Probabilistic Modeling Using BLOG

Lei Li

University of California, Berkeley

7/14/14

2014 PPAML Summer School, Portland, OR
AI: intelligent systems in the real world

The world has things in it!!

first-order logic

Stuart Russell: “Unifying Logic and AI”
Why did AI choose first-order logic?

- Provides a *declarative* substrate
  - Learn facts, rules from observation and communication
  - Combine and reuse in arbitrary ways
- *Expressive* enough for *general-purpose* intelligence
  - It provides *concise models*, essential for learning
  - E.g., rules of chess (32 pieces, 64 squares, ~100 moves)
    - ~100 000 000 000 000 000 000 000 000 000 000 000 pages
      as a state-to-state transition matrix (cf HMMs, automata)
    - WhiteKingOnC4@Move12
      - 1 page in first-order logic
      \[
      \forall x,y,t,\text{color},\text{piece} \: \text{On}(\text{color},\text{piece},x,y,t) \iff \ldots
      \]
AI: intelligent systems in the real world

The world has things in it!!

first-order logic

Stuart Russell: “Unifying Logic and AI”

The world is uncertain!!

probabilistic models
Modern AI

propositional logic + probability = Probabilistic Graphical Models
AI: intelligent systems in the real world

The world has things in it!!

The world is uncertain!!

first-order logic

probabilistic models

first order probabilistic models
New Dawn of AI

The world is uncertain and has things in it! But we do not know what they are

first order logic + Probability = Bayesian Logic
“At last, artificial intelligences are thinking along human lines.”
"At last, artificial intelligences are thinking along human lines."

“A technique [that] combines the logical underpinnings of the old AI with the power of statistics and probability ... is finally starting to disperse the fog of the long AI winter.”
Bayesian Logic (BLOG)

• As a logic: provide expressive power for representing and reasoning about real world objects with uncertainty
• As a programming language: declare the generative model but not the inference; the generic inference engine will figure out answers automatically
Outline

Part I:
1. First crash in BLOG: a running example
2. How to write a BLOG program

Part II:
3. Semantics of a BLOG program
4. Inference algorithms

Part III:
5. Debugging BLOG program
6. Extending BLOG
Tug of War

Q: who will win?
Tug of War

• Two teams play in each match
• A team consists of players randomly drawn from all candidates
• Each Player has strength value
• During match, players can be lazy -- pulling power shrinks
• Team with greater total pulling power wins
random Real strength(Person p) ~ Gaussian(10, 2);

random Boolean lazy(Person p, Match m)
  ~ BooleanDistrib(0.1);

random Real pulling_power(Person p, Match m) ~
  if lazy(p, m) then strength(p) / 2.0
  else strength(p);
random Boolean team1win(Match m) ~
    sum([pulling_power(p, m) for Person p in team1(m)])
> sum([pulling_power(p, m) for Person p in team2(m)]) ;
Evidence

\[
\text{obs \ team1(M0) = \{James, David\};}
\]

\[
\text{obs \ team2(M0) = \{Brian, John\};}
\]

\[
\text{obs \ team1win(M0) = true;}
\]

\[
\text{obs \ team1(M1) = \{James, David\};}
\]

\[
\text{obs \ team2(M1) = \{Bob, Andrew\};}
\]
Is James stronger than Brian?

query strength(James) > strength(Brian);

======== LW Trial Stats ========
Samples done: 1000000.  Time elapsed: 641.246 s.
Fraction of consistent worlds (running avg, all trials): 0.49
======== Query Results ========
Iteration: 1000000
Probability of (strength(James) > strength(Brian)) is 0.678
Who is winning in the new match?

\[
\text{obs } \text{team1}(M100) = \{\text{James, David}\}; \\
\text{obs } \text{team2}(M100) = \{\text{Bob, Andrew}\}; \\
\text{query } \text{team1win}(M100); \\
\]

====== LW Trial Stats ======
Samples done: 1000000.    Time elapsed: 641.246 s.
Fraction of consistent worlds (running avg, all trials): 0.49

====== Query Results ======
Iteration: 1000000

**Probability of team1win(M100) is 0.586**
What if diff. strength prior?

random Real strength(Person p) ~ Gaussian(10, 2);
random Real strength(Person p) ~ UniformReal(0, 5);
random Boolean lazy(Person p, Match m) ~ BooleanDistrib(0.1);
random Real pulling_power(Person p, Match m) ~
  if lazy(p, m) then strength(p) / 2.0
  else strength(p);
query strength(James) > strength(Brian);
query team1win(M100);

======== Query Results ========
Iteration: 1000000
Probability of (strength(James) > strength(Brian)) is 0.737
Probability of team1win(M100) is 0.651
Outline

Part I:
1. First crash in BLOG: a running example
2. How to write a BLOG program

Part II:
3. Semantics of a BLOG program
4. Inference algorithms

Part III:
5. Debugging BLOG program
6. Extending BLOG
BLOG Program

- List of statements separated by ;
- Types
- Distinct symbols
- Dependency statements
  - expression
- Number statement and origin function
- Evidence
- Query
A Gaussian random variable

random Real $x \sim \text{Gaussian}(0.5, 1.0)$;
A Bernoulli r.v.

\[
\text{random Integer } z \sim \text{Bernoulli}(0.5);
\]

syntax (more general version later)

\[
\text{random type\_name function\_name } \sim \text{expression}
\]
Expressions

- Literals: e.g. 1, 2.5, true, false
- Logic variable reference
- Operators: e.g. a + b
- Function Application (function call):
  abs( x )
- Distribution
- Quantified Formula
- Set Comprehension
- Map
- Conditionals
Built-in Types and Literals

• Boolean, true, false
• Integer, 1, 2, 3, ...
• Real, (IEEE745 double precision floating point)
• String, “hello world!”
• null
## BLOG distributions in the library

<table>
<thead>
<tr>
<th>Bernoulli</th>
<th>Geometric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beta</td>
<td>Laplace</td>
</tr>
<tr>
<td>Binomial</td>
<td>Multinomial</td>
</tr>
<tr>
<td>BooleanDistrib</td>
<td>MultivarGaussian</td>
</tr>
<tr>
<td>Categorical</td>
<td>NegativeBinomial</td>
</tr>
<tr>
<td>Dirichlet</td>
<td>Poisson</td>
</tr>
<tr>
<td>Discrete</td>
<td>UniformChoice</td>
</tr>
<tr>
<td>Exponential</td>
<td>UniformInt</td>
</tr>
<tr>
<td>Gamma</td>
<td>UniformReal</td>
</tr>
<tr>
<td>Gaussian</td>
<td>UniformVector</td>
</tr>
</tbody>
</table>
Operator expression

random Real x ~
  if z == 1 then Gaussian(0.5, 1.0)
  else Gaussian(-0.5, 1.0);

z == 1

z equals 1?
## Supported Operators

<table>
<thead>
<tr>
<th>precedence</th>
<th>interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>unary minus</td>
</tr>
<tr>
<td>%</td>
<td>mod</td>
</tr>
<tr>
<td>^</td>
<td>power</td>
</tr>
<tr>
<td>*, /</td>
<td></td>
</tr>
<tr>
<td>+, -</td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>negation</td>
</tr>
<tr>
<td>&amp;</td>
<td>and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;, &lt;, &gt;=, &lt;=, ==, !=</td>
<td></td>
</tr>
<tr>
<td>=&gt;</td>
<td>imply</td>
</tr>
</tbody>
</table>
Conditional expression

random Real x ~
    if z == 1 then Gaussian(0.5, 1.0)
    else Gaussian(-0.5, 1.0);

if boolean-expression then expression
    else expression
Case-in expression

random Real x ~
  case z in {  
    1 -> Gaussian(0.5, 1.0),
    2 -> Gaussian(-0.5, 1.0)
  };

  case cond_expression in map_expression;

  map_expression ::= {  
    left_expression -> right_expression
    <$> left_expression -> right_expression …>
  }

if z is 1, draw from Gaussian(0.5, 1)
if z is 2, draw from Gaussian(-0.5, 1)
**Map and Categorical**

Categorical\(\{10 \rightarrow 0.2, 20 \rightarrow 0.8\}\)

Categorical takes a Map

\[
\text{map\_expression} ::= \{ \\
\hspace{1em} \text{left\_expression} \rightarrow \text{right\_expression} \\
\hspace{1em} <, \text{left\_expression} \rightarrow \text{right\_expression} \\
\ldots \}
\]

*with 20% chance to get 10, 80% chance to get 20*
State evidence

\[ \text{obs } x = 0.2; \quad \text{observe } x's \text{ value } 0.2 \]

\[ \text{obs } \text{expression} = \text{expression}; \]

\text{Note: distribution expression not allowed}

\[ \text{obs Gaussian}(0, 1) = 1; \]
Issue a Query

query z;

what is the distribution of values of z?

query expression;

Note: distribution expression not allowed

query Gaussian(0, 1);
random Integer z ~ Bernoulli(0.5);
random Real x ~
    if z == 1 then Gaussian(0.5, 1.0)
    else Gaussian(-0.5, 1.0);
obs x = 0.2;
query z;
Enhancing Gaussian Mixture Models

• Support multiple observations
  Need more general random functions
Random Functions

```plaintext
random Real x(Integer i) ~
    if z == 1 then Gaussian(0.5, 1.0)
    else Gaussian(-0.5, 1.0);

random type_name func_symbol(Arg_type logical_var) ~ expression;
```

- state the relation btw func_symbol and expression
- must return distribution
Making Multiple Observations

\[
\text{obs } x(0) = 0.2;
\]
\[
\text{obs } x(1) = 1.0;
\]
\[
\text{obs } x(3) = 0.5;
\]
\[
\text{obs } x(4) = 0.6;
\]
Gaussian Native Bayes Model

random Integer \( z \) \( \sim \) Bernoulli(0.5);
random Real \( x(\text{Integer } i) \) \( \sim \)
  if \( z = 1 \) then Gaussian(0.5, 1.0)
  else Gaussian(-0.5, 1.0);
obst\( x(0) = 0.2; \)
obst\( x(1) = 1.0; \)
obst\( x(3) = 0.5; \)
obst\( x(4) = 0.6; \)
query \( z; \)
Gaussian Mixture Model (again)

random Integer $z(i) \sim \text{Bernoulli}(0.5)$;
random Real $x(i) \sim$
    if $z(i) == 1$ then Gaussian(0.5, 1.0)
    else Gaussian(-0.5, 1.0);
obs $x(0) = 0.2$;
obs $x(1) = 1.0$;
obs $x(3) = 0.5$;
obs $x(4) = 0.6$;
Gaussian Mixture Model – unknown mixture weight

random Real p ~ Beta(0.5, 1);
random Integer z(Integer i) ~ Bernoulli(p);
random Real x(Integer i) ~
  if z(i) == 1 then Gaussian(0.5, 1.0)
  else Gaussian(-0.5, 1.0);
obs x(0) = 0.2;
obs x(1) = 1.0;
obs x(3) = 0.5;
obs x(4) = 0.6;
query round(p * 10.0) / 10.0; // for bucketing
Gaussian Mixture Model – unknown component location

random Real p ~ Beta(0.5, 1);
random Integer z(Integer i) ~ Bernoulli(p);
random Real a ~ UniformReal(-1, 1);
random Real b ~ UniformReal(-1, 1);
random Real x(Integer i) ~
  if z(i) == 1 then Gaussian(a, 1.0)
  else Gaussian(b, 1.0);
obs x(0) = 0.2;
obs x(1) = 1.0;
obs x(3) = 0.5;
obs x(4) = 0.6;
query round(min({a, b}) * 10.0) / 10.0;
What if # Components not known? (Nonparametric Mixture Model)

- Need:
  - User defined type
  - Number statement (Open-universe)
  - Set comprehension
  - UniformChoice from Set
User defined type

type Component;
Number statement

type Component;
#Component ~ Poisson(2);

number of components follows Poission(2)

#type_name ~ expression;
Set comprehension expression

type Component;

\{c \text{ for } \text{Component } c\} \quad \text{refer to the set of all components}

\{\text{expression } \text{for } \text{type}_1 \text{name}_1 \ \text{var}_1, \text{type}_2 \text{name}_2 \ \text{var}_2 \ldots : \text{boolean expression}\ \}

\{p \text{ for } \text{Person } p: \ \text{age}(p) < 30\}

\text{the set of Person whose age is less than 30}
Explicit Set

\{expression1, expression2, \ldots\}

\{1, 2, 3\}
UniformChoice from a Set

type Component;
#Component ~ Poisson(2);
random Component z(Integer i) ~
  UniformChoice({c for Component c});

*randomly choosing from all components*
What if # Components not known?  
(Nonparametric Mixture Model)

type Component;

# Component ~ Poisson(2);

random Component z(Integer i) ~ UniformChoice({c for Component c});

random Real mean(Component c) ~ UniformReal(-1, 1);

random Real x(Integer i) ~ Gaussian(mean(z(i)), 1.0);

obs x(0) = 0.2;
obs x(1) = 1.0;
obs x(3) = 0.5;
obs x(4) = 0.6;

query size({c for Component c});

But there is a bug in the code... what if # Component = 0?
What if # Components not known?  
(Nonparametric Mixture Model)

type Component;
#Component ~ Poisson(2);
random Component z(Integer i) ~ UniformChoice({c for Component c});
random Real mean(Component c) ~ UniformReal(-1, 1);
random Real x(Integer i) ~
  if z(i) != null then Gaussian(mean(z(i)), 1.0);
obs x(0) = 0.2;
obs x(1) = 1.0;
obs x(3) = 0.5;
obs x(4) = 0.6;
query size({c for Component c});
Tug of War

type Person;
type Match;
distinct Person James, David,
    Jason, Brian, Mary, Nancy, Susan, Karen;
distinct Match M[4];

eight Person
four matches

These symbols are guaranteed to be different from each other.

distinct symbol definition
Tug of War

type Person;
type Match;
distinct Person James, David, Jason, Brian, Mary, Nancy, Susan, Karen;
distinct Match M[4];
random Real strength(Person p) ~ Gaussian(10, 2);
Tug of War: constructing teams

random Person team1player1(Match m)
  ~ UniformChoice({p for Person p});
random Person team1player2(Match m)
  ~ UniformChoice({p for Person p : p !=
team1player1(m)});
random Person team2player1(Match m)
  ~ UniformChoice({p for Person p : (p !=
team1player1(m)) & (p != team1player2(m))});
random Person team2player2(Match m)
  ~ UniformChoice({p for Person p : (p !=
team1player1(m)) & (p != team1player2(m))
  & (p != team2player1(m))});
random Boolean lazy(Person p, Match m) ~ BooleanDistrib(0.1);
random Real pulling_power(Person p, Match m) ~
    if lazy(p, m) then strength(p) / 2.0
    else strength(p);
random Boolean team1win(Match m) ~
    if (pulling_power(team1player1(m), m) + pulling_power(team1player2(m), m) > pulling_power(team2player1(m), m) + pulling_power(team2player2(m), m))
    then true
    else false;
Tug of War: evidence and query

obs team1player1(M[0]) = James;
obs team1player2(M[0]) = David;
obs team2player1(M[0]) = Brian;
obs team2player2(M[0]) = Jason;
obs team1player1(M[1]) = James;
obs team1player2(M[1]) = David;
obs team2player1(M[1]) = Mary;
obs team2player2(M[1]) = Nancy;
obs team1player1(M[2]) = James;
obs team1player2(M[2]) = Karen;
obs team1win(M[0]) = true;
query strength(James) > strength(Brian);
query team1win(M[1]);
query team1win(M[2]);

is James stronger than Brian?
is team1 winning in the match?
type Person;
type Match;
distinct Person James, David,
    Jason, Brian, Mary, Nancy, Susan, Karen;
distinct Match M[4];
random Real strength(Person p) ~
    Gaussian(10, 2);
random Person team1player1(Match m)
    ~ UniformChoice({p for Person p});
random Person team1player2(Match m)
    ~ UniformChoice({p for Person p : p !=
                        team1player1(m)});
random Person team2player1(Match m)
    ~ UniformChoice({p for Person p : (p !=
                        team1player1(m)) & (p != team1player2(m))});
random Person team2player2(Match m)
    ~ UniformChoice({p for Person p : (p !=
                        team1player1(m)) & (p != team2player1(m)) &
                        (p != team2player1(m))});
random Boolean lazy(Person p, Match m)
    ~ BooleanDistrib(0.1);
random Real pulling_power(Person p, Match m)
    (team2player1(M[3]) == Mary) &
    (team2player2(M[3]) == Susan);

    if lazy(p, m) then strength(p) / 2.0
    else strength(p);
random Boolean team1win(Match m) ~
    if (pulling_power(team1player1(m), m) +
        pulling_power(team1player2(m), m) +
        pulling_power(team2player1(m), m) +
        pulling_power(team2player2(m), m))
    then true
    else false;
obs team1player1(M[0]) = James;
obs team1player2(M[0]) = David;
obs team2player1(M[0]) = Brian;
obs team2player2(M[0]) = Jason;
obs team1player1(M[1]) = James;
obs team1player2(M[1]) = David;
obs team2player1(M[1]) = Mary;
obs team2player2(M[1]) = Nancy;
obs team1player1(M[2]) = James;
obs team2player2(M[2]) = Karen;
obs team1win(M[0]) = true;
query strength(James) > strength(Brian); // is James stronger than Brian?
query team1win(M[1]); // is team1 winning in second match?
query team1win(M[2]); // is team1 winning in third match?
query (!team1win(M[3])) &
query (team2player1(M[3]) == Mary) &
query (team2player2(M[3]) == Susan);
Aircraft Tracking

How many Aircraft? What are likely tracks?
Possible Interpretation

Radar Blips

- t=1
- t=2
- t=3

- t=1
- t=2
- t=3
Aircraft Tracking

Needs

• Symbol evidence
  – observed something but no idea about the identity

• Origin function and general number statement
  – hierarchical generation of objects

• Fixed function
Origin function

type Aircraft;
type Blip;
#Aircraft ~ Poisson(5);

origin Aircraft Source(Blip);
#Blip(Source=a) ~ Bernoulli(0.9);

the number of blips for each aircraft follows Bernoulli with 0.9

origin type_name1 fun_name(type_name2);

#type_name1(fun_name=var_name, ...) ~ expression;
Symbol Evidence

• observing some blips, but no idea about their identity and origin

\[
\text{obs } \{b \text{ for Blip } b\} = \{B1, B2, B3\};
\]

\[
\text{obs Set_comprehension} = \text{explicit_set_symbols};
\]
Fixed function

is x within epsilon range of y?

```plaintext
fixed Boolean inRange(Real x, Real y, Real epsilon) =
  (x > y - epsilon) & (x < y + epsilon);

fixed type_name func_symbol(Arg_type logical_var) = expression;
```

state the relation btw func_symbol and expression
contains no random symbol no Distribution!
<table>
<thead>
<tr>
<th>Built-in Function</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>ones</td>
</tr>
<tr>
<td>pi</td>
<td>abs</td>
</tr>
<tr>
<td>inv</td>
<td>exp</td>
</tr>
<tr>
<td>det</td>
<td>log</td>
</tr>
<tr>
<td>transpose</td>
<td>min</td>
</tr>
<tr>
<td>vstack</td>
<td>max</td>
</tr>
<tr>
<td>hstack</td>
<td>sum</td>
</tr>
<tr>
<td>eye</td>
<td>size</td>
</tr>
<tr>
<td>zeros</td>
<td>iota</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Pull requests are welcome!*
Input

• `loadRealMatrix` to Load a `RealMatrix` from a text file.
• The space-separated formats produced by `numpy` and `Matlab` are supported.
• e.g.

```java
fixed RealMatrix x = loadRealMatrix("data/score.txt");
```
Aircraft Tracking

type Aircraft;
type Blip;
#Aircraft ~ Poisson(5);
origin Aircraft Source(Blip);
#Blip(Source=a) ~ Bernoulli(0.9);
random Real Position(Aircraft a) ~
    UnivarGaussian(0, 10);
random Real ObsPos(Blip b) ~
    UnivarGaussian(Position(Source(b)), 1);
fixed Boolean inRange(Real x, Real y, Real epsilon) =
    (x > y - epsilon) & (x < y + epsilon);
obs {b for Blip b} = {B1, B2, B3};
fixed Real epsilon = 0.05;
obs inRange(ObsPos(B1), 5.0, epsilon) = true;
query size({a for Aircraft a});
Dynamic models

- Built-in type: Timestep

```plaintext
defined RealMatrix A = [1, 1, 0.5; 0, 1, 1; 0, 0, 1];
defined RealMatrix Q = [0.1, 0, 0; 0, 0.1, 0; 0, 0, 0.1];
defined RealMatrix C = [1, 0, 0];
defined RealMatrix mu0 = [0; 1; 1];
random RealMatrix state(Timestep t) ~
  if t == @0 then MultivarGaussian(mu0, Q)
  else MultivarGaussian(A * state(prev(t)), Q);
defined RealMatrix R = [0.1];
random RealMatrix location(Timestep t) ~ MultivarGaussian(C * state(t), R);
obs location(@0) = [0];
obs location(@1) = [0.5];
obs location(@2) = [1];
query state(@1);
query location(@3);
```
Part II: Semantics and Inference
Recall: BLOG Program

• List of statements separated by ;
• Types
• Distinct symbols
• Dependency statements
  – expression
• Number statement and origin function
• Evidence
• Query
Semantics of PPL

• Possible world semantics
  – A BLOG program defines probabilistic measure over all possible model structures
  – Contingent Bayesian Network
  – event = set of full instantiation

• Random execution semantics
  – Church/Figaro define execution traces, and probability measure over these traces
  – event = set of program execution traces
Recall: Aircraft Tracking

type Aircraft;
type Blip;
#Aircraft ~ Poisson(5);
origin Aircraft Source(Blip);
#Blip(Source=a) ~ Bernoulli(0.9);
random Real Position(Aircraft a) ~
    UnivarGaussian(0, 10);
random Real BlipLoc(Blip b) ~
    UnivarGaussian(Position(Source(b)), 1);
fixed Boolean inRange(Real x, Real y, Real epsilon) =
    (x > y - epsilon) & (x < y + epsilon);
obsv {b for Blip b} = {B1, B2, B3};
fixed Real epsilon = 0.05;
obsv inRange(BlipLoc(B1), 5.0, epsilon) = true;
query size({a for Aircraft a});
Semantics – Objects

• Objects indexed by type, origin objects, id
  <Aircraft, 1>, <Aircraft, 2>, ...
  <Blip, Source=<Aircraft, 1>, 1>,
  <Blip, Source=<Aircraft, 1>, 2> ......
  <Blip, Source=<Aircraft, 2>, 1>,
  <Blip, Source=<Aircraft, 2>, 2>,
  ...

• Built-in type has predefined objects
Semantics – basic random variable

• Random Function application variable:
  – function symbol indexed by tuple of objects

    Position_{Aircraft, 1}
    Position_{Aircraft, 2}
    Position_{Aircraft, 3}
    ...

    BlipLoc_{Blip, Aircraft, 1, 1}
    BlipLoc_{Blip, Aircraft, 1, 2} ...
    BlipLoc_{Blip, Aircraft, 2, 1}
    BlipLoc_{Blip, Aircraft, 2, 2}
    ...

Semantics – basic random variable

• Number variable:
  – similar to fun app var, indexed by tuple of origin objects

  #Aircraft
  #Blip_<Aircraft, 1>
  #Blip_<Aircraft, 2>
  ...

Semantics

• distinct symbols
  – i.e. constant zero-ary function
  – can be treated as object instance

• Formula: same as first order logic
Possible world

• Each possible world (w) specifies the values for all basic random variables (including random function application variable and number variable)

• Each basic var is associated with cpd defined in the dependency statement

\[
\text{Position}_{<\text{Aircraft, 1}>} \sim \text{Gaussian}(0, 10)
\]
\[
\text{Position}_{<\text{Aircraft, 2}>} \sim \text{Gaussian}(0, 10)
\]
Probability Measure

of a possible world is determined by product of conditional probabilities of basic random variables

\#Aircraft = 3

Position_{Aircraft, 1} = 2.6
Position_{Aircraft, 2} = 4.2
Position_{Aircraft, 3} = ...

...
Contingent Bayesian Network

Graphical Representation of Dependency
Every well-formed BLOG model specifies a unique proper probability distribution over all possible worlds definable given its vocabulary.

- No infinite receding ancestor chains;
- no conditioned cycles;
- all expressions finitely evaluable;
- Functions of countable sets
  - random Real fun(Real x) not allowed
Outline

Part I:
1. First crash in BLOG: a running example
2. How to write a BLOG program

Part II:
3. Semantics of a BLOG program
4. Inference algorithms

Part III:
5. Debugging BLOG program
6. Extending BLOG
Inference

After observing many evidences, what is “best guess” of a query?

How to answer?

- Exact algorithms (belief propagation/message passing)
- Variational approximation
- Sampling methods (Rejection, MCMC, SMC)
Graphical Model (CBN)

- # aircraft
  - a1, a2, ...

- Loc(a1) → # Blip
  - (a1)

- Loc(a2) → # Blip
  - (a2)

- Loc(...) → # Blip
  - (...)

- BlipLoc(blip(a1,1)) → # Blip
  - # Blip

- BlipLoc(blip(a1,2)) → # Blip

- BlipLoc(blip(a2,1)) → # Blip

- ...
Basic terminology

- **Partial Instantiation (Partial world)**
  - Assignment to random variables, not necessary complete
  - e.g. (#Aircraft=3, Location(A1) = 100, ... )

- **Supported** expression in a partial instantiation
  - Given an expression, can determine its distribution, or
  - Can determine its value
Parental Importance Sampling

1. start from evidence
2. check if the current node is “supported”
   - No, move to first un-instantiated parent, → step 2
   - Yes
     - basic var: propose value from importance distribution
     - derived expression: just calculate values
3. Are evidence “supported”
   - No: move to first unsupported evidence, → step 2
   - Yes: calculate importance weight $\pi(x)/q(x)$, done!
4. Query has value? No → step 2
Likelihood Weighting

$\text{# aircraft} \ a1,a2,...$

$\text{Loc}(a1)$

$\text{Loc}(a2)$

$\text{# Blip (a1)}$

$\text{# Blip (a2)}$

$\text{BlipLoc( blip(a1,1))}$

$\text{BlipLoc( blip(a1,2))}$

$\text{BlipLoc( blip(a2,1))}$

$\text{# Blip}$

$=3$

$\text{Loc}(\ldots)$

$\text{# Blip (\ldots)}$

$\ldots$
Likelihood Weighting

- # aircraft a1, a2, ...
- Loc(a1) → # Blip (a1) → BlipLoc( blip(a1,1))
- Loc(a2) → # Blip (a2) → BlipLoc( blip(a1,2))
- Loc(…)
- # Blip (…)
- BlipLoc( blip(a2,1))

= 3
Likelihood Weighting

# aircraft
a1, a2, ...

Loc(a1) → # Blip (a1) → BlipLoc(blip(a1,1))

Loc(a2) → # Blip (a2) → BlipLoc(blip(a1,2))

Loc(...) → # Blip (...) → BlipLoc(blip(a2,1))

# Blip

= 3
Likelihood Weighting

# aircraft
a1,a2,...

Loc(a1) → # Blip (a1) → BlipLoc(blip(a1,1))

Loc(a2) → # Blip (a2) → BlipLoc(blip(a1,2))

Loc(…) → # Blip (…) → BlipLoc(blip(a2,1))

# Blip

=3
Likelihood Weighting

# aircraft
a1,a2,....

Loc(a1) → # Blip (a1)

Loc(a2) → # Blip (a2)

Loc(...) → # Blip (...)

BlipLoc(blip(a1,1)) → # Blip

BlipLoc(blip(a1,2)) → # Blip

BlipLoc(blip(a2,1)) → # Blip

=3
Likelihood Weighting

# aircraft = 2
a1,a2

Loc(a1) -> # Blip (a1)

Loc(a2) -> # Blip (a2)

Loc(...) -> # Blip (...)

BlipLoc( blip(a1,1)) -> # Blip
BlipLoc( blip(a1,2)) -> # Blip

BlipLoc( blip(a2,1)) -> # Blip

=3
Likelihood Weighting

# aircraft = 2
a1,a2

Loc(a1) → # Blip (a1) → BlipLoc(blip(a1,1))

Loc(a2) → # Blip (a2) → BlipLoc(blip(a2,1))

Loc(...) → # Blip (...) → BlipLoc(blip(...,...))

=3
Likelihood Weighting

# aircraft = 2
a1,a2

Loc(a1) → # Blip (a1) = 2

Loc(a2) → # Blip (a2) = 1

BlipLoc(blip(a1,1)) → # Blip
BlipLoc(blip(a1,2)) → # Blip

BlipLoc(blip(a2,1)) → # Blip

Loc(...) → # Blip (...)

= 3
Likelihood Weighting

# aircraft = 2
a1,a2

Loc(a1) → # Blip (a1) = 2

Loc(a2) → # Blip (a2) = 1

BlipLoc(blip(a1,1))

BlipLoc(blip(a1,2))

# Blip

BlipLoc(blip(a2,1))

Loc(…)

# Blip (…)

…

likelihood/matching evidence? = 3
MCMC for BLOG

• Works for any* model
• No model-specific mathematical work required
• Small space requirement
• Metropolis-Hastings step involves computing the acceptance ratio $\frac{\pi(x')q(x'|x')}{\pi(x)q(x'|x)}$; everything cancels except local changes
• Query evaluation on states is also incrementalizable (cf DB systems)
Switch variable

• variable that is either
  – number variable; or
  – function app var appearing in condition of if-then-else, or case-in
  – e.g. #Aircraft

random Real pulling_power(Person p, Match m) ~
if lazy(p, m) then strength(p) / 2.0
else strength(p);
MCMC for BLOG (Milch et al UAI 2006)

1. Construct an initial consistent partial world

2. Loop
   1. randomly pick a basic variable from partial world
   2. propose a value for the variable
   3. If it is a switch variable, may need to sample additional variables (children and ancestors of children)
   4. accept with ratio $\pi(x')q(x'|x') / \pi(x)q(x'|x)$
MCMC for BLOG

- # aircraft = 2
  - a1, a2, ...

- # Blip = 1 (a1)

- # Blip = 2 (a2)

- # Blip = 3
MCMC for BLOG

# aircraft = 2 → 3
a1, a2, ...

# Blip = 1 (a1)

# Blip = 2 (a2)

# Blip = 3

= 3
MCMC for BLOG

# aircraft=2 \rightarrow 3
a1,a2,...

# Blip = 1 (a1)
# Blip = 2 (a2)
# Blip (...)

# Blip = 3
MCMC for BLOG

# aircraft=2 \rightarrow 3
a_1, a_2, \ldots

# Blip = 1
(a_1)

# Blip = 2
(a_2)

# Blip = 0
(a_3)

# Blip

= 3
Inference

• Theorem: BLOG inference algorithms (rejection sampling, importance sampling, MCMC) converge* to correct posteriors for any well-formed model, for any finitely evaluable first-order query
Efficient inference

• Real applications use special-purpose inference

• DARPA PPAML program is trying several solutions
  – Model-specific code generation reduces overhead
    • BLOG compiler gives 100x-300x speedup
    • Partial evaluator independent of inference algorithm
  – Modular design with “plug-in” expert samplers
    • E.g., sample a parse tree given sentence + PCFG
    • E.g., sample $X_1, \ldots, X_k$ given their sum
  – Data and process parallelism, special-purpose chips
  – Lifted inference
  – Adaptive MCMC proposals
Inference for Dynamic Models

• Sequential Monte Carlo
  – (Bootstrap) Particle Filter
  – Liu-West filter, works better for joint static parameter/variable
  – Stovik Filter, Parameter-linear Gaussian-systems
  – Extended Parameter Filter (Erol et al 2013), works much better for general state-space models continuous static parameters (dynamic variables can be arbitrary)

• Particle MCMC (Andrieu, Doucet, Holenstein)
Part III:
Practical Guide of Using BLOG
Resources

• BLOG website: bayesianlogic.cs.berkeley.edu (hosted on github)
• BLOG Language Reference: version 0.9
• BLOG User manual:
• BLOG User mailing list: https://groups.google.com/d/forum/blog-user
Demo: Trying BLOG web engine

http://patmos.banatao.berkeley.edu:8080/
BLOG software

- Requirement:
  - Java 1.6+
  - (optional) Scala 2.10.4+

- **Universal zip**: http://bayesianlogic.github.io/download/blog-0.9.1.zip

- **Linux debian/ubuntu pre-build package**: http://bayesianlogic.github.io/download/blog-0.9.1.deb

- **Windows installation package**: http://bayesianlogic.github.io/download/blog.msi
Interactive Shell & Debugging

- Open terminal (Mac/Linux)
  
  bin/iblog
Commands

- bin/blog <filename>
  - running main blog engine
- bin/dblog <filename>
  - running particle filtering
Practical Guide to Performance Tuning

• use \(-r\) to set random seed
• consider increase the number of samples/particles \((-n 1000000)\)
• consider MHSampler \((-s blog.sample.MHSampler)\)
• For dynamic model
  – consider PF: use dblog
  – consider Liu-West filter: \(-e blog.engine.LiuWestFilter\)
• use more memory
Output to structured format (GSON)

- machine readable format
- `-o filename`
- `pretty_pretty_print it on screen`
- `https://github.com/BayesianLogic/blog/blob/master/tools/pretty_print_json.py`
Extending BLOG: Custom Distribution

- Java, must implement
  `blog.distrib.CondProbDistrib`
  - `setParams`
  - `sampleVal`
  - `getProb`, `getLogProb`
- BLOG engine will look up distribution classes in the package `blog.distrib`.
- In addition, it will look up distribution classes under the default empty package.
- See `UniformInt` example
Extending BLOG: User Defined Function

- Java, A user-defined function must extend blog.model.AbstractFunctionInterp and provide a constructor that takes a single List argument
- See example in manual
Thanks!

• Contact: leili@cs.berkeley.edu
• BLOG probabilistic programming system at http://bayesianlogic.cs.berkeley.edu/

• TA: Constantin Berzan, Yi Wu
  (also here at PPAML summer school)
Backup