ProbL: Probabilistic Modeling Language

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

June 4, 2015

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 [1] does allow for model specification, but suffers from the same restrictive shortcomings as mentioned before.

2 Language Overview

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

2.1 Built-in Types

Literals	
true,false	boolean literals
-5,3.6,e,π	integer and real valued literals
Primitives	
int	integer
float	double-precision floating point number
bool	variable taking one of two values (true or false).
enum	variable taking one value from a pre-defined set of values.
Collections	
array	one or two dimensional set of primitive types

Table 1: Built-in Types

2.2 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. Other operators are:

!=, ==, <, <=, >, >=	Numerical relational
+, -, *, /	Arithmetic
&&, ||, !	Logical
$\sim$	Sample a value from the distribution on the RHS

Table 2: Operators

2.3 Control Flow

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

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

Table 3: Control Flow

2.4 Built-in Functions

sum()	summation of values, mathematical symbol Σ
prod()	product of values, mathematical symbol Π
sqrt(float f)	square root of f Π
exp(float a, float b)	exponential a^b
norm(float μ, float σ)	normal distribution with mean μ and variance σ
uniform(float a, float b)	uniform distribution between values of a and b
bin(int n, int k, float p)	binomial distribution with n trials, k successes with probability p

Table 4: Built-in Functions

2.5 User Defined Functions

A user is able to define their own distribution through the use of a function. A function can be defined this way:

```
fun function_name (arg1, arg2,...):return_type
{
}
```

3 Typical Use Cases

As mentioned before, graphical models can be used to represent a wide range of statistical models. Therefore, our language should be able to represent, for example, the following use cases:

- Linear regression (as in the sample program below)
- Principle Component Analysis (PCA)
- Latent Dirichlet Allocation (LDA)
- Hidden Markov Models (HMM's)

Of course, the strength of ProbL is in the ability it gives its users to use the above as building blocks and construct more complex models, and represent user-defined statistical dependencies.

4 Sample Program: 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;
}
model {
        y ~ norm(a + b * x, sigma);
}
```

References

[1] David J Lunn, Andrew Thomas, Nicky Best, and David Spiegelhalter. Winbugs-a bayesian modelling framework: concepts, structure, and extensibility. *Statistics and computing*, 10(4):325–337, 2000.