast.stages name);
ignore (List.map
  (fun recipe -> write_out
     (path ^ "Recipe_" ^ recipe.name ^ ".java")
     (CodeGen.gen_recipe recipe))
sast.recipes)
```

Listing B.7: snlc.ml

### B.8 SNL Script

```plaintext
#!/bin/bash
./snlc -j $1 > /dev/null
x=$1
y=${x%%.snl}
javac "$y.java"
echo "To run program, enter: java $y"
```

Listing B.8: snl

### B.9 Java Backend

```plaintext
public class SNLObject {
  // used for comparison in typeCheck
  private enum Type {
    INT, FLOAT, BOOL, STRING, LIST;
  }

  // all of the different meta data available for Object wrapper
  private Type type;
  private int valueInt;
  private double valueFloat;
  private boolean valueBool;
  private String valueString;
  private SNLObject[] valueList;

  // constructor for int object
  public SNLObject(int vInt) {
    type = Type.INT;
    valueInt = vInt;
  }

  // constructor for float object
  public SNLObject(double vFloat) {
    type = Type.FLOAT;
    valueFloat = vFloat;
  }

  // constructor for bool object
  public SNLObject(boolean vBool) {
    type = Type.BOOL;
  }
```
```java
            valueBool = vBool;

        }

        // constructor for string object
        public SNLObject(String vString) {
            type = Type.STRING;
            stringValue = vString;
        }

        // constructor for list object
        // is moved because of Java requirements
        public SNLObject(SNLObject ... objects) {
            type = Type.LIST;
            valueList = new SNLObject[objects.length];
            for (int i = 0; i < objects.length; i++)
                valueList[i] = objects[i];
        }

        // copy constructor
        public SNLObject(SNLObject old) {
            type = old.type;
            switch (type) {
                case INT:
                    valueInt = old.getInt();
                    break;
                case FLOAT:
                    valueFloat = old.getFloat();
                    break;
                case BOOL:
                    valueBool = old.getBool();
                    break;
                case STRING:
                    stringValue = old.getString();
                    break;
                case LIST:
                    valueList = new SNLObject[old.getArr().length];
                    for (int i = 0; i < old.getArr().length; i++)
                        valueList[i] = old.getArr()[i];
                    break;
            }
        }

        // Getter methods for private data.
        private double getFloat() {
            return valueFloat;
        }

        private String getString() {
            return stringValue;
        }

        // These three are the only public ones
        // because of if statements and access.
        public boolean getBool() {
            return valueBool;
        }

        public SNLObject[] getArr() {
            return valueList;
        }
```
public int getInt() {
    return valueInt;
}

// goes from a string to a number
public SNLObject word_to_number() {
    return new SNLObject(Integer.parseInt(getString()));
}

// goes from a number to a string
public SNLObject number_to_word() {
    SNLObject ret = null;
    if (type == Type.INT)
        ret = new SNLObject(String.valueOf((int)getInt()));
    else if (type == Type.FLOAT)
        ret = new SNLObject(String.valueOf(getFloat()));
    return ret;
}

// helper method to check types
private static boolean typeMatch(SNLObject subject, SNLObject desired) {
    return subject.type == desired.type;
}

// this is the '+' operator
public SNLObject add(SNLObject right) {
    SNLObject snlo = null;
    if (typeMatch(this, right)) {
        // add two ints
        if (type == Type.INT)
            snlo = new SNLObject(this.getInt() + right.getInt());
        else if (type == Type.FLOAT)
            snlo = new SNLObject(this.getFloat() + right.getFloat());
        else if (type == Type.STRING)
            snlo = new SNLObject(this.getString() + right.getString());
    } else if (type == Type.FLOAT && right.type == Type.INT)
        snlo = new SNLObject(this.getFloat() + right.getInt());
    else if (type == Type.INT && right.type == Type.FLOAT)
        snlo = new SNLObject(this.getInt() + right.getFloat());
    return snlo;
}

// this is the '-' binary operator
public SNLObject sub(SNLObject right) {
    SNLObject snlo = null;
    if (typeMatch(this, right)) {
        // sub two ints
        if (type == Type.INT)
            snlo = new SNLObject(this.getInt() - right.getInt());
        else if (type == Type.FLOAT)
            snlo = new SNLObject((float)getInt() - (float)right.getInt());
    } else if (type == Type.FLOAT && right.type == Type.INT)
        snlo = new SNLObject((float)getInt() - right.getInt());
    return snlo;
}
SNLO = new SNLObject(this.getFloat() - right.getFloat());
}
// can also sub float and int
else if (type == Type.FLOAT && right.type == Type.INT)
    snlo = new SNLObject(this.getFloat() - right.getInt());
// can also sub int and float
else if (type == Type.INT && right.type == Type.FLOAT)
    snlo = new SNLObject(this.getInt() - right.getFloat());
// return is null if something went wrong at runtime
return snlo;

// this is the ‘*’ operator
public SNLObject mult(SNLObject right) {
    SNLObject snlo = null;
    // if types match
    if (typeMatch(this, right)) {
        // mult two ints
        if (type == Type.INT)
            snlo = new SNLObject(this.getInt() * right.getInt());
        // mult two floats
        if (type == Type.FLOAT)
            snlo = new SNLObject(this.getFloat() * right.getFloat());
    }
    // can also mult float and int
    else if (type == Type.FLOAT && right.type == Type.INT)
        snlo = new SNLObject(this.getFloat() * right.getInt());
    // can also mult int and float
    else if (type == Type.INT && right.type == Type.FLOAT)
        snlo = new SNLObject(this.getInt() * right.getFloat());
    // return is null if something went wrong at runtime
    return snlo;
}

// this is the ‘/’ operator
// errors like divide by zero caught at runtime
public SNLObject div(SNLObject right) {
    SNLObject snlo = null;
    // if types match
    if (typeMatch(this, right)) {
        // mult two ints
        if (type == Type.INT)
            snlo = new SNLObject(this.getInt() / right.getInt());
        // mult two floats
        if (type == Type.FLOAT)
            snlo = new SNLObject(this.getFloat() / right.getFloat());
    }
    // can also mult float and int
    else if (type == Type.FLOAT && right.type == Type.INT)
        snlo = new SNLObject(this.getFloat() / right.getInt());
    // can also mult int and float
    else if (type == Type.INT && right.type == Type.FLOAT)
        snlo = new SNLObject(this.getInt() / right.getFloat());
    // return is null if something went wrong at runtime
    return snlo;
}

// this is the ‘−’ unary operator
public SNLObject neg() {
    SNLObject snlo = null;
// can neg int
if (type == Type.INT)
    snlo = new SNLObjec(getInt() * (-1));
// can neg float
else if (type == Type.FLOAT)
    snlo = new SNLObjec(getFloat() * (-1));
// return is null if something went wrong at runtime
return snlo;

// this is the '=' binary operator
public SNLObjec eq(SNLObjec right) {
    SNLObjec snlo = null;
    // if types match
    if (typeMatch(this, right)) {
        // eq two ints
        if (type == Type.INT)
            snlo = new SNLObjec(this.getInt() == right.getInt());
        // eq two floats
        else if (type == Type.FLOAT)
            snlo = new SNLObjec(this.getFloat() == right.getFloat());
        // eq two bools
        else if (type == Type.BOOL)
            snlo = new SNLObjec(this.getBool() == right.getBool());
        // eq two strings
        else if (type == Type.STRING)
            snlo = new SNLObjec(this.getString().equals(right.getString()));
    } else {
        // eq for a float and an int
        // 4.0 and 4 evaluate to the same
        if (type == Type.FLOAT && right.type == Type.INT) {
            Integer tmp = new Integer(right.getInt());
            snlo = new SNLObjec(this.getFloat() == tmp.floatValue());
        } else if (type == Type.INT && right.type == Type.FLOAT) {
            Integer tmp = new Integer(getInt());
            snlo = new SNLObjec(tmp.floatValue() == right.getFloat());
        }
    }
    // return is null if something went wrong at runtime
    return snlo;
}

// this is the '!=' binary operator
public SNLObjec neq(SNLObjec right) {
    SNLObjec snlo = null;
    // if types match
    if (typeMatch(this, right)) {
        // neq two ints
        if (type == Type.INT)
            snlo = new SNLObjec(this.getInt() != right.getInt());
        // neq two floats
        else if (type == Type.FLOAT)
            snlo = new SNLObjec(this.getFloat() != right.getFloat());
        // neq two bools
        else if (type == Type.BOOL)
            snlo = new SNLObjec(this.getBool() != right.getBool());
        // neq two strings
        else if (type == Type.STRING)
snlo = new SNLObject(
    !this.getString().equals(right.getString()));
}
else {
    // neq for a float and an int
    // 4.0 and 4 evaluate to the same
    if (type == Type.FLOAT && right.type == Type.INT) {
        Integer tmp = new Integer(right.getInt());
        snlo = new SNLObject(this.getFloat() != tmp.floatValue());
    }
    if (type == Type.INT && right.type == Type.FLOAT) {
        Integer tmp = new Integer(getInt());
        snlo = new SNLObject(tmp.floatValue() != right.getFloat());
    }
    }
    // return is null if something went wrong at runtime
    return snlo;
}

// this is the '<' binary operator
public SNLObject lt(SNLObject right) {
    SNLObject snlo = null;
    // if types match
    if (typeMatch(this, right)) {
        // lt two ints
        if (type == Type.INT)
            snlo = new SNLObject(this.getInt() < right.getInt());
        // lt two floats
        if (type == Type.FLOAT)
            snlo = new SNLObject(this.getFloat() < right.getFloat());
    } else {
        // lt for a float and an int
        if (type == Type.FLOAT && right.type == Type.INT) {
            Integer tmp = new Integer(right.getInt());
            snlo = new SNLObject(this.getFloat() < tmp.floatValue());
        }
        if (type == Type.INT && right.type == Type.FLOAT) {
            Integer tmp = new Integer(getInt());
            snlo = new SNLObject(tmp.floatValue() < right.getFloat());
        }
    }
    // return is null if something went wrong at runtime
    return snlo;
}

// this is the '<=' binary operator
public SNLObject leq(SNLObject right) {
    SNLObject snlo = null;
    // if types match
    if (typeMatch(this, right)) {
        // leq two ints
        if (type == Type.INT)
            snlo = new SNLObject(this.getInt() <= right.getInt());
        // leq two floats
        if (type == Type.FLOAT)
            snlo = new SNLObject(this.getFloat() <= right.getFloat());
    } else {
        // leq for a float and an int
        if (type == Type.FLOAT && right.type == Type.INT) {
            Integer tmp = new Integer(right.getInt());
            snlo = new SNLObject(this.getFloat() <= tmp.floatValue());
        }
        if (type == Type.INT && right.type == Type.FLOAT) {
            Integer tmp = new Integer(getInt());
            snlo = new SNLObject(tmp.floatValue() <= right.getFloat());
        }
    }
    // return is null if something went wrong at runtime
    return snlo;
}
if (type == Type.INT && right.type == Type.FLOAT) {
    Integer tmp = new Integer(getInt());
    snlo = new SNLObject(tmp.floatValue() <= right.getFloat());
}

// return is null if something went wrong at runtime
return snlo;

// this is the '>' binary operator
public SNLObject gt(SNLObject right) {
    SNLObject snlo = null;
    if (typeMatch(this, right)) {
        // gt two ints
        if (type == Type.INT)
            snlo = new SNLObject(this.getInt() > right.getInt());
        // gt two floats
        if (type == Type.FLOAT)
            snlo = new SNLObject(this.getFloat() > right.getFloat());
    } else {
        // gt for a float and an int
        if (type == Type.FLOAT && right.type == Type.INT) {
            Integer tmp = new Integer(right.getInt());
            snlo = new SNLObject(this.getFloat() > tmp.floatValue());
        } else {
            // gt for a float and an int
            if (type == Type.FLOAT && right.type == Type.INT) {
                Integer tmp = new Integer(getInt());
                snlo = new SNLObject(tmp.floatValue() > right.getFloat());
            }
        }
    }
    // return is null if something went wrong at runtime
    return snlo;
}

// this is the '>=' binary operator
public SNLObject geq(SNLObject right) {
    SNLObject snlo = null;
    if (typeMatch(this, right)) {
        // geq two ints
        if (type == Type.INT)
            snlo = new SNLObject(this.getInt() >= right.getInt());
        // geq two floats
        if (type == Type.FLOAT)
            snlo = new SNLObject(this.getFloat() >= right.getFloat());
    } else {
        // geq for a float and an int
        if (type == Type.FLOAT && right.type == Type.INT) {
            Integer tmp = new Integer(right.getInt());
            snlo = new SNLObject(this.getFloat() >= tmp.floatValue());
        } else {
            // geq for a float and an int
            if (type == Type.FLOAT && right.type == Type.FLOAT) {
                Integer tmp = new Integer(getInt());
                snlo = new SNLObject(tmp.floatValue() >= right.getFloat());
            }
        }
    }
    // return is null if something went wrong at runtime
    return snlo;
return snlo;
}

// this is the 'and' binary operator
public SNLObject and(SNLObject right) {
    SNLObject snlo = null;
    // if types match
    if (typeMatch(this, right)) {
        // and two bools
        if (type == Type.BOOL)
            snlo = new SNLObject(this.getBool() && right.getBool());
    }
    // return is null if something went wrong at runtime
    return snlo;
}

// this is the 'or' binary operator
public SNLObject or(SNLObject right) {
    SNLObject snlo = null;
    // if types match
    if (typeMatch(this, right)) {
        // or two bools
        if (type == Type.BOOL)
            snlo = new SNLObject(this.getBool() || right.getBool());
    }
    // return is null if something went wrong at runtime
    return snlo;
}

// this is the 'not' unary operator
public SNLObject not() {
    SNLObject snlo = null;
    if (type == Type.BOOL)
        snlo = new SNLObject(!getBool());
    // return is null if something went wrong at runtime
    return snlo;
}

// this is to append an element to the list
public void app(SNLObject obj) {
    SNLObject[] tmp = new SNLObject[valueList.length + 1];
    System.arraycopy(valueList, 0, tmp, 0, valueList.length);
    tmp[tmp.length - 1] = obj;
    valueList = tmp;
}

// insert into a list
public void insert(SNLObject index, SNLObject obj) {
    int insertLocation = index.getInt();
    SNLObject[] tmp = new SNLObject[valueList.length + 1];
    System.arraycopy(valueList, 0, tmp, 0, insertLocation);
    tmp[insertLocation] = obj;
    for (int i = insertLocation + 1; i < tmp.length; i++)
        tmp[i] = valueList[i - 1];
    valueList = tmp;
}

// remove index from a list
public SNLObject remove(SNLObject index) {
    int rmLocation = index.getInt();
SNObject[] tmp = new SNObject[valueList.length - 1];
System.arraycopy(valueList, 0, tmp, 0, rmLocation);
SNObject ret = valueList[rmLocation];
for (int i = rmLocation; i < tmp.length; i++)
    tmp[i] = valueList[i + 1];
valueList = tmp;
return ret;
}

// remove from the tail of a list
public SNObject remove_back() {
    return remove(new SNObject(valueList.length - 1));
}

// get the length of the list
public SNObject length() {
    return new SNObject(valueList.length);
}

public String toString() {
    switch (type) {
    case INT:
        return Integer.toString(getInt());
    case FLOAT:
        return Double.toString(getFloat());
    case BOOL:
        return Boolean.toString(getBool());
    case STRING:
        return getString();
    case LIST:
        String s = "[ ";
        for (int i = 0; i < valueList.length - 1; i++) {
            s = s + valueList[i].toString() + ", ";
        }
        s = s + valueList[valueList.length - 1].toString() + "]";
        return s;
    }
    return null;
}

Listing B.9: SNObject.java

B.10  Test Script

#!/usr/bin/env python
import argparse
import glob
import os
import shutil
import subprocess
import tempfile

AST_BIN = ".\snlc"
TOTAL_PASS = 0
TOTAL_FAIL = 0
PARSER = ArgumentParser(description='Run SNL tests."
PARSER.add_argument('-v', action='store_true',
        help='Print all passing tests."
args = parser.parse_args()

def run_ast_tests(files, cmd_arg):
    """
    Runs tests to build ASTs from code files.
    The input files must end with the extension '.snl', which may not appear
    anywhere else in the file name.
    The expected output files must be named exactly as the input files except
    that they end with the extension '.out' instead of '.snl'.
    Args:
        files: a list of the names of input files, all of which end in '.snl',
        cmd_arg: the corresponding argument to pass into the AST-printing binary,
        e.g. '-e' to test expr and '-s' for stmt.
    """
    global TOTAL_PASS
    global TOTAL_FAIL
    for test in files:
        with open(test.replace('.snl', '.out'), 'r') as f:
            expected_output = f.read()  
        try:
            output = subprocess.check_output([AST_BIN, cmd_arg, test])
        except subprocess.CalledProcessError as e:
            print 'Error processing %s\n' % test, e
            TOTAL_FAIL += 1
            continue
        if expected_output != output:
            TOTAL_FAIL += 1
            print '\n\nFAIL: %s' % test
            print 'EXPECTED:
%s' % expected_output
            print 'ACTUAL:
%s' % output
        else:
            TOTAL_PASS += 1
        if args.v:
            print 'PASS: %s' % test

def run_expr_tests():
    """
    Runs all the tests in the tests/expr directory.
    """
    print 'Running expr tests...'
    expr_tests = glob.glob('tests/expr/*.snl')
    run_ast_tests(expr_tests, '-e')
    print 'Finished running expr tests.\n'

def run_stmt_tests():
    """
    Runs all the tests in the tests stmt directory.
    """
    print 'Running stmt tests...'
    stmt_tests = glob.glob('tests/stmt/*.snl')
    run_ast_tests(stmt_tests, '-s')
    print 'Finished running stmt tests.\n'
```python
def run_program_tests():
    """
    Runs all the tests in the tests/program directory.
    """
    print('Running program tests...')
    program_tests = glob.glob('tests/program/*.snl')
    run_ast_tests(program_tests, '-p')
    print('Finished running program tests.

def run_failing_tests():
    """
    Runs all the tests in the tests/failing directory.
    """
    global TOTAL_PASS
    global TOTAL_FAIL
    print('Running failing tests...')
    failing_tests = glob.glob('tests/failing/*.snl')
    with open(os.devnull, 'wb') as DEVNULL:
        for test in failing_tests:
            try:
                output = subprocess.check_output([AST_BIN, '-j', test,
                    '-output_path', os.devnull],
                    stderr=DEVNULL)
            print('
FAIL: %s % test
TOTAL_FAIL += 1
except subprocess.CalledProcessError as e:
    TOTAL_PASS += 1
    if args.v:
        print('PASS: %s % test
print 'Finished running failing tests.

def run_java_tests():
    """
    Runs all the tests in the tests/java directory.
    """
    global TOTAL_PASS
    global TOTAL_FAIL
    print('Running compiler tests...')
    compiler_tests = glob.glob('tests/java/*.snl')
    temp_dir = tempfile.mkdtemp()
    subprocess.call(['javac', '-d', temp_dir, 'SNLObject.java'])
    with open(os.devnull, 'wb') as DEVNULL:
        for test in compiler_tests:
            with open(test.replace('snl', 'out'), 'r') as f:
                expected_output = f.read()
            try:
                name = test[len('tests/java/'):-len('snl')]
                subprocess.call([AST_BIN,
                    '-j', test,
                    '-output_path', temp_dir],
                    stdout=DEVNULL)
                subprocess.call(['javac', '-d', temp_dir] +
                    glob.glob(temp_dir + '/.java'))
                output = subprocess.check_output(['java',
                    '-classpath', temp_dir, name])
            except subprocess.CalledProcessError as e:
                print('Error processing %s
' % test, e
```

TOTAL_FAIL += 1
continue
finally:
    for f in os.listdir(temp_dir):
        if not f.startswith('SNLObject'):
            os.remove(os.path.join(temp_dir, f))
    if expected_output != output:
        TOTAL_FAIL += 1
        print \n\nFAIL: %s % test
        print 'EXPECTED: %s' % expected_output
        print 'ACTUAL: %s' % output
    else:
        TOTAL_PASS += 1
        if args.v:
            print 'PASS: %s % test
        print 'Finished running compiler tests.\n'
def main():
    run_expr_tests()
    run_stmt_tests()
    run_program_tests()
    run_failing_tests()
    run_java_tests()
    print '%d out of %d tests passing.' % (TOTAL_PASS, TOTAL_PASS + TOTAL_FAIL)
if __name__ == '__main__':
    main()

Listing B.10: run_tests.py

B.11 Makefile

```
compiler: snlc.ml objects
    ocamic -c snlc.ml
    ocamic -o snlc ast.cmo parser.cmo scanner.cmo str.cmo codegen.cmo analyzer.cmo snlc.cmo
objects: scanner parser generator
    ocamic -c ast.ml sast.ml parser.mli scanner.ml parser.ml analyzer.ml codegen.ml
generator: analyzer.ml codegen.ml
parser: parser.mly
    ocamlyacc -v parser.mly
scanner: scanner.mll
    ocamlllex scanner.mll
.PHONY: test
  test: compiler
    ./run_tests.py
  .PHONY: clean
  clean:
      rm -f parser.mli scanner.ml parser.ml parser.output *.cmo *.cmi snlc ~
```
B.12 Tests

See ‘tests’ folder in source code directory.