Figaro™

a probabilistic modeling language

charles river analytics

CAPABILITIES

Figaro™ is a powerful language for probabilistic modeling. It provides the ability to use programming language constructs, such as data structures, conditionals, looping, recursion, and abstraction to build models of rich complex systems. Figaro's built-in reasoning algorithms automatically apply to models written in the language, so modelers can experiment with models immediately without having to write reasoning code. The extensive library of model elements enables quickly combining the right components to build models. Figaro also integrates tightly with Java and Scala applications. As a result, Figaro is ideal both for rapid prototyping of probabilistic models and integrating them with applications.

ARCHITECTURE

- **Figaro** is implemented as a library in Scala, an object-oriented and functional language that is interoperable with Java and compiles to the Java virtual machine.
- **Figaro** defines an extensible library of model element classes, which are the building blocks for models. Elements can be combined and manipulated from within a Scala program.
- **Figaro** elements can be constrained by any Scala function.
- **Figaro** expresses a wide variety of probabilistic graphical models, including directed and undirected models, models involving inter-related objects, models involving discrete and continuous elements, and open universe models in which we do not know what or how many objects exist.

```
class Person {
  val smokes = Flip(0.6)
}

val alice, bob, clara = new Person
val friends = List((alice, bob), (bob, clara))
clara.smokes.observe(true)

def smokingInfluence(pair: (Boolean, Boolean)) =
  if (pair._1 == pair._2) 3.0; else 1.0

for { (p1, p2) <- friends } {
  ^^(p1.smokes, p2.smokes).constraint = smokingInfluence
}
```

- **Figaro** provides a number of built-in reasoning algorithms for computing probabilities, such as variable elimination, importance sampling, and Metropolis-Hastings. Algorithms have both one-shot and anytime versions.
- **Figaro** enables sets of elements to be grouped in a universe. This feature enables the expression of dynamic models, which can be reasoned with using particle filtering.

Probabilistic modeling is a core activity at Charles River Analytics. They can be applied in a wide variety of domains, such as data and information fusion, intelligence analysis, and cyber security. Over the years, we have been creating probabilistic models for an increasing number of applications.

As their range of applications grows wide, our probabilistic models grow in richness and diversity. Models may be hierarchical, relational, spatio-temporal, recursive or infinite, among others. Developing the representation, inference, and learning algorithms for new models is a significant task. Probabilistic programming has the potential to make this task much easier by allowing you to represent rich, complex probabilistic models using the full power of programming languages, including data structures, control mechanisms, functions and abstraction. The language provides inference and learning algorithms so you do not need to implement them. Most importantly, a probabilistic programming language (PPL) provides the language and tools with which to think of and formulate new models for complex domains.

```
class Room {
  val color = Dist(0.7 -> 'red, 0.3 -> 'green)
  val large = Flip(0.4)
}
abstract class Apartment {
  def marysRoom: Model[Room]
  def samsRoom: Model[Room]
  // Mary will tend to choose a large room
  marysRoom.addConstraint(room => if (room.large) 1.0; else 0.3)
  // Sam will tend to choose a green room
  samsRoom.addConstraint(room => if (room.color == 'green) 1.0; else 0.5)
  // If Mary's room is large, she likes it, otherwise she likes it with 20% probability
  def sizeOk: Model[Boolean] = Or(Apply(marysRoom, room => room.large), Flip(0.2))
  // If Sam's room is green, she likes it, otherwise she likes it with 50% probability
  def colorOk: Model[Boolean] = Or(Eq(Apply(samsRoom, _.color), 'green), Flip(0.5))
  // Mary and Sam like not to share a room, but if they do share they like it with 30% probability
  def sharingOk: Model[Boolean] = Or(Not(Eq(marysRoom, samsRoom)), Flip(0.3))
  def maryLikes = And(sizeOk, colorOk, sharingOk)
}
class OneRoomApartment extends Apartment {
  val room = new Room
  val marysRoom = room
  val samsRoom = room
}
class TwoRoomApartment extends Apartment {
  val room1 = new Room
  val room2 = new Room
  val assignment = Dist(0.5 -> (room1, room2), 0.5 -> (room2, room1))
  val marysRoom = assignment._1
  val samsRoom = assignment._2
}
val apartment = new TwoRoomApartment
// What is the probability that Mary likes the apartment given that room 1 is green?
apartment.room1.color.observe('green)
val algorithm = VariableElimination(apartment.maryLikes)
algorithm.start()
println(algorithm.probability(maryLikes, true))
```

To this point, most of the research on PPLs has focused on improving the power of a language, both in terms of what it can represent and in terms of the algorithms it uses to reason about models written in the language. Little research has focused on usability of the languages. Ultimately, the goal of PPL research is to provide languages that can be used by a wider audience, not just experts in probabilistic programming. This motivated us to develop Figaro, a PPL that is designed to be both extremely powerful and very usable.

An issue we found in other PPLs was that models were self-contained units. In contrast, we designed Figaro to be a language with composable elements that could be constructed and manipulated from within ordinary programs. This design decision is essential for integrating Figaro models into our applications. In particular, we chose Scala for its interoperability with the widely-used Java language and its support for functional programming, which is essential to probabilistic programming.

Charles River Analytics Inc. (www.cra.com)
625 Mount Auburn Street, Cambridge, MA 02138
For more information, please contact figaro@cra.com

charles river analytics