The Semantics of Figaro, an Embedded Probabilistic Programming Language

Avi Pfeffer
Charles River Analytics
apfeffer@cra.com

Brian Ruttenberg
Charles River Analytics
bruttenberg@cra.com

The goal of this paper is to show how semantics can be defined for a real-world probabilistic programming language. Figaro has had numerous applications in areas such as malware analysis, climate modeling, vehicle health maintenance, and intelligence analysis.

1 Basic Figaro

Figaro is a relatively mature probabilistic programming language embedded as a library in Scala. This embedding has several advantages, including ease of integration into applications and the ability to use arbitrary Scala functions in a Figaro model. However, the Scala embedding makes it harder to analyze programs and understand their behavior. In this abstract, we describe how we define the semantics of Figaro.

The central data structure in Figaro is a Scala class Element[T], which represents a random process that produces a value in the value type T. The main constructs for Element[T] in Figaro's library are:

- Primitive elements such as Flip, which define a probability distribution over values.
- Apply$(a_1, ..., a_n, f)$, where a_i is an Element$[T_i]$ and f is a Scala function from $T_1 \times ... \times T_n$ to T. This represents the process of generating values from a_1 to a_n and applying the function f to the result.
- Chain(a, f), where a is an Element$[U]$ and f is a Scala function $U \to$ Element$[T]$. This represents the process of generating a value x from a, computing an element $e = f(x)$, and producing the outcome from e.

Since the function f in the definition of Apply and Chain is an arbitrary Scala function, it could potentially cause side effects, which would make the language very hard to understand. We require that f terminate on all inputs with no side effects.

2 Abstracting Away Scala Functions

The best way to understand Figaro is to treat the Scala functions as black boxes; once the Scala code has been abstracted away, the Figaro model has a simpler structure that doesn't require evaluating Scala functions. To define this structure, we create a language called SimplePPL, as follows.

The central concept in SimplePPL is a program, which contains a sequence of random variables (RVs). Each RV r has a type T_r. A program Q has a set of *free* variables $\mathcal{F}_Q$, and consists of a sequence of definitions of the form $r =$ expression. The set of RVs defined by Q is denoted $\mathcal{RV}_Q$. An RV is *available* if it is either in $\mathcal{F}_Q$ or defined previously in Q. The set of available RVs with respect to an RV r is denoted $\mathcal{A}_r$.

An expression defining an RV r is one of the following:

- A primitive, which defines a probability distribution over values.
- $A(r_1, \ldots, r_n, g)$, where $r_1, \ldots, r_n$ are available RVs and g is a function (in the mathematical sense) $T_{r_1} \times ... \times T_{r_n} \to T_r$.
- $C(r_1, g)$, where r_1 is an available RV and f is a function $T_{r_1} \to \mathcal{Q}$, where $\mathcal{Q}$ is the space of programs such that for each $Q' \in \mathcal{Q}$, $\mathcal{F}_{Q'} \subseteq \mathcal{A}_r$ and the final RV in the program has type T_r.

Figaro programs can be converted into SimplePPL programs naturally using a converter H, defined as follows.

- For a Figaro primitive d, $H(p)$ is the program $r = d$, where r is a unique RV.
- For a Figaro element $e =$ Apply$(a_1, ..., a_n, f)$, we define $H(e)$ as follows. First, for each a_i, we create the SimplePPL program $H(a_i)$. We then add the definition $r = A(r_1, ..., r_n, g)$, where r is a unique RV, r_i is the final RV in the program $H(a_i)$, and the mathematical function g is obtained by computing, for each combination of arguments $(x_1, ..., x_n)$, the value produced by the Scala function f, which by assumption terminates with a deterministic value on all inputs.
- For a Figaro element $e =$ Chain(a_1, f), we create the SimplePPL program $H(a_1)$ with final RV r_1.

We then add the definition $r = C(r_1, g)$, where r is a unique RV, and $g(x) = H(f(x))$.

3 SimplePPL Semantics

If the SimplePPL program is finite, the semantics is straightforward. However, the converter H could recurse infinitely. Therefore, we define the semantics by assigning a probability to assignments of values to prefixes of the SimplePPL program.

To specify the value of a prefix, we create a special value $*$ for chain RVs that are not expanded. We augment every type T_r into a new type $S_r = T_r \cup \{*\}$.

Definition 1. *A* level-n expansion *of a program Q is a set $V_n(Q)$ defined as follows:*

- $V_0(Q) = \mathcal{RV}_Q$.
- $V_n(Q) = V_{n-1}(Q) \bigcup_{Q'} \mathcal{RV}_{\mathcal{U}(Q')}$, *where Q' ranges over programs $g(x)$ for some $C(r_1, g)$ in $V_{n-1}(Q)$ and $x \in T_{r_1}$. $\mathcal{U}(Q')$ indicates that we use a unique copy of the program Q', because the RVs resulting from different applications of g to x are distinct.*

Definition 2. *The* level-n partial expansions *of a program Q are a set $\mathbf{W}_n(Q)$ of subsets of V_n, defined as follows:*

- $\mathbf{W}_0(Q) = \{V_0(Q), \emptyset\}$.
- $\mathbf{W}_n = \{W : W = W_{n-1} \bigcup_{Q'} W_{Q'}\}$, *where $W_{n-1} \in \mathbf{W}_{n-1}(Q)$, Q' ranges over programs $g(x)$ for some $C(r_1, g) \in W_{n-1}$ and $x \in T_{r_1}$, and $W_{Q'} \in \mathbf{W}_0(Q')$.*

The partial expansions of a program Q are the set $\mathbf{W}(Q) = \bigcup_{n=1}^{\infty} \mathbf{W}_n(Q)$.

Intuitively, a partial expansion includes variables defined in Q up to a bounded depth. The definition ensures that if the variables defined within a chain function for r are included in a partial expansion, so is r, and also that the expansion includes all or none of the RVs in $\mathcal{RV}_Q$ for any Q.

Definition 3. *A* partial assignment *X for a program Q with partial expansion W is an assignment of a value $x \in S_r$ for all r in W, such that*

- *If r is a primitive, $X(r) \neq *$.*
- *If r is $A(r_1, ..., r_n, g)$ and any $X(r_i) = *$, $X(r) = *$.*
- *If r is $A(r_1, ..., r_n, g)$ and all $X(r_i) \in T_{r_i}$, $X(r) = g(X(r_1), ..., X(r_n))$.*
- *If r is $C(r_1, g)$ and $X(r_1) = *$, $X(r) = *$.*
- *If r is $C(r_1, g)$, $X(r_1) = v_1 \in T_{r_1}$, and W does not include $\mathcal{RV}_{g(v_1)}$, $X(r) = *$.*
- *If r is $C(r_1, g)$, $X(r_1) = v_1 \in T_{r_1}$, and W does include $\mathcal{RV}_{g(v_1)}$, $X(r) = X(r')$, where r' is the final RV in $g(v_1)$.*

From the definition of partial assignment, we see that it is fully determined by the structure of W and the values of primitives. We assign a probability to the partial assignment X by the probability of all the primitive assignments in X:

$$P(X) = \prod_{r \in W : r \text{ is defined by a primitive } d_r} d_r(X(r))$$

Definition 4. *The* complete expansion *of Q is $V(Q) = \bigcup_{n=1}^{\infty} V_n(Q)$. A* complete assignment *for Q is a function X that assigns a value in T_r for every $r \in V(Q)$.*

A partial assignment can be viewed as a set of complete assignments, where a complete assignment X_V is in the partial assignment X_W if, for $r \in W$, $X_W(r) = X_V(r)$ or $X_W(r) = *$. Two partial assignments are consistent if for every RV in common, either they have the same value or one of them is $*$. Partial assignments are closed under finite intersection. We extend P to a probability measure over the space of complete assignments with the σ-algebra generated by partial assignments. We need to make sure that P is additive. This can be shown using the following proposition, which says that the probability of a partial assignment is the sum of probabilities of its "one-step" expansions.

Proposition 1. *Let X be a partial assignment for Q with partial expansion W. Let $r = C(r_1, g) \in W$ be such that W does not contain $\mathcal{RV}_{Q'}$, where $Q' = g(X(r_1))$. Let $W' = W \cup \mathcal{RV}_{Q'}$. Then $P(X) = \sum_{X'} P(X')$, where X' ranges over partial assignments for Q with partial expansion W' consistent with X.*

Proof: Any X' must agree with X on all primitive RVs in W, as well as assigning a value to primitive RVs in $\mathcal{RV}_{Q'}$. Therefore

$$\begin{aligned} P(X') &= \prod_{r \in W} d_r(X(r)) \prod_{r \in \mathcal{RV}_{Q'}} d_r(X(r)) \\ &= P(X) \prod_{r \in \mathcal{RV}_{Q'}} d_r(X(r)) \end{aligned}$$

where the multiplication only includes RVs r defined by a primitive d_r. Since the values of primitive RVs and the structure W' uniquely determine X', X' ranges over all possible assignments of values to primitive RVs in $\mathcal{RV}_{Q'}$. The result follows by the law of total probability. □

Acknowledgements

This work was supported by DARPA contract FA8750-14-C-0011.