BFiT: From Possible-World Semantics to Random-Evaluation Semantics in an Open Universe

Yi Wu†, Lei Li*, Stuart Russell†
† EECS, University of California, Berkeley * Baidu Research

Goals & Solution

Goals

- Allow *BLOG* to benefit from progress in *Figaro's* inference engine
- Understand how to code open-universe models efficiently in an embedded, functional PPL
- Understand relationship between possible-world and random-evaluation semantics in practice

Solution: We developed a compiler ***BFiT*** with ***dynamic memoization*** techniques to translate a *BLOG* program to a *Figaro* program with the same output result.

Background

Possible-World Semantics: a program with PW semantics defines a probability measure over possible worlds. [**BLOG**,MLNs,BUGS]

Random-Evaluation Semantics: a program with RE semantics defines a probability measure over execution traces or partial traces [IBAL,Church,**Figaro**]

Open Universe Probability Models: OUPMs model uncertainties in the existence and identity of objects and the relations among them

NIPS 2014, Montreal, Quebec, Canada
Workshop on Probabilistic Programming

OUPM Example & Challenge

Person-Login Model (BLOG)

```
type Person, Login;

#Person ~ Poisson(5);                          // Number Statement

random Boolean  Honest(Person x)               // Random Function
  ~ BooleanDistrib(0.9);

origin Person Owner(Login);                    // Origin Function

#Login(Owner = x) ~ if Honest(x)               // Number Statement w/ Origin Function
  then 1 else Poisson(10);

random Login A
  ~ UniformChoice
  ( {x for Login x});
query Honest(Owner(A));
```

Challenges: efficient *data structure* design for ***Number Statement*** in ***Scala***

Evaluation

Theorem: *BFiT* always produces a target code in Figaro from a BLOG model with a constant blowup factor in program size.

Experiment : Lines of Code in Blog & Figaro

Model	Loc. in BLOG	Loc. in Figaro	Model	Loc. in BLOG	Loc. in Figaro
CSI	14	71	**Urnball**	38	126
Burglary	22	87	**Citation**	40	178
Hurricane	37	168	**TugWar**	55	278

Claim: *BLOG* models OUPMs more concisely.

Dynamic Memorization by BFiT

Translated Program by BFiT

```
val N_Person = Poisson(5);
val Persons =
  MakeList(N_Person,()=>Select(1.0->new Person));
```

Create a ***list*** to store all the persons

```
var MEMO_Honest = Map[Person,Element[Boolean]]();
def Honest(x:Person):Element[Boolean]={
if (_MEMO_Honest.contains(x))
    return _MEMO_Honest(x);
  val ret = Flip(0.9);
  MEMO_Honest += x -> ret;
return ret; };
```

Use a ***Map*** to ***memoize*** each random function

```
class Login(ORIGIN_Owner:Person)
{ val Owner = ORIGIN_Owner; };
```

Represent ***Origin Function*** as a ***field***

```
class Same (Owner:Person){
 val n = If(Honest(Owner),Constant(1),Poisson(10));
 val L = MakeList(n,Select(1.0->new Login(Owner)));
};
```

Create a list of *logins* with the **same *Owner***

```
def create_Logins(L:List[Person]):List[Same] =
{var ret = new ListBuffer[Same];
 for(l<-L) ret+=new Same(l); ret.toList};
class All_Login{
  lazy val A = Apply(Persons, create_Logins
  lazy val total = Chain(A,(A:List[Same])
=>{ val B = Inject(A.map(_.L):_*);
    Apply(B,(B:List[List[Login]])=>B.flatten)});};
lazy val Logins =  Chain(
  Select(1.0->new All_Login),(b:All_Login)=>b.total);
lazy val N_Login =
  Apply(AI_Login,(L:List[Login])=>L.length);
```

Combine lists to a *new List*

Proposed New Syntax

```
class Person extends BaseClass("Person");
class Login extends BaseClass("Login");
CreateObj[Person](Poisson(5));
MakeOrigin[Person,Login]("Owner",(p:Person)
 => If(Honest(p),Constant(1),Poisson(10)));
```