ProbL: Probabilistic Modeling Language
Final Report

Nir Grinberg, Andrew Wong, Diana Liskovich, Sam Tkach
{ng2470,aw2192,dl2956,st2794}@columbia.edu

July 3, 2015

Contents

1 Introduction

The vast amounts of data readily available for processing (a.k.a the buzz around "Big Data") have created an even greater need for careful modeling and analysis of these large datasets. One of the most common techniques for modeling data is using probabilistic models, in particular, using Directed Graphical Models (DGM's), which describe the dependencies between random variables in a probabilistic model. DGM's are Directed Acyclic Graphs (DAG's) where nodes are random variables and directed edges, connecting node u to v, represent parenting such that: $P(u, v) = P(u)P(v|u)$. Despite the somewhat confining definition, DGM's are suitable for modeling a wide range of important artificial intelligence and machine learning models: from generalized linear regression, factor analysis and PCA, through hidden markov models, time-series models, Kalman filters, Bolzman Machines and hierarchical mixture models.

Still, specifying a probabilistic graphical model in C, Java or even R is not native to the language, which oftentimes implies that people re-implement common statistical inference algorithms (e.g. Gibbs Sampling) for each model. The use of third-party packages like *mcmc* in R is restrictive and does not easily allow for user-defined distributions or functions. Third-party modeling software such as WinBUGS does allow for model specification, but suffers from the same restrictive shortcomings as mentioned before.

Probl is meant to be an answer to these problems. It is designed to be simple to use for statisticians and mathematicians who need to do parameter estimation on some distribution without needing to develop a structured environment for these computations.

2 Language Tutorial

Probl is a language that is easy to pick up for anyone familiar with C-like imperative languages. All programs in the language take this form:

2.1 Program Structure

```
input
{
/* ... block of variable declarations ... */
}

output
{
/* ... block of variable declarations ... */
}

/* ... any number of function declarations ... */

model
{
/* ... block of statements ... */
}
```

The variables contained in the input and output blocks are considered global. The input variables are automatically populated by reading from a csv file holding data. The output variables are automatically written to an output file. Specifics and syntax are similar to C, and will be covered in the next section.

2.2 Probabilistic Assignments

ProbL is meant to abstract out all the technical details of statistical inference in graphical models. The strength of graphical models lays in their ability to represent a wide range of statistical models. Therefore, our language should be able to represent, for example, the following use cases (and many others):

- Linear regression
- Principle Component Analysis (PCA)
- Latent Dirichlet Allocation (LDA)
- Hidden Markov Models (HMM's)

Our algorithm of choice was Gibbs Sampling, which is fairly simple, iterative, optimization process. All it takes to infer model parameters using Gibbs Sampling is a set of updating equations, given to the compiler when users specify probabilistic functions. For example, our sample test program ('tests/test-gibbs.probl') demonstrates inference of mean values for a model common in particle physics:
$P(x, y) = kx^2 * exp(-xy^2 - y^2 + 2y - 4x)$

In order to draw inferences about this model using Gibbs Sampling one needs to have the conditional distributions of $X|Y = y$ and $Y|X = x$, which for this model turn out to be fairly simple:

$P(X|y) = Gamma(3, y^2 + 4)$

$P(Y|x) = Normal(\frac{1}{1+x}, (\frac{1}{2(1+x)})^2)$

All it takes in ProbL to infer the means of the full model is to specify the above two updating equation, which can be done as follows:

```
fun~ bivar(): x, y {
  x = gamma_sample(3.0,y*y+4.0,0.0);
  y = 1.0/(x+1) + random_sample()/sqrt(2*x+2);
}
```

Our test program demonstrates that the program arrives at reasonable averages for this simple model - given average value of $y = 1.1$ the average value of $x = 15.5$, which is good numerical approximation for the average of a the gamma distribution in this case with $\alpha = 3.0$ and $\beta = 1.1^2 + 4 = 5.21$ of 15.63.

Of course, without including a mathematical library that has basic mathematical functions and matrix operations (e.g. for inverting a matrix) it's hard to implement more sophisticated models at this time. Nevertheless, our language contains all the necessary building blocks for implementing such features and allow for the specification of a wide range of statistical models, from linear regression to Kalman filters and more.

2.3 Compilation

Compilation is simple. To compile a program via the command line using a shell such as bash or zsh, run the following.

```
$ ./probl source_file.probl
```

This will output a .java file, which can be compiled using a Java compiler of the programmer's choosing. One of the nice sice effects of this is that, since nearly every computer has some version of Java, the programs are very portable.

3 Language Reference Manual

3.1 Types

3.1.1 Primitive Types

- int - The 32-bit int data type can hold integer values in the range of -2,147,483,648 to 2,147,483,647.
- float - The float data type stores real number values as double-precision floating-point numbers. Its minimum value is no greater than 1e37 and its maximum value is no less than 1e37.
- string - The string data type represents a sequence of characters.
- bool - The bool data type represents a boolean value. It takes one of two values: true or false.
- enum - The enum data type represents a variable that takes one value from a pre-defined set of values.

Here are some examples of declaring and defining a variable of type int:

```
int i = 100;
```

Here are some examples of declaring and defining a variable of type float:

```
float f = .101;
```

Here are some examples of declaring and defining a variable of type string:

```
string s = 'abc';
```

Here are some examples of declaring and defining a variable of type bool:

```
bool b = true;
bool b = false;
```

Here are some examples of declaring and defining an enum:

```
enum DIRECTIONS = {north, south, west east};
enum DIRECTIONS = {north, south, west, east} dir;
enum DIRECTIONS dir = north;
```

3.1.2 Collections

- array - The array data structure allows you store one or more elements consecutively in memory. Array elements are indexed beginning at position 0. Arrays can be one dimensional or two dimensional.

Here are some examples of declaring and defining an array:

```
int[3] a = [10, 20, 30];
bool[2][2] b = [[true, true],[false, false]];
b = [[true, true],[false, false]];
a[1] = 5;
b[0][0] = false;
```

3.2 Lexical Conventions

The objective of ProbL is to allow for easy specification of Directed Graphical Models and efficient inference of such models.

3.2.1 Control Flow

ProbL would support standard looping, conditional statements and block control flows as in *C*. The table below summarizes this.

;	end line
/* */	begin/end comment block
for, while	standard looping constructs
if, else	standard conditional statement
{ }	code block start and end (respectively)

Table 1: Control Flow

3.2.2 Comments

ProbL has the same multi-line comment methods as C.

3.2.3 Identifiers

Identifiers are used to create variables and functions. Each identifier will be a combination of digits, letters, and symbols. Letters can be lowercase or uppercase ASCII characters, ProbL will be case sensitive. Digits will be the ASCII characters of 0-9.

3.2.4 Keywords

ProbL reserves the following words for various purposes. As such, they should not be used as identifiers.

int	data type for an integer
float	data type for a floating point number
enum	data type taking one value from a pre-defined set of values.
string	data type for a sequence of ASCII characters
if, else	conditional statement
bool	data type for a Boolean value
true, false	boolean literals
while	execute and loop the contents until the condition is false
and, or, not	boolean logical operators
fun	defines a standard function taking arguments and returning a value of a given type
print	prints information to standard out
return	returns a value
input, output, model	denote data, parameters and model section of the program

Table 2: Keywords

3.2.5 Operators

Our most important operator is the $\sim$ operator, which defines the distribution of a random variable. For instance, in the sample program below we define y to be normally distributed using the $\sim$ operator with parameters that correspond to a linear regression model. The operators are:

!=, ==, <, <=, >, >=	Numerical relational
+, -, *, /, %, ^	Arithmetic
and, or, not	Logical
$\sim$	Sample a value from the distribution on the RHS
=	Assignment

Table 3: Operators

Mathematical operators act differently in the context of the $\sim$ operator.

3.2.6 Constants

Constants are Boolean types or sequences of digits. For a Boolean type, a constant is true or false. For a sequence of digits, a minus sign indicates negative value.

3.2.7 Whitespace

Blank characters represent Whitespace. Whitespace is ignored by the compiler and is used to separate tokens from one another, unless they are inside a string literal of course.

3.2.8 Scope

Scope of variables is straightforward in ProbL. If a variable is declared outside a block (i.e.- loop, conditional, or function), it is considered as a global variable in that file. Variables declared inside blocks are local to those blocks. In nested blocks, variables can be referenced in the block in which it was defined as well as in any block inside that block. The input and output variables are considered global variables.

3.3 Syntax

Much of the ProbL syntax is similar to that of C. Function definitions, conditional statements, and loops work largely the same way. There are, however, several syntactic qualities unique to ProbL. The following sections will cover the syntax for all aspects of the language.

3.3.1 Operators and Symbols

Assignment in ProbL works as follows. The = operator is used for all non-probabilistic datatypes, while the $\sim$ operator is used to assign a distribution to a random variable or to link random variables conditionally. Probabilistic assignment is evaluated differently from regular assignment in a few ways.

```
/* probabilistic function declaration */
fun~ f(float[] x, float[] y):a,b { ... }

/* probabilistic function call */
float a;
float b;
a,b ~ f(x,y);
```

Arithmetic symbols for addition, subtraction, multiplication, division, and exponentiation are binary operators. So are the two assignment operators, and

The following are miscellaneous (non-operator) symbols used for various things in the language:

/* */	begin and end comment block
;	signifies end of a line of code
:	used in function declaration
,	used to separate arguments, elements of collections
{ }	begin or end a block of code, define enums
( )	used for arithmetic grouping, functions, loops, and conditionals
[ ]	used for array element access and instantiation, define collections
' '	used to denote string literals

Table 4: Symbols

3.3.2 Function Definitions

Functions are defined in the following form. When writing a normal function, use the "return" keyword to make it return a value.

```
/* normal function */
fun function_name (type1 arg1, type2 arg2,...):return_type
{
        /* ... function code here ... */
        return output_value;
}

/* probabilistic function */
fun~ function_name (<args as above>):<comma-delimited parameters>
{
        /* parameters assigned to their gibbs update functions */
}
```

All functions must return a value.

3.3.3 Loops and Conditionals

ProbL supports for and while loops as well as if-else statements similar to those in C. while and if take a boolean expression, and for uses a more complicated syntax. Here is an example:

```
for(int i ; i<10 ; i = i + 1) /* iterate int i from 0 to 10 */
```

```
{                                               /* with step size 1 */
        int j = 0;
        while(j < i)
        {
                j = j + 1;
        }

        if(j == i)
        {
                j = j - 1;
        }
        else
        {
                j = i;
        }
}
```

3.3.4 Modeling

The core functionality of the language is its modeling capabilities. Each program needs to have a modeling block, which defines the graphical model to apply to data. Using the optional "input" and "output" keywords we let programmers of ProbL specify what data to use and which parameters to estimate. Since reading input and estimating parameters is the core of our language we'll provide standard ways for our users to read in and output a csv file in a simple format. This currently consists of a user providing an input file named input.csv consisting a header for each column of data specifying the variable name. Once compiled, executing a program that uses the optional input and/or output blocks would require an argument passed in that specify where in the file system the input/output should be. For instance, here is the skeleton of a program that makes use of these features on 2 arrays of data, x and y:

```
input {
        /* data */
  float[] x;
  float[] y;
}
output {
  /* variable declarations of parameters to estimate */
}
fun~ norm(float[] x, float[] y):a, b
{
/* gibbs sampling updating equations for parameters (i.e.- a, b) */
}

model {
        /* block of statements to execute */
        a,b ~ norm(x,y);
```

}

This would return an array of float containing the estimated parameters in the order in which they were declared. These features can also be used outside of a function. See the sample program for a concrete example.

3.4 Standard Library Functions

ProbL provides a minimal set of functions that are at the core of any statistical model represented and inferred using ProbL. These include the basic `rand` function to generate random samples from a uniform distribution, basic I/O functions and a `len` function to find the length of an array or a string.

3.4.1 Randomization

The most essential building block for randomization is the function `rand`:

- `rand():float` - generates a float in the range 0 to 1 uniformly at random
- `srand(int seed):int` - initialize the pseudo-random process using the provided seed.

Using these building blocks, more complex statistical distributions can be defined. A common sampling technique is using the inverse cumulative distribution function (CDF) with a uniform sample from $[0, 1]$, which can be easily obtained using `rand`.

3.4.2 I/O functions

We will support a few simple I/O functions:

- `print(string str):int` - print the provided string to programs output.
- `open(string filename, string mode):int` - returns a file descriptor for the function opened in the specified mode ('r' for reading text files or 'w' for writing text files).
- `close(int file):int` - closes the file specified using `file`.
- `read(int file):string` - read till the end the previously opened file.
- `write(int file, string content):int` - writes `content` into previously opened file.

3.4.3 Miscellaneous

- `len(string str):int` - returns the length of the provided string.
- `len(int[] a):int` - returns the length of the first dimension of an array (of integers in this example).

3.5 Sample Programs

Refer to section 5.6 for current test programs and their respective generated Java programs.

3.5.1 Linear Regression

Below is an example program that given two vectors (x and y) of observed variables would infer the parameters of the linear regression model (a, b and *sigma*):

```
input {
        float[] x;
        float[] y;
}
output {
        float a;
        float b;
        float sigma;
}
fun~ norm(float[] x, float[] y):a,b,sigma
{
        /* Gibbs sampling update computations for each parameter being estimate
}
model {
        a,b,sigma ~ norm(x, y);
}
```

3.5.2 Baysian Model of Coin Tosses

Below is an example program that given a vector (y) of observed outcomes of a coin toss infers the probability the coin is biased (*theta*):

```
input {
        bool[] y;
}
output {
        float theta;
}
fun beta_dist(float a, float b): float {
        /* implementation of the beta distribution, */
        /* returns a number in (0,1) */
}
fun gamma_dist(float alpha, float beta): float {
        /* implementation of the gamma distribution, */
        /* returns a number in (0,inf) */
}
fun binomial(float theta): bool {
        /* implementation of the binomial distribution, */
        /* returns a boolean. */
}
```

```
model {
        a ~ gamma_dist(0.5, 0.8);
    b ~ gamma_dist(1.0, 2.0);
    theta ~ beta_dist(a, b);
        y ~ binomial(theta);
}
```

4 Project Plan

The process for creating Probl was fairly standard to the class. We divided up work into individual parts when possible, using git (https://github.com/deeml/ProbL) to collaborate on the code. The compiler itself was written in Ocaml, and our target language was Java, although we originally planned for it to be C.

4.1 Team Roles

The following responsibilities were set at the beginning of the development process. All four of us were to work together on the scanner and the parser. Once these were in relatively good shape, we split the work:

Nir Grinberg	probabilistic functionality, test programs, code generation
Diana Liskovich	Input and Output, code generation
Sam Tkach	parsing, code generation, and semantic checking
Andrew Wong	tests and documentation

Table 5: Project Responsibilities

4.2 Timeline

6.17.15	Scanner and Parser built
6.24.15	Hello World successfully compiled
6.25.15	Java generation for general programs
7.1.15	Generation of probabilistic code
7.3.15	Semantic checking completed

Table 6: Project Timeline

4.3 Development Environment

We each used our own preferred development environment, as well as the Unix command line, since we all use either Mac or Linux. Text editors like vim and sublime were our main platforms for writing Ocaml code.

4.4 Programming Style Guide

Here, we define certain useful practices to make code written in Probl easy to read and understand. The fixed program structure in our language goes a long way in forcing the programmer to lay out code sensibly. The other main philosophical point here is "verbose" programming– using whitespace and splitting statements onto multiple lines. While this is not strictly necessary, it makes understanding code blocks (especially those that make up the body of distribution estimation functions, which can be heavy on arithmetic calculations) somewhat easier. This includes giving curly braces their own lines surrounding blocks, and including whitespace in between sections of code that are conceptually distinct.

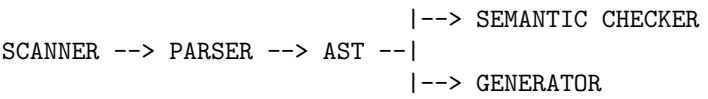
4.5 Project Log

See section 4.2 for an overview, or see Appendix B for the full git log.

5 Architectural Design

5.1 Structure

The following is a flow diagram showing the modules that are included in our translator's pipeline and the order in which they interact with one another.



5.2 Inter-Component Interfaces

The components of our translator follow a linear path of interaction. The scanner tokenizes the source code, and outputs this to the parser, which constructs the abstract syntax tree for the language based on the grammar and rules specified. Once the tree is built, it is passed to the generator and the semantic checker. The semantic checker traverses the AST looking for errors such as type mismatches, undeclared/previously declared functions and variables, and making sure expressions are of the proper type in specific situations (e.g.- return type, loop statements, conditionals). It then reports these errors, and the generator creates a Java program by traversing the tree and using string generation rules.

5.3 Component Implementation Roles

Scanner	All
Parser	Nir, Sam, Diana
AST	Sam, Nir, Diana
Semantic Checker	Sam
Generator	Sam, Nir, Diana

Table 7: Component Authors

5.4 Testing Process

Our tests check both the generated output Java code as well as the output from execution of the program. Our testing process was somewhat disorganized, since we struggled getting the basic flow of statements and control of a program correctly. Until the order of statements is not done it's hard to write separate test programs and work incrementally. However, once the basic compiler was in place, we stated working and testing programs more methodologically with small incremental changes.

All tests can be found under the folder 'tests', with 3 files for each test: test-hello-world.probl is the test program in ProbL, test-hello-world.java is the desired java output and test-hello-world.out is the desired output from the program. All tests are run through a shell script called ./testall, adapted from the MicroC script.

5.5 Responsibilities

All tests and scripts in the test suite were written by Nir except for input and output test written by Diana.

5.6 Sample Test Program

```
input{}
output{}
fun random_sample(): float {
    float u;
    u = rand();
    if (u<0.5) {
        return sqrt(-0.5*3.141593*log(1-(2*u-1)^2));
    } else {
        return -sqrt(-0.5*3.141593*log(1-(1-2*u)^2));
    }
}
model
{
    srand(1234);
    int i;
    i=0;
    float avg;
    avg=0.0;
    while (i<1000) {
        avg = avg + random_sample();
        i = i+1;
    }
    print('close to zero');
    print(avg/1000);
}
```

6 Lessons Learned

The following are individual statements from our team members about our learning experience during this project.

6.1 Nir Grinberg

Overall, I believe the class gave me the theoretical and practical foundations for thinking about compilation. It was nice to learn about the theoretical achievements that made compilation possible and efficient. The use of Ocaml made it very simple and with relatively smooth process for building a decent compiler. Working on the project itself was hard - the amount of time it took us to get to a point where we can divide up the work and progress incrementally was relatively long. Contributed to that are the somewhat cryptic error messages from Ocaml. In addition, the fact that all of us live and work on different schedules made it hard to co-locate and work together in person. Nevertheless, I think we arrived at nice proof-of-concept that definitely taught us a lot.

6.2 Diana Liskovich

This class was an very interesting learning experience, both in terms of new material, exploring old concepts in new ways, and in terms of project management and collaboration. Using Ocaml forced me to think differently when writing code, and deciding on a target language challenged me to think about the details of C and Java as well as their limitations. In general this was a huge amount of code to produce, and making sure all of us were able to add to the source without creating conflicts required planning and communication. One challenging part of the project was finding ways for us to work together efficiently even though we all live in different places and have very different schedules. Various online tools like GitHub and Overleaf were very useful. One lesson we learned right away was to definitely make smaller increments to our code and thoroughly test it before moving on. We made the mistake of implementing too much at first, but that guided out work for the rest of the term where we broke up goals even more and made sure to complete tests before moving on. Overall working on creating our own language allowed us to learn a great deal not only about compilers but teamwork as well, and different ways of thinking. My advice for future groups is to proceed in very small increments, make sure to have test cases early on, be flexible with the idea of the language, and make it as easy as possible to communicate with your team members.

6.3 Sam Tkach

Over the course of the term, I think my learning experience could be described in three ways. First, designing and implementing a programming language would be a very rewarding process if I had the time to commit to it. Second, working with a team is difficult if you are not all centrally located. Finally, I shouldn't have planned on sleeping at all. Our project was rushed all the way to the finish, and unfortunately, it shows. My advice to future students would be to start much earlier than you think you should, and to make sure there is no miscommunication amongst your teammates as to who is responsible for what.

This was definitely a pitfall of ours. Regardless of the mistakes made along the way, learning how to use Ocaml was probably my favorite part of the class as someone who hadn't dabbled in functional programming before this. I certainly gained an appreciation for the more strictly mathematical paradigm that it has to offer, and will probably use Ocaml for future projects, which is not something that I would have guessed at the beginning of the term.

6.4 Andrew Wong

As is very first summer course i have ever taken, this class was a very different experience. The class was went a lightening speed, which is always difficult when learning something new. Learning a new language, OCaml, forced me to have a new perspective when coding and learning about compilers enlightened me on a topic that I always knew was very crucial to Computer Science, but never really paid attention to. The key to completing this project was collaboration, and our team was very successful at using modern online collaboration tools. Working mostly with GitHub, Overleaf, and Google Docs allowed us to collaborate as much as possible without getting in each other's way. One thing i learned was how simple changing a code generator is. At first, our language was going to generate C code, but after we ran into many issues, we changed to generating Java. At first i was very worried, I thought that changing to generating a completely different language must lead to more issues. Fortunately, i was incorrect as changing the generator turned out to be simple. My advice for future groups is to meet more often and regularly in person. Although the online collaboration tools are great, sometimes what is needed most is multiple eyes on the same problem and the ability to discuss and brainstorm, which is very difficult to do over the internet.

7 Appendix A: Project Source

scanner.mll

```
{ open Parser }

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '['      { LSQRPAREN }
| ']'      { RSQRPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| ';'      { SEMI }
| ':'      { COLON }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '%'      { MODULO }
| '^'      { EXPONENT }
| '='      { ASSIGN }
| '~'      { PROBASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "not"    { LOGICALNOT }
| "and"    { LOGICALAND }
| "or"    { LOGICALOR }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "fun"    { FUNC }
| "return" { RETURN }
| "int"    { INT }
| "float"  { FLOAT }
| "bool"   { BOOLEAN }
| "string" { STRING }
| "enum"   { ENUM }
| "input"  { INPUT }
| "output" { OUTPUT }
| "model"  { MODEL }
| ['0'-'9']*['.']['0'-'9']+ as lxm { FLOAT_LITERAL (float_of_string lxm) }
```

```
| ['0'-'9']+ as lxm { INT_LITERAL(int_of_string lxm) }
| '\''([^ '\'']* as lxm)'\'' { STRING_LITERAL(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| "true"   { TRUE }
| "false"  { FALSE }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment = parse
  "*/" { token lexbuf }
| _    { comment lexbuf }
```

parser.mly

```
%{ open Ast %}

%token SEMI COLON LPAREN RPAREN LBRACE RBRACE LSQRPAREN RSQRPAREN COMMA
%token PLUS MINUS TIMES DIVIDE MODULO EXPONENT ASSIGN PROBASSIGN
%token EQ NEQ LT LEQ GT GEQ
%token RETURN IF ELSE FOR WHILE INT
%token LOGICALNOT LOGICALAND LOGICALOR
%token FUNC
%token INT FLOAT BOOLEAN STRING ENUM
%token INPUT OUTPUT MODEL
%token <int> INT_LITERAL
%token <string> STRING_LITERAL
%token <float> FLOAT_LITERAL
%token <string> ID
%token EOF
%token RETURN
%token TRUE FALSE

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left LOGICALAND LOGICALOR
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MODULO
%left EXPONENT
%nonassoc UMINUS
%nonassoc LOGICALNOT

%start program
%type <Ast.program> program

%%

program:
    /* nothing */ { [], [], [], [] }
 | INPUT LBRACE vdecl_opt RBRACE OUTPUT LBRACE vdecl_opt RBRACE fdecl_list
   MODEL LBRACE stmt_list RBRACE { $3, $7, $9, $12 }

fdecl_list:
  | fdecl_opt { List.rev $1 }

fdecl_opt:
    /* nothing */ { [] }
  | fdecl_opt fdecl { $2 :: $1 }
```

```
fdecl:
   FUNC ID LPAREN formals_opt RPAREN COLON dtype LBRACE stmt_list RBRACE
     { { fname = $2;
   formals = $4;
       params = [];
     body = $9;
     ret_type = $7 } }
 | FUNC PROBASSIGN ID LPAREN formals_opt RPAREN COLON actuals_opt LBRACE
  stmt_list RBRACE
    { { fname = $3;
        formals = $5;
        params = $8;
        body = $10;
        ret_type = Float } }

dtype:
    INT     { Int }
  | FLOAT   { Float }
  | BOOLEAN { Boolean }
  | STRING  { String }
  | dtype LSQRPAREN INT_LITERAL RSQRPAREN { Array($1, $3) }

formals_opt:
    /* nothing */ { [] }
  | formal_list   { List.rev $1 }

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

type_decl:
    dtype ID { { vname = $2; vtype = $1 } }


vdecl_opt:
   vdecl_list   { List.rev $1 }

vdecl_list:
  /*  nothing */  { [] }
  | type_decl                    { [$1] }
  | vdecl_list SEMI type_decl       { $3 :: $1 }
  | vdecl_list COMMA type_decl SEMI { $3 :: $1 }
  | vdecl_list SEMI      { $1 }

stmt_list:
    stmt_list_opt { List.rev $1 }

stmt_list_opt:
    /* nothing */  { [] }
```

```
  | stmt_list_opt stmt { $2 :: $1 }

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

stmt:
    expr SEMI { Expr($1) }
  | RETURN expr SEMI { Return($2) }
  | LBRACE stmt_list RBRACE { Block($2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt
     { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expr_opt SEMI expr_opt RPAREN  stmt
     { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }
  | type_decl SEMI { Vdecl($1) }
  | var_list PROBASSIGN ID LPAREN actuals_opt RPAREN SEMI { Pcall($1, $3, $5)  }

expr_opt:
    /* nothing */ { Noexpr }
  | expr          { $1 }

expr:
    INT_LITERAL      { Int_literal($1) }
  | FLOAT_LITERAL    { Float_literal($1) }
  | STRING_LITERAL   { String_literal($1) }
  | ID               { Id($1) }
  | expr PLUS   expr { Binop($1, Add,   $3) }
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | MINUS expr %prec UMINUS { Binop(Int_literal(0), Sub, $2) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | expr EXPONENT expr { Binop($1, Exp, $3) }
  | expr EQ     expr { Binop($1, Equal, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater,  $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
  | expr LOGICALAND expr { Binop($1, And,   $3) }
  | expr LOGICALOR expr { Binop($1, Or,   $3) }
  | LOGICALNOT expr  { Lnot($2) }
  | ID ASSIGN expr   { Assign($1, $3) }
  | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
  | LPAREN expr RPAREN { $2 }

actuals_opt:
    /* nothing */ { [] }
  | actuals_list  { List.rev $1 }
```

```
actuals_list:
    expr                    { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }
```

ast.ml

```
type op = Add | Sub | Mult | Div | Exp | Mod | Equal | Neq | Less | Leq | Greater | Geq |

type dtype =
    Int
  | Float
  | Boolean
  | String
  | Array of dtype * int
  (* | Enum *)
  (* | Array of dtype * int *)

type expr =
    String_literal of string
  | Float_literal of float
  | Int_literal of int
  | Id of string
  | Binop of expr * op * expr
  | Lnot of expr
  | Assign of string * expr
  | Call of string * expr list
  | Noexpr

type var_decl = {
    vname : string;
    vtype : dtype;
  }

type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt
  | Vdecl of var_decl
  | Pcall of string list * string * expr list

type func_decl = {
    fname : string;
    formals : var_decl list;
    params : expr list;
    body : stmt list;
    ret_type : dtype;
  }

type program = var_decl list * var_decl list * func_decl list * stmt list
```

generator.ml

```
open Printf
open Ast

(* primitive types *)
let rec string_of_type = function
      Int -> "Integer"
    | Float -> "double"
    | String -> "String"
    | Boolean -> "Boolean"
    | Array(t, l) -> string_of_type t ^ "[]"


(* global variable declaration *)
let string_of_vdecl vd = "public static " ^ string_of_type vd.vtype ^ " " ^ vd.vname
                         ^ match vd.vtype with
                            | Int -> " = 0;\n"
                            | Float -> " = 0.0;\n"
                            | Boolean -> " = false;\n"
                            | String -> " = null;\n"
                            | Array(t, l) -> " = new " ^ string_of_type t
                                             ^ "[" ^ string_of_int l ^ "];\n"

let string_of_formal f = string_of_type f.vtype ^ " " ^ f.vname

(* expressions *)
let rec string_of_expr = function
   (* String_literal(l) -> "\"" ^ l ^ "\"" *)
    String_literal(l) -> "\"" ^ l ^ "\""
  | Float_literal(l) -> string_of_float l
  | Int_literal(l) -> string_of_int l
  | Id(s) -> s
  | Binop(e1, o, e2) ->
    (match o with
        Exp -> "Math.pow(" ^ string_of_expr e1 ^ "," ^ string_of_expr e2 ^ ")"
      | _ ->
            string_of_expr e1 ^ " " ^
            (match o with
             Add -> "+" | Sub -> "-" | Mult -> "*" | Div -> "/"
            | Equal -> "==" | Neq -> "!="
            | Less -> "<" | Leq -> "<=" | Greater -> ">" | Geq -> ">="
            | And -> "&&" | Or -> "||"
            | _ -> "") ^ " " ^
            string_of_expr e2
    )
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | Lnot(e) -> "!(" ^ string_of_expr e ^")"
  | Call(f, el) -> (match f with
```

```
    | "print" -> "System.out.println(" ^ String.concat ", " (List.map string_of_expr el)^"
    | "srand" -> "__rand = new Random(" ^ String.concat ", " (List.map string_of_expr el)^
    | "rand"  -> "__rand.nextFloat()"
    | "log"   -> "Math.log(" ^ String.concat ", " (List.map string_of_expr el)^ ")"
    | "sqrt"  -> "Math.sqrt(" ^ String.concat ", " (List.map string_of_expr el)^ ")"
    | _  ->  f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
    )

  | Noexpr -> ""

(* statement *)
 let rec string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
  | Pcall(p, f, a) ->  let string_of_pcassign pn = pn ^ " = __" ^ f ^ "." ^ pn ^ ";\n" in
                         f ^ " __" ^ f ^ " = new program." ^ f ^ "();\n"
                       ^ "__" ^ f ^ ".run("
                           ^ String.concat ", " (List.map string_of_expr a) ^ ");\n"
                       ^ String.concat "" (List.map string_of_pcassign p)
  | Vdecl(v) ->
     string_of_type v.vtype ^ " " ^ v.vname
     ^ match v.vtype with
        | Int -> " = 0;\n"
        | Float -> " = 0.0;\n"
        | Boolean -> " = false;\n"
        | String -> " = null;\n"
        | Array(t, l) -> " = new " ^ string_of_type t
                         ^ "[" ^ string_of_int l ^ "];\n"


(* normal fdecl *)

let string_of_rfdecl f = "private static " ^ string_of_type f.ret_type ^ " "
                     ^ f.fname ^ "("
                     ^ String.concat ", " (List.map string_of_formal f.formals)
                     ^ ")\n"
                     ^ "{\n" ^ String.concat "" (List.map string_of_stmt f.body)
                     ^ "}\n"

(* sampling / parameter estimation *)
let sid pn = string_of_expr pn
```

```
let string_of_param pn = string_of_vdecl { vname = (sid pn) ; vtype = Float }

let string_of_pinit pn = "double[] __" ^ (sid pn) ^ " = new double[500];\n"
                       ^ (sid pn) ^ " = __rand.nextFloat();\n"

let string_of_pincr pn = "__" ^ (sid pn) ^ "[i] = " ^ (sid pn) ^ ";\n"

let string_of_ploop pfd = "for (int i=0; i<500; i++){\n"
                          ^ "for (int j=0;j<10;j++){\n "
                          ^ String.concat "" (List.map string_of_stmt pfd.body)
                          ^ "}\n" ^ String.concat "" (List.map string_of_pincr pfd.params)
                          ^ "}\n"

let string_of_mean pn = (sid pn) ^ " = mean(__" ^ (sid pn) ^ ");\n"


let string_of_pfdecl p = "private static class " ^ p.fname
                         ^ "\n{\n" ^ String.concat ""
                                     (List.map string_of_param p.params)
                         ^ "\n"

                         ^ "public double mean(double[] d) { double sum = 0.0;\n"
                         ^ "for(double i : d) { sum += i; } return "
                         ^ "sum/((double) d.length);\n}"

                         ^ "\npublic void run(" ^ String.concat ", "
                           (List.map string_of_formal p.formals) ^ ")\n{\n"

                         ^ "double[] res = new double[" ^ string_of_int
                           (List.length p.params) ^ "];\n"

                         ^ String.concat "" (List.map string_of_pinit p.params)

                         ^ string_of_ploop p

                         ^ String.concat "" (List.map string_of_mean p.params)
                         ^ "\n}\n}"

let string_of_fdecl f = if f.params = [] then
                                string_of_rfdecl f
                        else string_of_pfdecl f

let string_array_type = function
      Int -> "Integer"
    | Float -> "Double"
    | String -> "String"
    | Boolean -> "Boolean"
    | Array(t, l) -> string_of_type t
```

```
let string_create_arraylist var =
  "new ArrayList<" ^ string_array_type var.vtype ^ " >(Arrays.asList(" ^ var.vname ^ "))"

let string_create_var_namemap var = "VALUES_BY_NAME.put(\"" ^ var.vname ^ "\", " ^ string_

let string_create_var_typemap var = "TYPES_OF_VARS.put(\"" ^ var.vname ^ "\", \"" ^ string

let string_convert_to_array var =
  var.vname ^ " = ("^ string_of_type var.vtype ^") VALUES_BY_NAME.get(\"" ^ var.vname^ "\"


let string_of_get_input =
  "String __csvFile = \"input.csv\";\n
    BufferedReader __br = null;\n
    String __line = \"\";\n
    String __csvSplitBy = \",\";\n
    try {\n

    ArrayList<String> __variables = new ArrayList<String>();\n
    __br = new BufferedReader(new FileReader(__csvFile));\n
    int __j = 0;\n
    while ((__line = __br.readLine()) != null) {\n
              String[] __result = __line.split(__csvSplitBy);\n

              if (__j == 0)\n
              { \n
                int __size = __result.length;\n

                for(int __i=0; __i<__size; ++__i)\n
                {\n
                  if (VALUES_BY_NAME.containsKey(__result[__i]))\n
                  {
                    __variables.add(__result[__i]);\n
                  }
                  else {__variables.add(\"\");}\n

                  }\n
                ++__j;\n
                continue; \n

              }\n

              for(int __i=0; __i < __variables.size(); ++__i) \n
              {\n
                if (! __variables.get(__i).equals(\"\"))\n
                    {\n
                    if(TYPES_OF_VARS.get(__variables.get(__i)).equals(\"Integer\"))\n
                    {\n
                      VALUES_BY_NAME.get(__variables.get(__i)).add(Integer.parseInt(__resu
                    }\n
```

```
                    else if(TYPES_OF_VARS.get(__variables.get(__i)).equals(\"Double\"))
                    {\n
                      VALUES_BY_NAME.get(__variables.get(__i)).add(Double.parseDouble(__re
                    }\n
                    else if(TYPES_OF_VARS.get(__variables.get(__i)).equals(\"Boolean\"))\n
                    {\n
                      VALUES_BY_NAME.get(__variables.get(__i)).add(Boolean.parseBoolean(__
                    }\n
                    else \n
                    {\n
                      VALUES_BY_NAME.get(__variables.get(__i)).add(__result[__i]);\n
                    }\n
                }\n


        }\n

      }} catch (FileNotFoundException e) {\n
              e.printStackTrace();\n
              System.out.println(\"ProbL: Error - file not found.\");\n
            } catch (IOException e) {\n
              //e.printStackTrace();\n
            } finally {\n
              if (__br != null) {\n
                try {\n
                  __br.close();\n
                } catch (IOException e) {\n
                //e.printStackTrace();\n

                }\n
              }\n
            }\n"

let imports = "import java.io.*;\nimport java.util.*;\nimport java.lang.*;\n\n"

(* program *)
let string_of_program (input, output, funcs, stmts) =
  imports
  ^ "class program \n{\n"
    ^ "//input\n" ^ String.concat "" (List.map string_of_vdecl input) ^ "\n"
    ^ "//output\n" ^ String.concat "" (List.map string_of_vdecl output) ^ "\n"

    ^ "public static HashMap<String, ArrayList> VALUES_BY_NAME;\n"
    ^ "public static HashMap<String, String> TYPES_OF_VARS;\n"

    ^ "private static Random __rand = new Random(System.currentTimeMillis());\n"
    ^  String.concat "" (List.map string_of_fdecl funcs) ^ "\n"
    ^ "public static void main(String[] args){\n"

    ^ "VALUES_BY_NAME = new HashMap<String, ArrayList>();\n"
```

```
  ^ "TYPES_OF_VARS = new HashMap<String, String>();\n"
  ^ String.concat "" (List.map string_create_var_namemap input) ^ "\n"
  ^ String.concat "" (List.map string_create_var_typemap input) ^ "\n"
  ^ string_of_get_input ^ "\n"
  ^ String.concat "" (List.map string_convert_to_array input) ^ "\n"

   ^ String.concat "" (List.map string_of_stmt stmts) ^ "\nreturn ;\n"
  ^ "}\n"
^ "}\n"
```

semantics.ml

```
open Printf
open Ast

exception Error of string

(* types *)

type symbol_table = {
    parent : symbol_table option;
    mutable variables : Ast.var_decl list;
}

type environment = {
    scope : symbol_table;
    mutable funcs : Ast.func_decl list; (* always the same- functions are global *)
    return_type : Ast.dtype;    (* only used inside function definition *)
}

(*built-in functions*)
let core = [ "print"; "rand"; "srand"; "sqrt"; "log" ]

let rec string_of_type = function
    | Int -> "int"
    | Float -> "float"
    | String -> "string"
    | Boolean -> "bool"
    | Array(t, l) -> (string_of_type t) ^ "[]"

let rec find_var (scope : symbol_table) name =
    try
        List.find (fun v -> v.vname = name) scope.variables
    with Not_found ->
        match scope.parent with
          Some(parent) -> find_var parent name
        | _ -> raise Not_found

let find_func env name =
    if List.mem name core then { fname = name; formals = [];
                                 params = []; body = []; ret_type = Float}
    else
        List.find (fun f -> f.fname = name) env.funcs

let var_exists (scope : symbol_table) name =
    try let _ = find_var scope name in true
    with Not_found -> false

let func_exists env name =
```

```
    try let _ = find_func env name in true
    with Not_found -> false

(* building environments *)

let init_env : environment =
    let s = { parent = None;
            variables = [{ vname = "true"; vtype = Boolean};
                         {vname = "false"; vtype = Boolean}] } in
    { scope = s; funcs = []; return_type = Float; }


let create_f_env env f =
    let s = {  parent = Some env.scope; variables = env.scope.variables }
    in { scope = s; funcs = env.funcs; return_type = f.ret_type }


(* function declarations *)

let check_fname env f =
    if not ( func_exists env f.fname )
    then  env.funcs <- ( f :: env.funcs )
    else raise(Error("function " ^ f.fname ^ " is previously defined."))

(* variable declarations for input and output blocks *)

let vnames l = List.map (fun v -> v.vname) l

let check_vdecl env vd =
    if not ( List.mem vd.vname (vnames env.scope.variables) )
    then env.scope.variables <- (vd :: env.scope.variables)
    else raise(Error("variable " ^ vd.vname ^ " is previously defined."))

(* expressions *)

let rec check_expr env = function
    Int_literal(v) -> Int
  | Float_literal(v) -> Float
  | String_literal(v) -> String
  | Id(v) ->
          let vdecl = try
              find_var env.scope v
          with Not_found ->
              raise (Error("undeclared identifiier : " ^ v))
          in
          vdecl.vtype
  | Binop(e1, o, e2) ->
          let bool_ops = [Equal; Neq; Less; Leq; Greater; Geq; And; Or] in
          if (check_expr env e1) != (check_expr env e2) then
              raise(Error("expressions in binary operators should be of equivalent type"))
```

```
          else
              if List.mem o bool_ops then Boolean else check_expr env e1
  | Lnot(e) ->
          if (check_expr env e) != Boolean then
                    raise(Error("\'not\' operator was used on a non-boolean
                    expression"))
          else Boolean
  | Assign(v, e) ->
         if not (var_exists env.scope v) then raise(Error("variable " ^ v ^
         "was referenced before declaration"))
         else
             let vd = find_var env.scope v in
             if vd.vtype = check_expr env e then vd.vtype
             else raise(Error("type mismatch in assignment : " ^
             (string_of_type vd.vtype) ^ " " ^ vd.vname ^ " = "
             ^ string_of_type (check_expr env e)))


(*       try
              let vd = find_var env.scope v in
              if vd.vtype = check_expr env e then
                  vd.vtype
              else raise(Error("types must match for assignment."))
          with Not_found ->
              raise(Error("variable " ^ v ^ " assigned before declaration."))
*)
  | Call(f, l) -> if not (func_exists env f)
                    then raise(Error("function " ^ f ^ " is undefined"))
                    else let fd = (find_func env f) in fd.ret_type
  | Noexpr -> Int
(* statements *)

let rec check_stmt env = function
    Expr(s) -> check_expr env s ;
  | If(e, s1, s2) ->
          let et = check_expr env e in
          if et != Ast.Boolean then
              raise (Error("predicate of if statement must be of type bool"))
          else
              begin
                  check_stmt env s1;
                  check_stmt env s2;
              end

  | Vdecl(v) ->

          if var_exists env.scope v.vname then
               raise (Error("previously declared variable : " ^ v.vname))
          else
              env.scope.variables <- (v :: env.scope.variables); v.vtype
```

```
  | Return(e) ->
         let et = check_expr env e in
         if et != env.return_type then
             raise (Error("return types do not match"))
         else et

  | For(e1, e2, e3, s) ->
         if (check_expr env e1 = Ast.Int) &&
            (check_expr env e2 = Ast.Boolean) &&
            (check_expr env e3 = Ast.Int)
         then
             check_stmt env s
         else raise(Error("for loop expressions must be of types int, bool,
         int ."))

  | While(e, s) ->
         if check_expr env e != Ast.Boolean then
             raise(Error("while loop expression must be of type bool."))
         else
             check_stmt env s

  | Block(l) ->
         let helper sl = check_stmt env sl in
         let _ = (List.map helper l) in Boolean

  | Pcall(p, f, a) -> Array(Float, (List.length p))


(* check a function declaration for errors *)

let read_formals env f =
    let fn v = check_vdecl env v in
    List.map fn f.formals

let param_name = function
    Id(s) -> s
  | _ -> ""

let read_params env f =
    if f.params != [] then
        let fn p =
            env.scope.variables <- ({ vname = param_name p; vtype = Float } ::
                env.scope.variables)
        in List.map fn f.params
    else []

let check_fdecl env f =
    let e = create_f_env env f (* function body environment *)
    in let helper s = check_stmt e s
```

```
    in
    begin
        check_fname env f;
        read_formals e f;
        read_params env f;
        List.map helper f.body;
    end


(* vdecl list and fdecl list *)


let vlist_check env l =
    let helper k = check_vdecl env k in
    List.map helper l



let flist_check env l =
    let helper k = check_fdecl env k in
    List.map helper l

let block_check env l =
    let helper k = check_stmt env k in
    List.map helper l

(* program *)

let check_program p =
    let (input, output, funcs, model) = p
    and env = init_env in
    begin
        vlist_check env input;
        vlist_check env output;
        flist_check env funcs;
        block_check env model;
    end
```

8 Appendix B: Project Log

```
commit d2f155c93b63310fffc2defe881f850538dec008
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
Date:   Thu Jul 2 23:33:57 2015 -0400

    All working and clean

commit 7a9073236701e2f45f15b3365729197b3b0ad0a3
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
Date:   Thu Jul 2 22:45:02 2015 -0400

    with working gibbs, but other failures


commit 7618ded163d2ba6b8b67810df6a1cb6903ae98b1
Author: tkach <tkach.sam@gmail.com>
Date:   Thu Jul 2 19:42:11 2015 -0400

    add command-line args, semantic checking (currently throws warnings but works)

commit 2a70da95d1e54416e9bb426147e9d3b81f8647df
Author: tkach <tkach.sam@gmail.com>
Date:   Thu Jul 2 19:28:52 2015 -0400

    fix function declaration order

commit d81b26767cf9060afb2f6436a8ecaf69b79aaa42
Author: Diana <diana@Dianas-MBP.home>
Date:   Thu Jul 2 09:03:23 2015 -0400

    updated generator

commit c449c65d94f8e6dd7e855e103e5f2a08c2fd3636
Author: Diana <diana@Dianas-MBP.home>
Date:   Thu Jul 2 08:55:26 2015 -0400

    added input processing and test for it

commit ebc00ff3995014ea521019aef0648ba24f2df0ce
Author: Diana <diana@Dianas-MBP.home>
Date:   Thu Jul 2 07:19:54 2015 -0400

    changed types to class types

commit 260f641b19f4abab721110ba1ebe3281c3b19859
Author: Diana <diana@Dianas-MBP.home>
Date:   Thu Jul 2 07:14:55 2015 -0400

    changed types to class types
```

```
commit 993b329ddc98760cd63375118fa1cfe44b060e57
Author: Diana <diana@Dianas-MBP.home>
Date:   Thu Jul 2 07:13:24 2015 -0400

    changed types to class types

commit cbf6c5f02d0c0e7ba3cbe7b27169b0b5f7745110
Author: tkach <tkach.sam@gmail.com>
Date:   Wed Jul 1 16:48:46 2015 -0400

    generate java output for gibbs test

commit 429c7b45bcacc91c887cfcfeec1eea642f13466b
Author: tkach <tkach.sam@gmail.com>
Date:   Wed Jul 1 13:21:58 2015 -0400

     add probabilistic function generation

commit 3d824823c72e4fb126a26c413ac9607876429298
Author: tkach <tkach.sam@gmail.com>
Date:   Wed Jul 1 04:55:13 2015 -0400

    add parsing and generation for probabilistic features

commit b61014e4c8d0c245017c1e76042fd9b5fb38bab7
Author: tkach <tkach.sam@gmail.com>
Date:   Wed Jul 1 03:45:10 2015 -0400

    fix statement order, associativity; add some generation

commit 484803e611f35fb63af269dc31ad04e6c0405e14
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
Date:   Tue Jun 30 21:23:44 2015 -0400

    added and/or operators
    sample from gamma
    gibbs example

commit 8ead1993ea2930d2214700254363ea4b98fb30d5
Author: tkach <tkach.sam@gmail.com>
Date:   Tue Jun 30 19:54:41 2015 -0400

    fix shift-reduce conflicts

commit 31b2ab42a5ce5f6540ce182bed1b0e3242e26702
Merge: dc72a96 c01b287
Author: tkach <tkach.sam@gmail.com>
Date:   Tue Jun 30 19:31:48 2015 -0400
```

```
    Merge branch 'master' of https://github.com/deeml/ProbL

commit dc72a965ae9d45491b6f5490343802568e6c14ef
Author: tkach <tkach.sam@gmail.com>
Date:   Tue Jun 30 19:29:37 2015 -0400

    fix reduce-reduce conflicts

commit c01b28739af23f6e4027d0ff3baf76f06d49c729
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
Date:   Tue Jun 30 02:04:34 2015 -0400

    Added support math power operator, log, sqrt
    generate samples from Normal distribution

commit 76439fb5f987ad453861ffcdd685fd4d8508898c
Author: tkach <tkach.sam@gmail.com>
Date:   Sun Jun 28 01:05:24 2015 -0400

    parsing and code generation fixes

commit b2b376ab07d53dd6de66c0b065f085454c07d421
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
Date:   Sat Jun 27 18:11:11 2015 -0400

    added support for random numbers
    fixed order of statements

commit d21fd85b23fcec84d35579f624e7363c28407ce9
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
Date:   Sat Jun 27 17:35:17 2015 -0400

    fixed java codegen
    updated test suite for using java
    added support for floats

commit 4ef709c15ac90e968842eec61063e30a881d103c
Author: tkach <tkach.sam@gmail.com>
Date:   Sat Jun 27 17:01:00 2015 -0400

    begin java generation

commit 6500eba24af429c75d23b9b0fbe08d0ac7c5d392
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
Date:   Sat Jun 27 13:42:08 2015 -0400

    added testing capabilities: for each test program in tests/*.probl, provide tests/*.c

commit bfe4207cc09cec862e528bb3f6853ded0b128e33
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
```

```
Date:   Sat Jun 27 11:50:44 2015 -0400

    fixed string quotation and removed compiled files from repo

commit f37a60960cf041aaa6f48d94caad2e6268592e06
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
Date:   Sat Jun 27 11:37:19 2015 -0400

    cleaned some redundant files

commit dbdcc2c947d4d129b4dc89a2ed560d8e36808227
Author: tkach <tkach.sam@gmail.com>
Date:   Fri Jun 26 23:29:14 2015 -0400

    hello world now compiles to C

commit f9cd9355fa2e93ce5c27decdd2c937b1f36af12f
Author: tkach <tkach.sam@gmail.com>
Date:   Fri Jun 26 20:42:22 2015 -0400

    fully compiles, throws parsing error at runtime

commit 99ccc72328d55e2bae4686ea1e989b437cfbc67f
Author: tkach <tkach.sam@gmail.com>
Date:   Thu Jun 25 20:28:10 2015 -0400

    debugging full compilation process for hello world

commit f4e13de694296b3ad542fda71e8a1bfef89be1bc
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
Date:   Wed Jun 24 23:02:12 2015 -0400

    first version that actually compiles!!! (nothing tested yet..)

commit 65ad2d41ff34d56f227c71b0261d0e6262b0a7ff
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
Date:   Wed Jun 24 20:50:28 2015 -0400

    fixes to make lex + yak work

commit 52d2c6d3c735df8d92eb308613a1af701383b29f
Author: tkach <tkach.sam@gmail.com>
Date:   Tue Jun 23 18:37:05 2015 -0400

    work on parser and ast

commit a1b364448a80c3bdfaabf42887fa6c683b5fe9ee
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
Date:   Mon Jun 22 23:03:16 2015 -0400
```

```
    added types

commit 4d9e6e8a8d40ee1e26402a96422cead63134517c
Author: Diana <diana@dyn-209-2-218-109.dyn.columbia.edu>
Date:   Mon Jun 22 23:00:19 2015 -0400

    added fdecl stuff

commit efa76f357a36300ce2b32f3773a0a97d9fe1db49
Author: Diana <diana@dyn-209-2-218-109.dyn.columbia.edu>
Date:   Mon Jun 22 22:20:29 2015 -0400

    some changes

commit 66f4144c05dc1b982d3716800682e469c3789107
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
Date:   Mon Jun 22 21:46:08 2015 -0400

    added keyword model

commit f7fae1a0d864a7c889e8779d0299afced528d79d
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
Date:   Mon Jun 22 21:38:12 2015 -0400

    updated grammar

commit 2e1c9a838d3483f9635e037f9953aa3648a160a7
Author: Diana <diana@dyn-209-2-218-109.dyn.columbia.edu>
Date:   Mon Jun 22 21:13:17 2015 -0400

    added tokens

commit 21c9765f36fc1cb92be7e214d2aa0db04bba6156
Author: Diana <diana@dianas-mbp.usny.ibm.com>
Date:   Tue Jun 16 15:00:51 2015 -0400

    added to vdecl

commit 429c67e62686a42fbeb3756472cbfc651b3ebd2b
Author: tkach <tkach.sam@gmail.com>
Date:   Sun Jun 14 22:11:14 2015 -0400

    fix typos in grammar

commit 084333766500bbc2502d812b365d7b549201dc01
Author: tkach <tkach.sam@gmail.com>
Date:   Sun Jun 14 22:00:44 2015 -0400

    start work on parser; fix scanner
```

```
commit 903b408548fdb90c051cb0388d58027b31364f70
Merge: f0023b5 ff9ac94
Author: tkach <tkach.sam@gmail.com>
Date:   Sat Jun 13 19:17:41 2015 -0400

    Merge branch 'master' of https://github.com/deeml/ProbL

commit f0023b57e5548babee3d97ad5ba9ecf7044355c3
Author: tkach <tkach.sam@gmail.com>
Date:   Sat Jun 13 19:05:51 2015 -0400

    string literals and exponents in scanner

commit ff9ac94e9dc5e4494bcac3e75173fee703fc3bea
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
Date:   Sat Jun 13 18:54:32 2015 -0400

    added unary minus operator

commit c656db61965979a1229caef7f030464763c58c7a
Author: deeml <dl2956@columbia.edu>
Date:   Sat Jun 13 18:22:56 2015 -0400

    changed LITERAL to INT_LITERAL

commit 0b7a0b113c38366b327844c584ba63d82fbc57cf
Author: deeml <dl2956@columbia.edu>
Date:   Sat Jun 13 18:21:41 2015 -0400

    Added floats

commit 37e44b7c3c4aca7a09976ee04a3e53bd6ed1e4be
Author: tkach <tkach.sam@gmail.com>
Date:   Sat Jun 13 18:10:24 2015 -0400

    fix string literals for scanner

commit 11b1b49dfe22b05957bc6fb20c48e61acf84e4c1
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
Date:   Sat Jun 13 18:06:25 2015 -0400

    first version of our lecture

commit 03832120a36f85e732c8ca84bd9dc352179ac28e
Merge: 34c1467 f7b92a5
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
Date:   Sat Jun 13 16:22:27 2015 -0400

    Merge branch 'master' of https://github.com/deeml/ProbL
```

```
commit 34c1467bf76c6ca57d8763067515a7ff2050850c
Author: Nir Grinberg <grinberg.nir+github@gmail.com>
Date:   Sat Jun 13 16:20:16 2015 -0400

    initial commit, copying the MicroC example

commit f7b92a5b960ffaab41b2a218d01a18e3f3c0b5a5
Author: deeml <dl2956@columbia.edu>
Date:   Sat Jun 13 16:17:45 2015 -0400

    Update README.md

commit 24fc0a6788607f3d8d05ff44e3ec9aa3f924408e
Author: deeml <dl2956@columbia.edu>
Date:   Sat Jun 13 16:10:15 2015 -0400

    Initial commit
```