Plan for Language Specification

Dhruv Rajan

June 8, 2017

1 Figaro's Representation

1.1 Primitive Types

In Figaro, every data structure is an *element*. Every element holds a *value*, with some associated *value type*. Thus, all elements are of the form `Element[type]`. Distinctions are made between four categories of elements:

1. *atomic* self contained, does not depend on another element (`Normal`)
2. *compound* build out of other elements (`If (...), Apply (...)`)
3. *discrete* element whose value type is discrete (`Poisson`)
4. *continuous* element whose value type is continuous (`Gamma`)

1.2 Observing Variables

Elements are defined by distributions of their possible values. A simple `Normal` element can only hold values on the interval $[-1, 1]$, with corresponding probabilities. Thus, each element may be considered a random variable, on its given distribution. Figaro provides methodology for symbolic manipulation of these random variables, allowing dependent variables (compound elements) to be utilized.

No element is given an immediate value until it is observed. The `observe()` method alters only the element on which it is called, so that when an inference algorithm (such as `VariableElimination.probability`) is run on that

element, or any variable in the same dependency network, it can calculate the relevant conditional probabilities.

1.3 Creating Compound Elements

Atomic elements can be combined to form compound elements. The simplest method for combining them is the `If` construct, which allows for slightly complex conditioning on distributions

1.3.1 If

The type signature for `If` is as follows:

```
If (test:  Element[Boolean],
    then:  Element[T],
    else:  Element[T])
   => cond:  Element[T]
```

1.3.2 Apply

The type signature for `apply` is as follows:

```
Apply (test:  Element[T],
       fn:  T -> U)
```