Chai - A language for Monte Carlo Simulation

Nikhil Bhat (nb2490)

October 28, 2013

1. Introduction

Chai is a language specifically designed for Monte Carlo simulation, i.e. simulation of random events. The language is strongly typed and has Random variables as a primitive type along with other numerical types. We also provide a type Matrix for random vectors. Each program of this language is contained has a definitions part and a simulation block, which should appear at the very end. Definitions part is used to build up a set of random variables from the built in ones. For this purpose the language is equipped with Functions, Generators and Stats. Generators are next sources randomness that is independent of each other.

- **Functions** are supposed to be like mathematical functions. They can map an \( \mathbb{R}^n \times \mathbb{I}^m \) to a number or an array.

- **Generators** are the source of randomness. The language with provide built in generators for the well known random variables.

- **Stats** are the opposite of Generators in the sense that they are used to integrate over the randomness. These will build on the built in Stat, the Eoperator.

Using these constructs and assignments. The last part of the program is the simulation block. This is analogous to the main function in programming languages such as Java and C. Here we can query summary stats from the random variables we can construct. The language will also provide tools to print these in a nice format.

2. Lexical Conventions

The language has the tokens identifiers, keywords, constants, expression operators and separators. White space characters are ignored except in the case of comments with the endline character. We use the convention of choosing the longest token.

2.1. Comments

All lines starting with # are ignored as comments.

2.2. Identifiers

Identifiers are similar to C - first character is an alphabetic and rest is a sequence of letters and digits.
2.3. Keywords

Following is the list of reserved keywords -

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>Integers</td>
</tr>
<tr>
<td>Float</td>
<td>Floating point numbers</td>
</tr>
<tr>
<td>Boolean</td>
<td>True or False</td>
</tr>
<tr>
<td>Matrix</td>
<td>A two dimensional array of Floats</td>
</tr>
<tr>
<td>RandFloat</td>
<td>Random number of Float type</td>
</tr>
<tr>
<td>RandInt</td>
<td>Random number of Int type</td>
</tr>
<tr>
<td>RandBoolean</td>
<td>Random Boolean</td>
</tr>
<tr>
<td>RandMatrix</td>
<td>Random Matrix</td>
</tr>
<tr>
<td>T[]</td>
<td>An array of a primitive type T, except an array</td>
</tr>
</tbody>
</table>

2.4. Constants

We have integer and floating point constants. Both follow the C conventions.

2.5. Types

The language has the following types:

We have arrays. However, there is no provision for multi dimensional arrays. Floats and RandFloats are called float-types, Ints and RandInts are int-types. All of these are numeric-types. Similarly we have matrix-types and boolean-types. All the types starting with Rand are grouped as Random types.

3. Expressions

The expression can be one of the following:

- Identifier
- Constant
- ( expression )
- expression1 [ expression2 ] - provided that expression1 is an array and expression2 is an Int.
- expression(comma separated expressions) - expression1 must be a Function or a Stat.
- expression(comma separated list of parameter assignments) - A generator will typically have named parameters, these can be assigned at the time of a call to the generator. For example, if a generator of called MyExp has a parameter named lambda, we can use the expression MyExp(lambda <- 0.5).
• unary-op expression - Where unary operator could be a - or a !, where the former expectes a numeric-type on the right side and the latter expects a boolean-type.

• expression1 binary-op expression2 - Binary operators could be numeric ones such as: +, -, *, / and **. All of which expect numeric-type or matrix-type. Further we could have relational operators such as >, <, >=, =<, == and !=. In addition, we have the binary operators on boolean-types, || and &&.

• Expectation - this has the form E(expression|conditional assignment list), where the '|' conditional assignment list' is optional. the conditional assignment list is a comma separated list of: identifier = constant.

• Assignment - Assignement can be of the form:
  – identifier = expression
  – array-identifier[int-constant] = expression

  The value of this expression is value assigned to the left side.

4. Statement
A statement can be either:

• expression ;
• if (expression) { statement-list }
• if (expression) { statement-list } else { statement-list }
• while (expression) {statement-list}

A statement list is a statement or a statement followed by a list of statements.

5. Arguments
A single argument has the syntax identifier:type, for example lambda:Float. An argument-list is comma separated list of arguments.

6. Function Declaration
A function declaration is of the form:
    Function identifier(argument-list):type { statement-list; return expression; }

7. Stat Declaration
Stat declaration is similar to the function declaration:
    Stat identifier(argument-list):type { statement-list; return expectation; }

    The diffence is that return type must be an expectation.
8. Generator Declaration

Generator declaration is of the form:

\[ \text{Generator identifier(argument-list)} \{ \text{statement-list; return expression; } \} \]

This is almost identical to the function declaration. But generators can be called by named arguments.

9. Simulation Block

Simulation block is of the form:

\[ \text{Simulate (seed = int-constant, trials = int-constant)} \{ \text{stat-list } \} \]

A stat list is a list of stat statements where, a stat statement can be:

- expectation;
- A call to a Stat followed by a ‘;’.

10. Built-ins

The language will provide some built-ins functions, generators and stats.

- NewMatrix, A function which takes two arguments which specify the size of the matrix. For example NewMatrix(2,2) creates a $2 \times 2$ matrix of all zeros.
- FloatArray, IntArray, BooleanArray, MatrixArray - functions used to create new arrays. It takes one argument which is the size of the Matrix.
- length is a function that return the length of the array.
- Uniform is a generation which takes as input two floating point numbers and produces a RandFloat in between the two numbers
- UniformInt takes two integers as input and produces a RandInt uniformly in between the two.
- ... (more to come)

11. Example Programs

Here’s a program which computes the expected average squared distance between neighbors in a random permutation of numbers between 1 to n.

\[ \text{Generator RandomPermutation (len : Int) : RandInt } [ ] \{ \]
\[ \text{arr = IntArray(len);} \]
\[ i = 0; \]
\[ \text{while (i < len)} \{ \]
\[ \text{arr[i] = i;} \]
\[ i = i + i; \]
\[ \} \]
\[ i = 0; \]
while (i < len) {
    ind = UniformInt(i, len -1);
    tmp = arr[ind];
    arr[ind] = arr[i];
    arr[i] = tmp;
}
return arr;
}

Stat MeanDistance(arr:RandInt[]) {
    dis = 0;
    i = 0;
    while (i < length(arr) - 1) {
        dis = dis + (arr[i] - arr[i+1])**2;
    }
    return E(dis);
}

Simulate (seed = 123, trials = 1000) {
    MeanDistance(RandomPermutation(10));
}