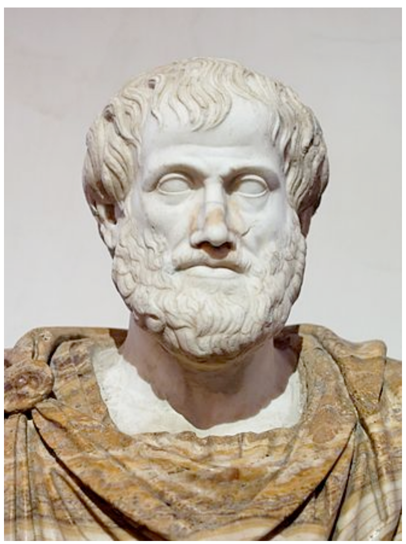
Probabilistic Modeling Using BLOG

Lei Li **University of California, Berkeley**

AI: intelligent systems in the real world

The world has things in it!!

first-order logic



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)
    - ~100 000 000 000 000 000 000 000 000 000 000 000 000 pages as a state-to-state transition matrix (cf HMMs, automata)

    R.B.KB.RPPP..PPP..N..N…..PP….q.pp..Q..n..n..ppp..pppr.b.kb.r

    - ~100 000 pages in propositional logic (cf circuits, graphical models)

    WhiteKingOnC4@Move12

    - 1 page in first-order logic

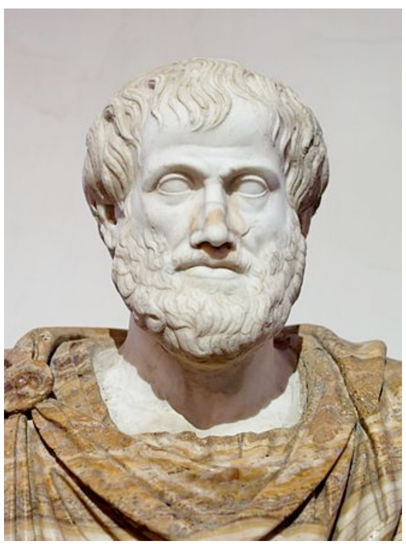
    $\forall x,y,t,color,piece\ On(color,piece,x,y,t) \Leftrightarrow \ldots$

AI: intelligent systems in the real world

The world has things in it!!

The world is uncertain!!

first-order logic

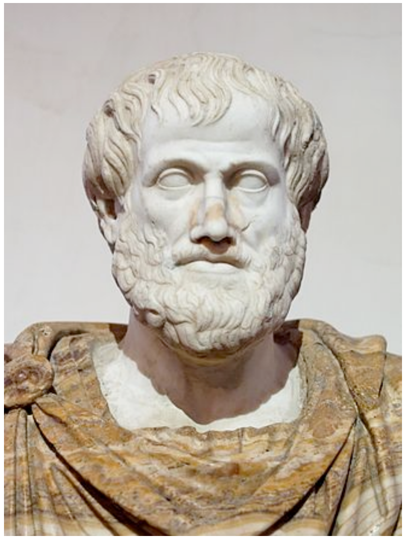


probabilistic models

Stuart Russell: “Unifying Logic and AI”

Modern AI

propositional logic



+

probability

=

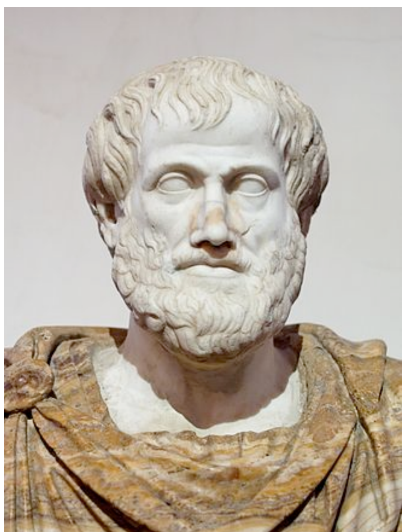
Probabilistic Graphical Models

AI: intelligent systems in the real world

The world has things in it!!

The world is uncertain!!

first-order logic



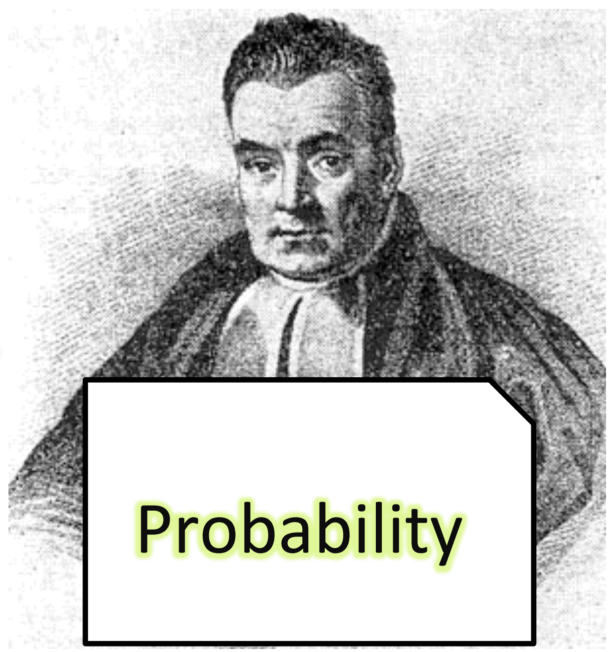
probabilistic models

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

+



=

Bayesian Logic

Anil Ananthaswamy, "*I, Algorithm: A new dawn for AI*,"
New Scientist, Jan 29, 2011

"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:

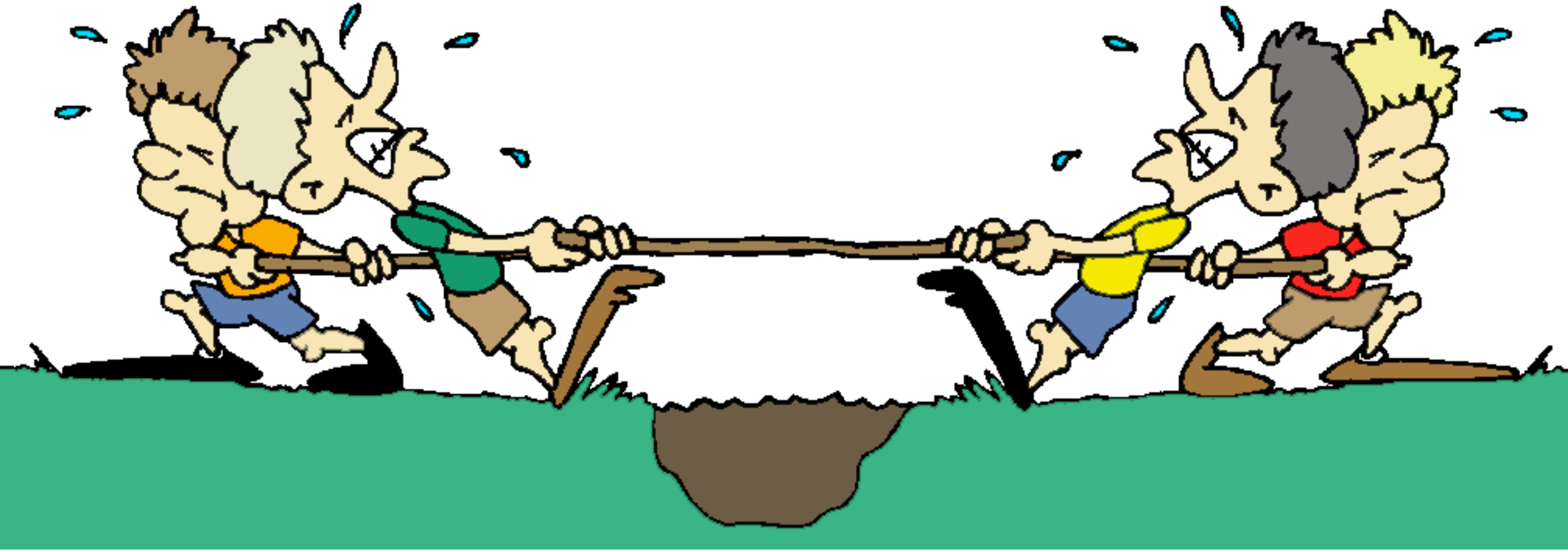
3. Semantics of a BLOG program
4. Inference algorithms

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 draw from all candidates
- Each Player has strength value
- During match, players can be lazy -- pulling power shrinks
- Team with greater total pulling power wins

Pulling power

```
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);
```

Winning team

```
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

```
obs team1(M0) = {James, David};
obs team2(M0) = {Brian, John};
obs team1win(M0) = true;

obs team1(M1) = {James, David};
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?

```
obs team1(M100) = {James, David};
obs team2(M100) = {Bob, Andrew};

query 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);
```

Just One line change!!

Query again

```
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 ~ Gaussian(0.5, 1.0);
```

defining random symbols

Type

function symbol

Parameters of distribution

Distribution name

A Bernoulli r.v.

```
random Integer z ~ Bernoulli(0.5);
```

syntax (more general version later)

```
random type_name function_name ~
  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

Bernoulli	Geometric
Beta	Laplace
Binomial	Multinomial
BooleanDistrib	MultivarGaussian
Categorical	NegativeBinomial
Dirichlet	Poisson
Discrete	UniformChoice
Exponential	UniformInt
Gamma	UniformReal
Gaussian	UniformVector

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

precedence	interpretation
-	unary minus
%	mod
^	power
*, /	
+, -	
!	negation
&	and
|	or
>, <, >=, <=, ==, !=	
=>	imply

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)
   };
```

if z is 1, draw from Gaussian(0.5, 1)

if z is 2, draw from Gaussian(-0.5, 1)

```
case cond_expression in map_expression;
```

```
map_expression ::= {
   left_expression ->  right_expression
   <, left_expression ->  right_expression …>
}
```

Map and Categorical

```
Categorical({10 -> 0.2, 20 -> 0.8})
```

Categorical takes a Map

with 20% chance to get 10,
80% chance to get 20

```
map_expression ::= {
  left_expression ->  right_expression
  <, left_expression ->  right_expression
…>
}
```

State evidence

```
obs x = 0.2;
```

observe x's value 0.2

```
obs expression = expression;
```

Note: distribution expression not allowed

~~**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);**~~

Putting together:
Gaussian Mixture Model

```
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

```
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

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

Gaussian Native Bayes Model

```
random Integer z ~ Bernoulli(0.5);
random Real x(Integer i) ~
  if z == 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 z;
```

Gaussian Mixture Model (again)

```
random Integer z(Integer i) ~ Bernoulli(0.5);
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;
```

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 for Component c}
```

refer to the set of all components

```
{expression for type_name1 var_name1,
type_name2 var_name2…: boolean_expression }
```

```
{p for Person p: age(p) < 30}
```

the set of Person whose age is less than 30

Explicit Set

```
{expression1, expression2, …}
```

```
{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))});
```

Tug of War: match result

```
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

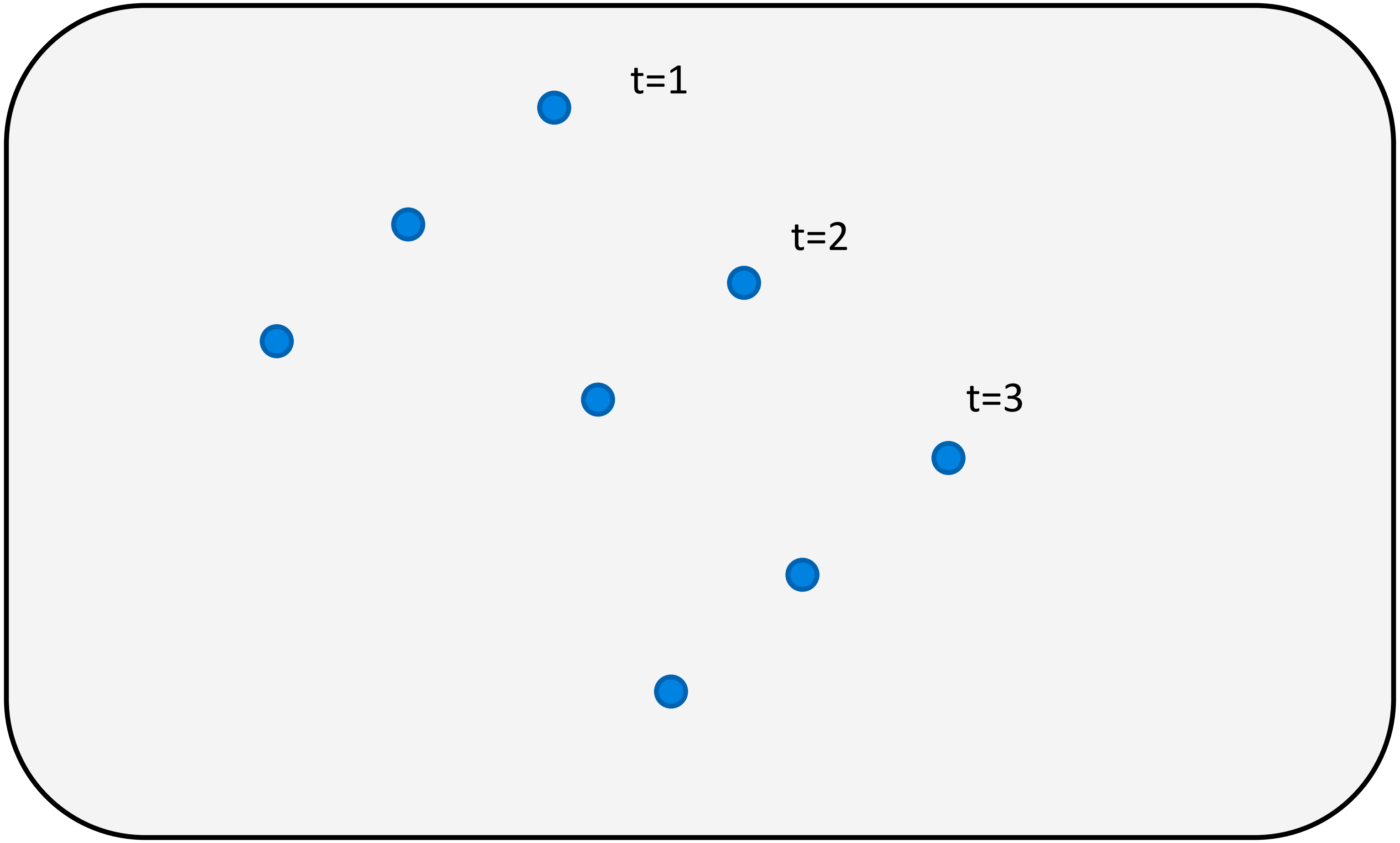
```
obs team1player1(M[0]) = James;
obs team1player2(M[0]) = David;
obs team2player1(M[0]) = Brian;
obs team2player2(M[0]) = Jason;
obs team1player1(M[1]) = James;
obs team1player2(M[1]) = David;
obs team2player1(M[1]) = Mary;
obs team2player2(M[1]) = Nancy;
obs team1player1(M[2]) = James;
obs team1player2(M[2]) = 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?

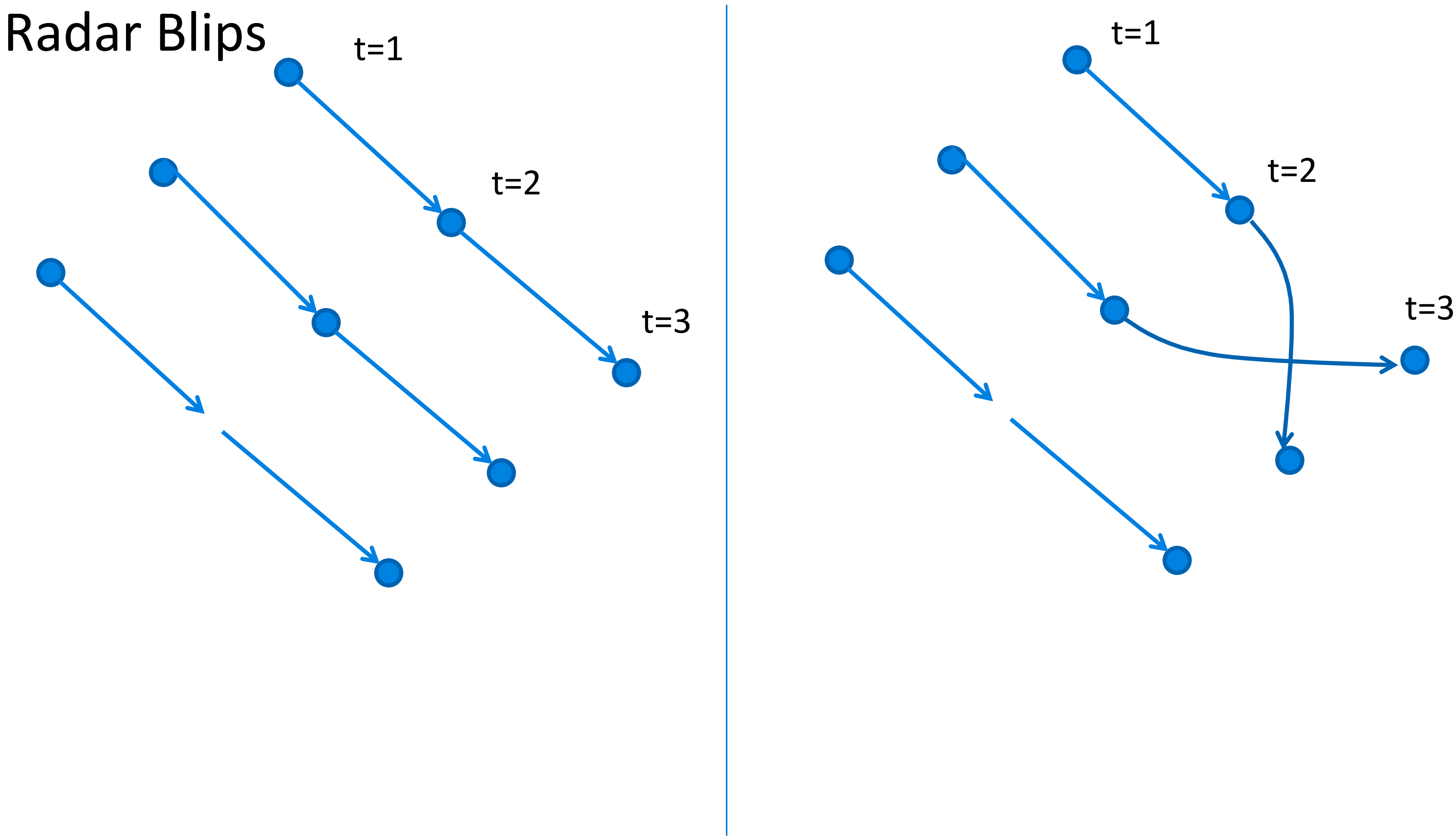
```
type Person;
type Match;
distinct Person James, David,
  Jason, Brian, Mary, Nancy, Susan, Karen;
distinct Match M[4];
random Real strength(Person p) ~
Gaussian(10, 2);
random Person team1player1(Match m)
  ~ UniformChoice({p for Person p});
random Person team1player2(Match m)
  ~ UniformChoice({p 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 != 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;
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); //
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])) &
(team2player1(M[3]) == Mary) &
(team2player2(M[3]) == Susan);
```

Aircraft Tracking



How many Aircraft?   What are likely tracks?

Possible Interpretation



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

```
obs {b for Blip b} = {B1, B2, B3};
```

```
obs Set_comprehension = explicit_set_symbols;
```

Fixed function

is x within epsilon range of y?

```
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!

Built-in Function

e	ones	toInt
pi	abs	toReal
inv	exp	sin
det	log	cos
transpose	min	tan
vstack	max	atan2
hstack	sum	prev
eye	size	next
zeros	iota	loadRealMatrix
		isEmptyString

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.

```
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

```
fixed RealMatrix A = [1, 1, 0.5; 0, 1, 1; 0, 0, 1];
fixed RealMatrix Q = [0.1, 0, 0; 0, 0.1, 0; 0, 0, 0.1];
fixed RealMatrix C = [1, 0, 0];
fixed RealMatrix mu0 = [0; 1; 1];
random RealMatrix state(Timestep t) ~
  if t == @0 then MultivarGaussian(mu0, Q)
  else MultivarGaussian(A * state(prev(t)), Q);
fixed 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);
obs {b for Blip b} = {B1, B2, B3};
fixed Real epsilon = 0.05;
obs 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>,
…
```

- Builtin 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

```
Position_<Aircraft, 1> ~ Gaussian(0, 10)

Position_<Aircraft, 2> ~ Gaussian(0, 10)
```

Probability Measure

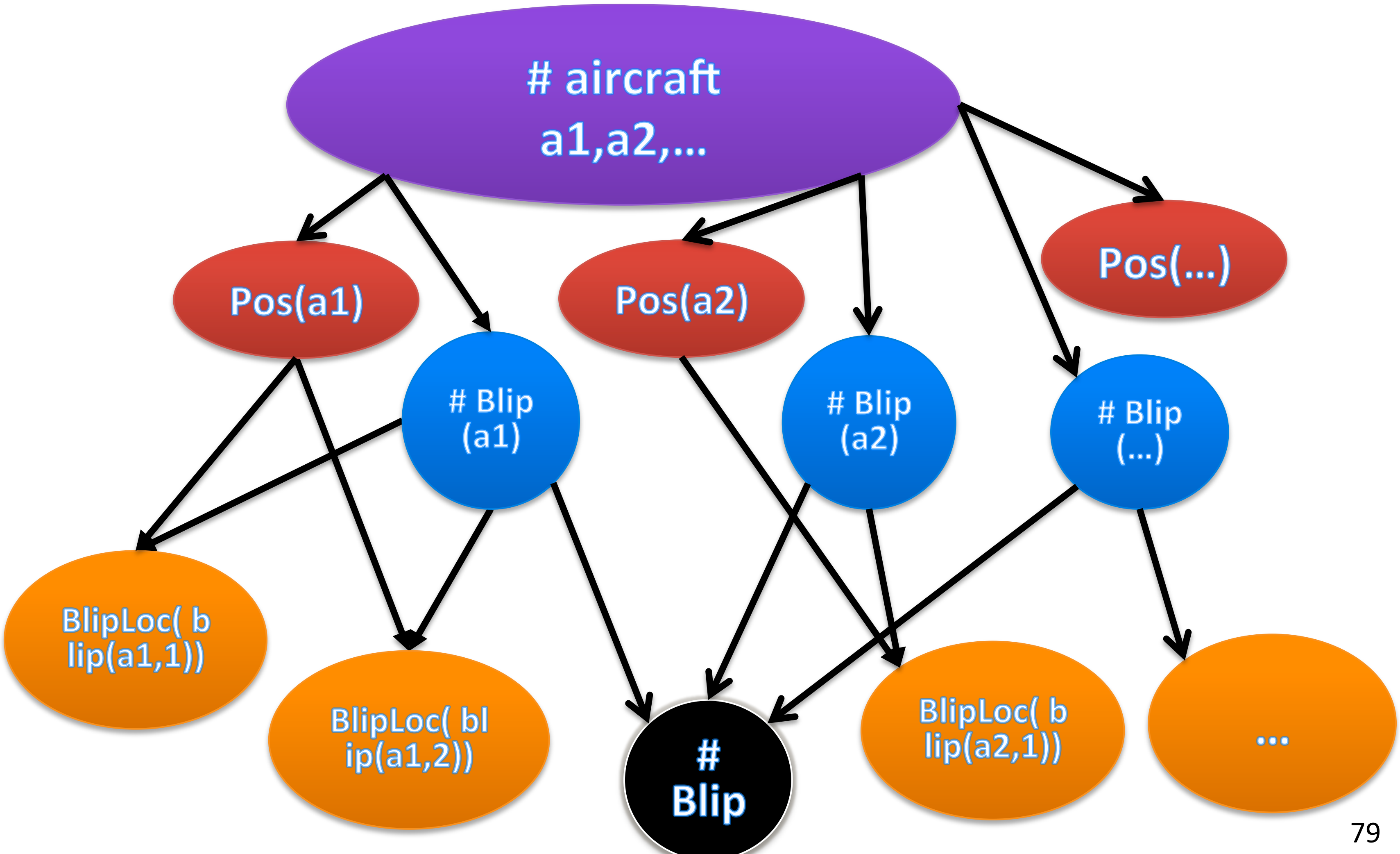
of a possible world is determined by product of conditional probabilities of basic random variables

```
#Aircraft = 3

Position_<Aircraft, 1> = 2.6
Position_<Aircraft, 2> = 4.2
Position_<Aircraft, 3> = …
…
```

Contingent Bayesian Network

Graphical Representation of Dependency



Semantics

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

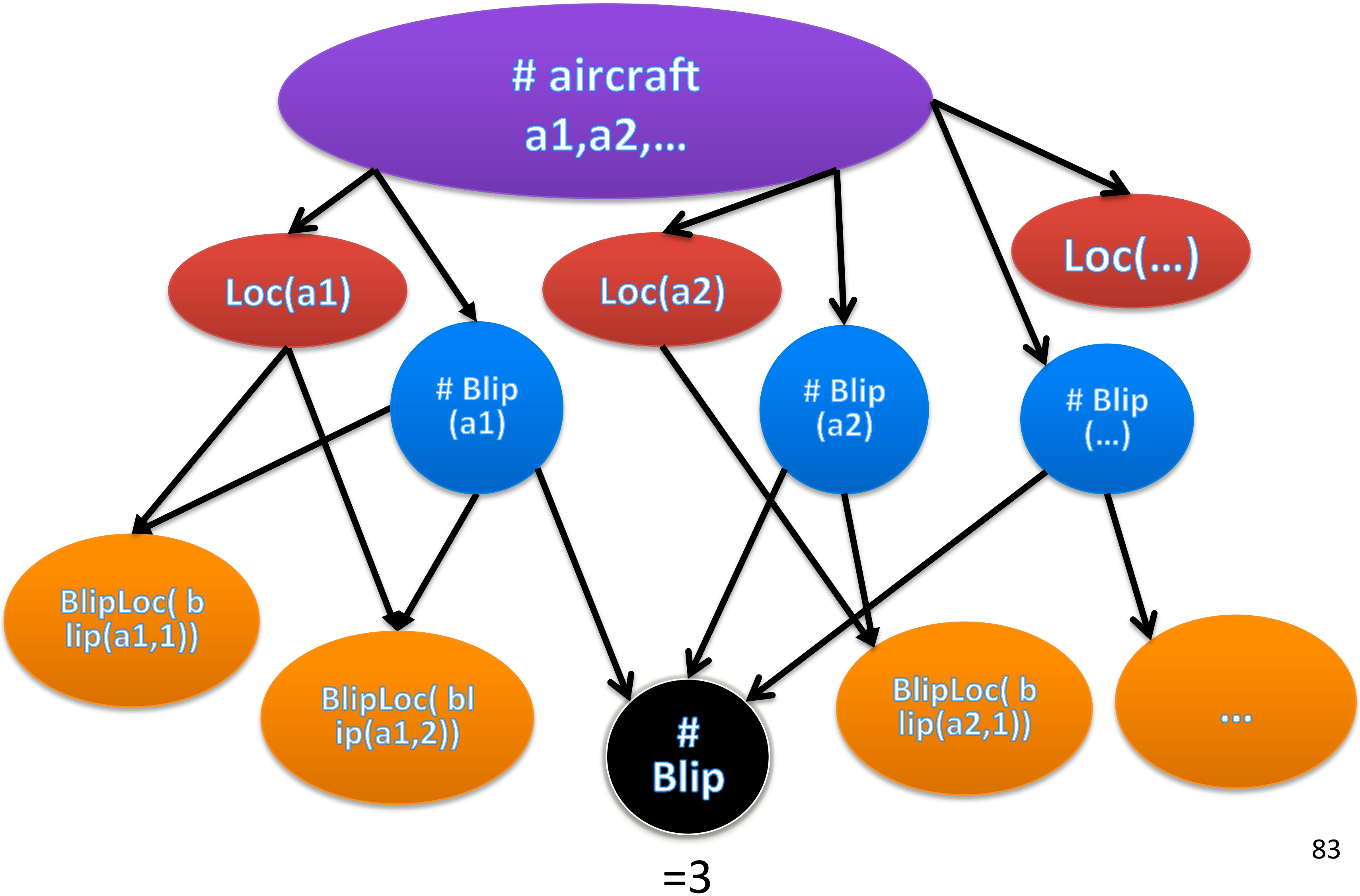
Inference

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)



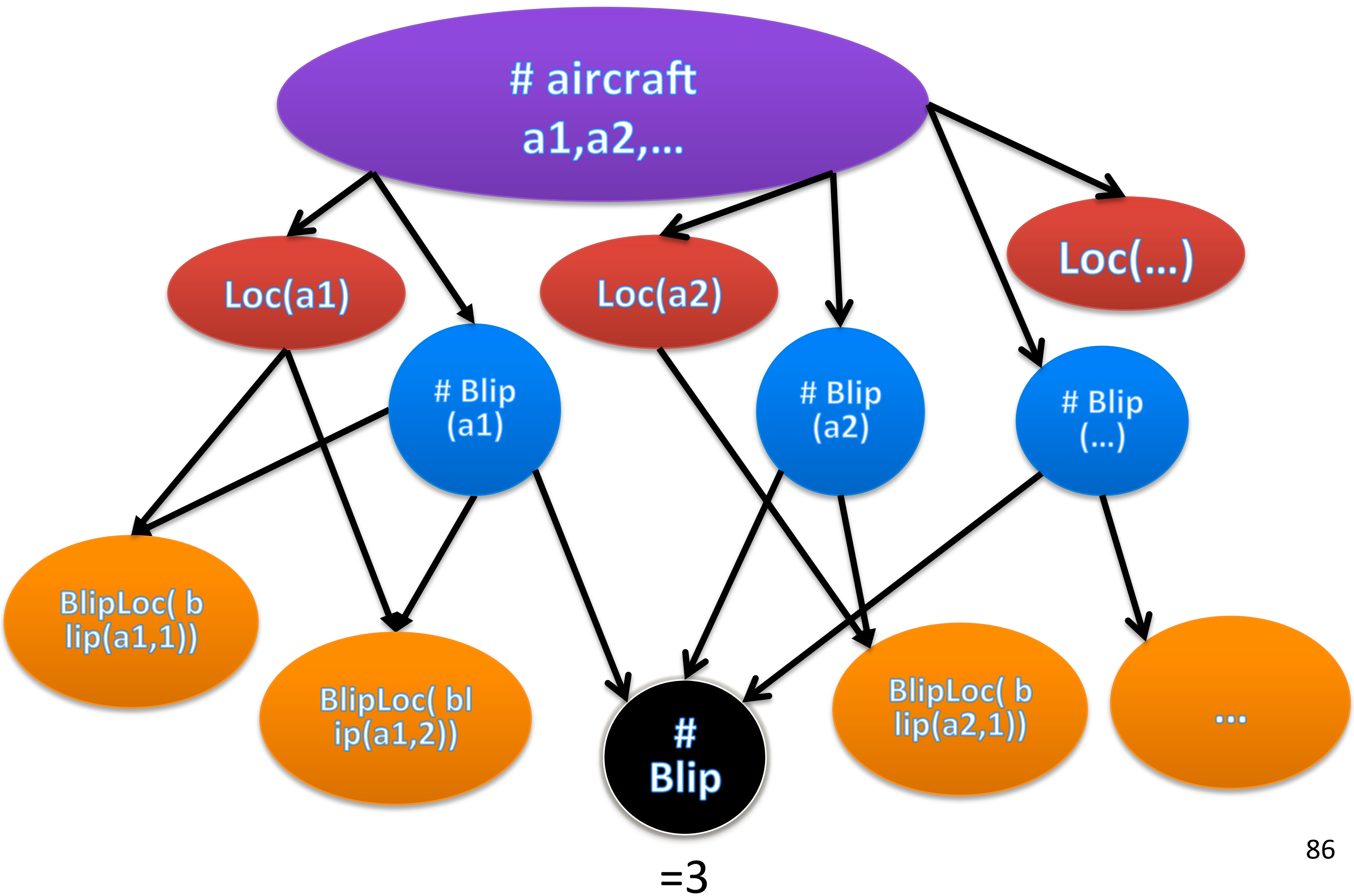
Basic terminology

- *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

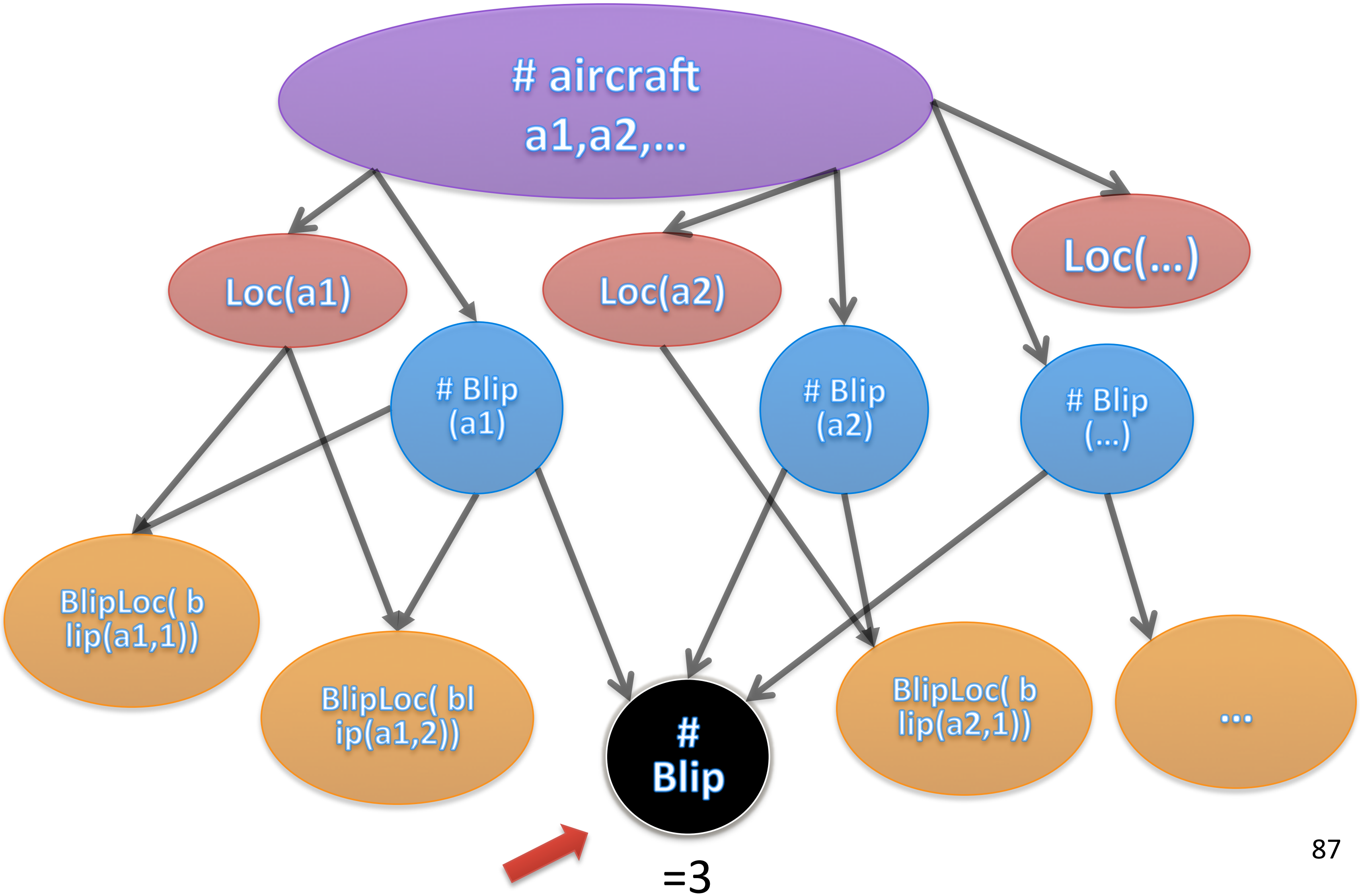
Parental Importance Sampling

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 **π(x)/q(x)**, done!
4. Query has value? No → step 2

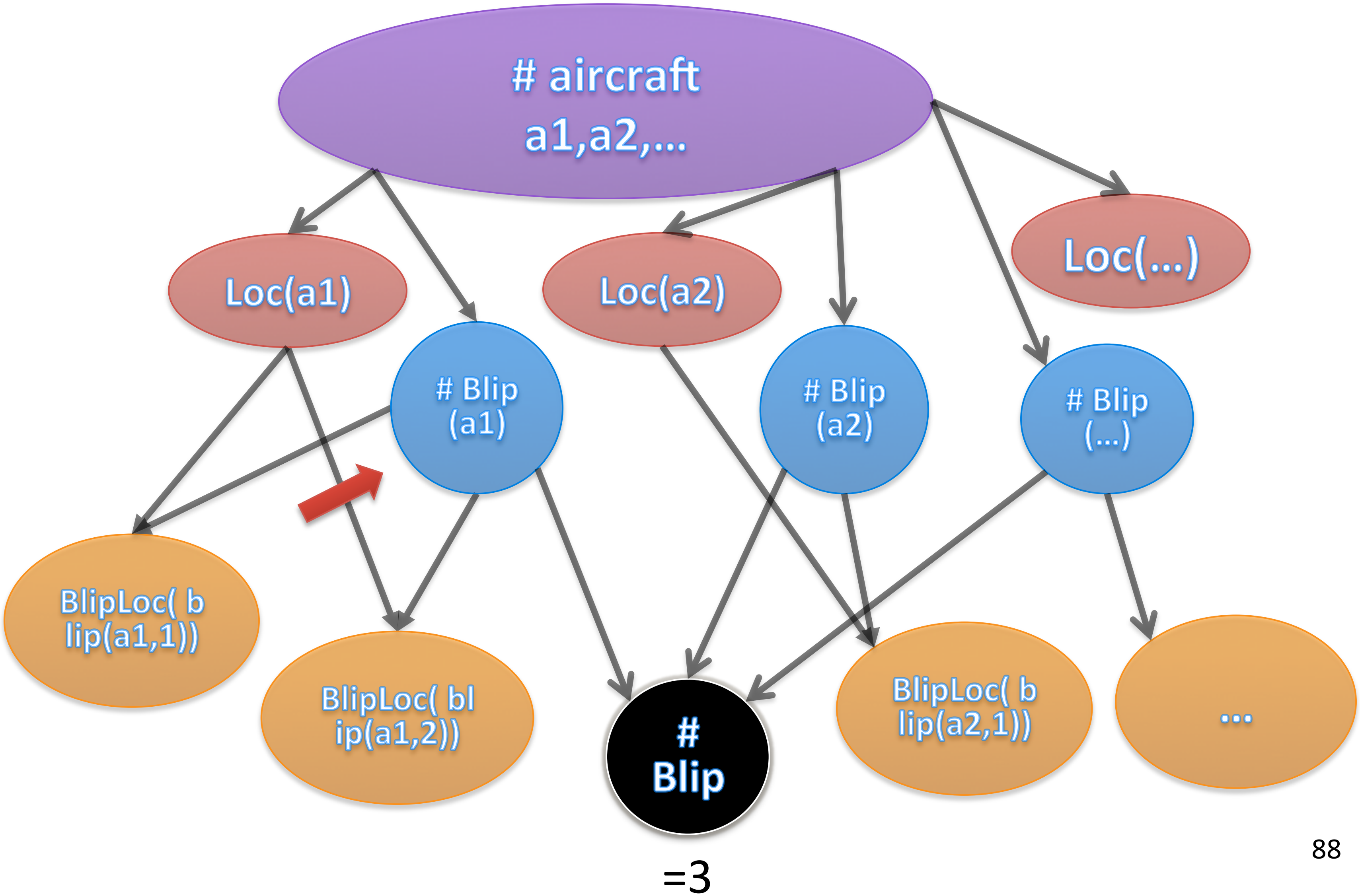
Likelihood Weighting



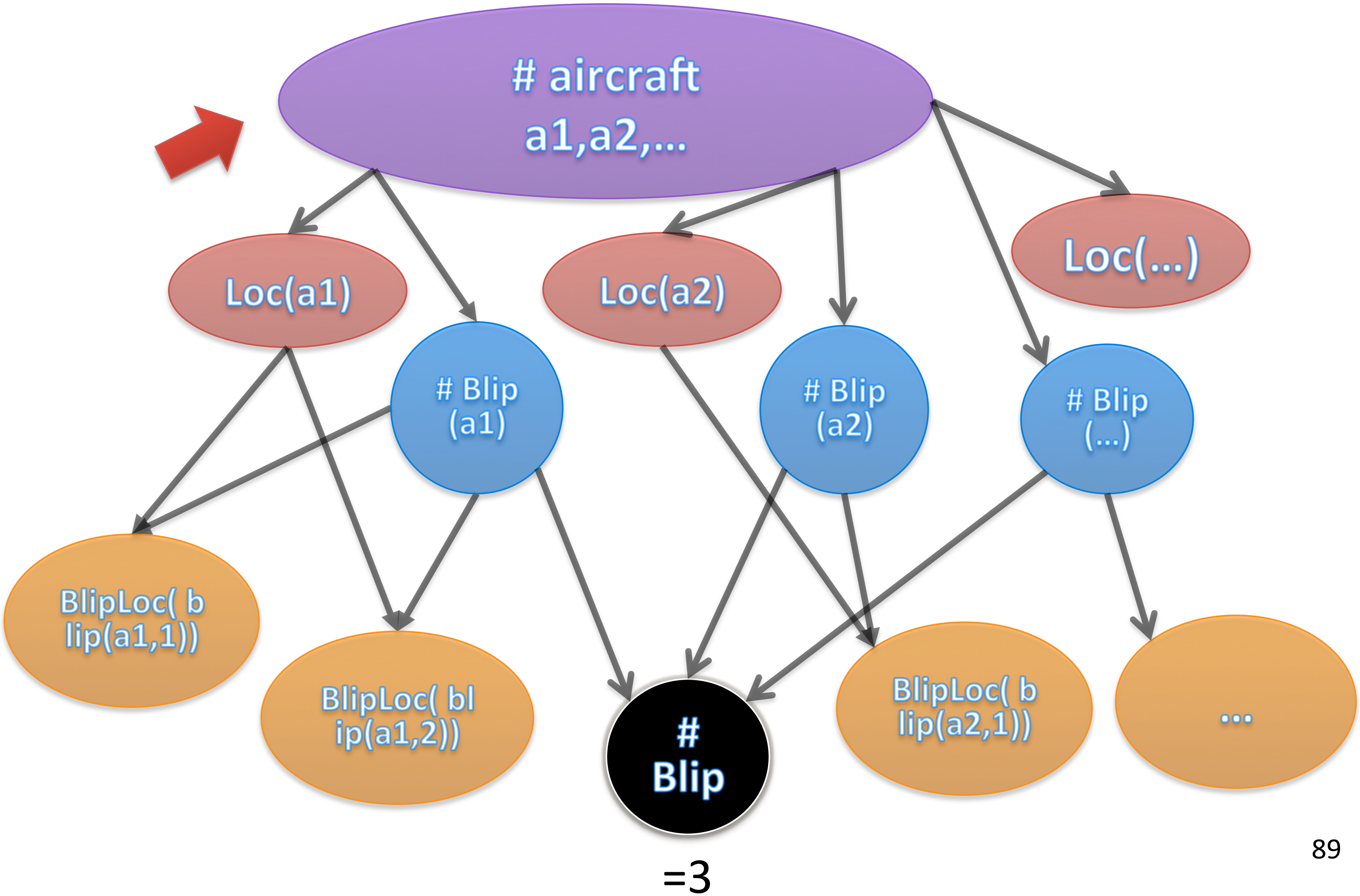
Likelihood Weighting



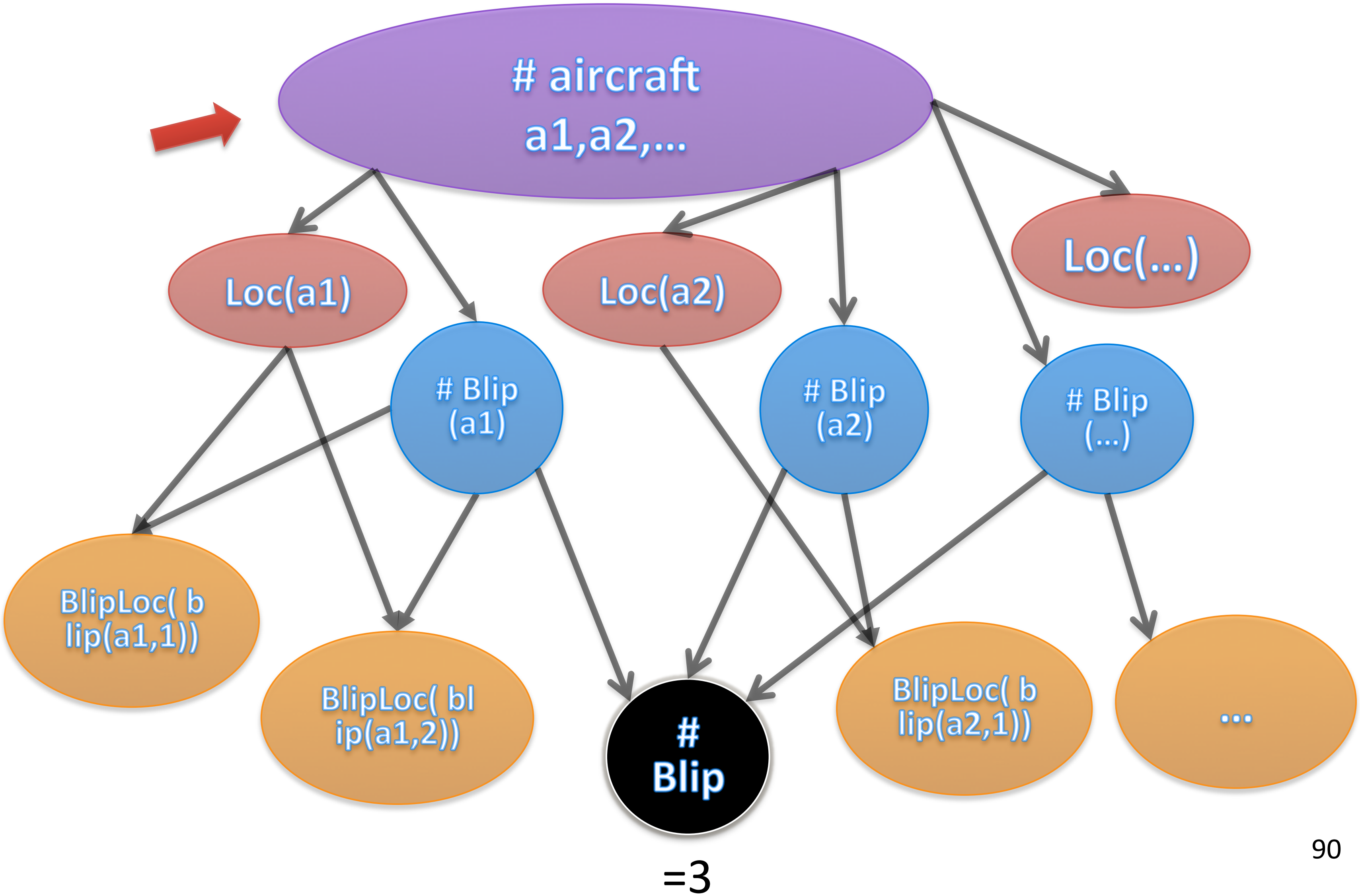
Likelihood Weighting



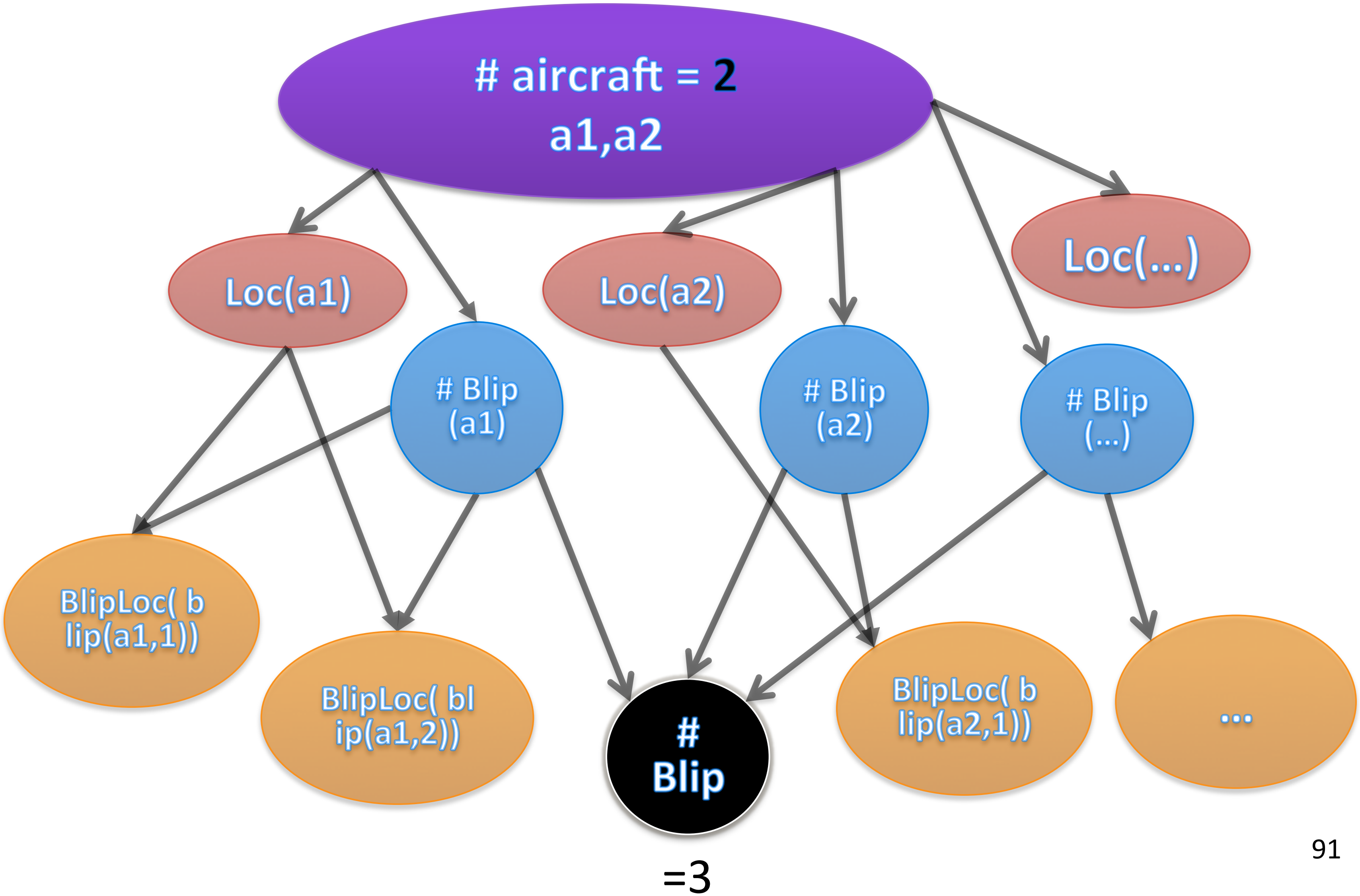
Likelihood Weighting



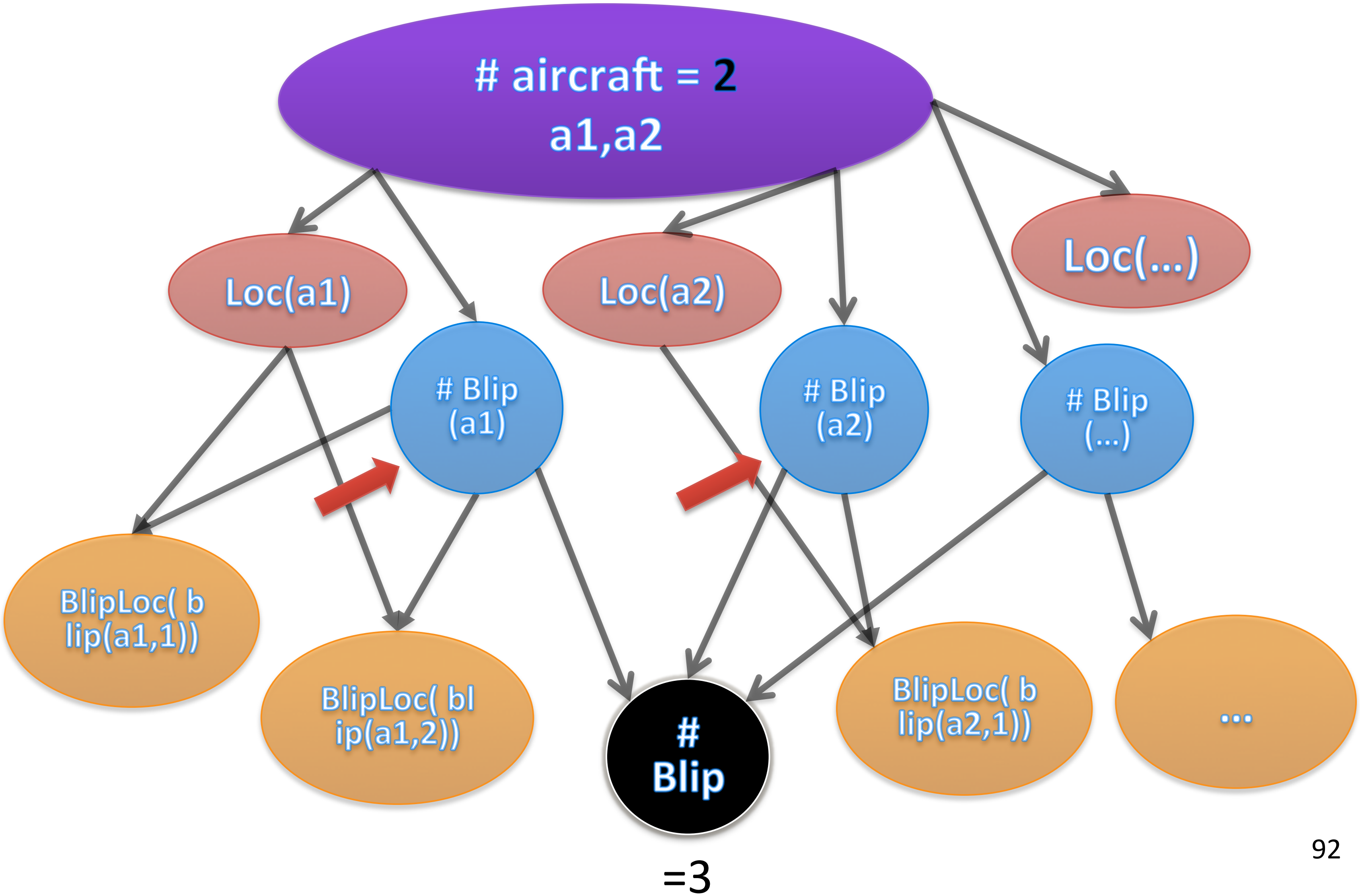
Likelihood Weighting



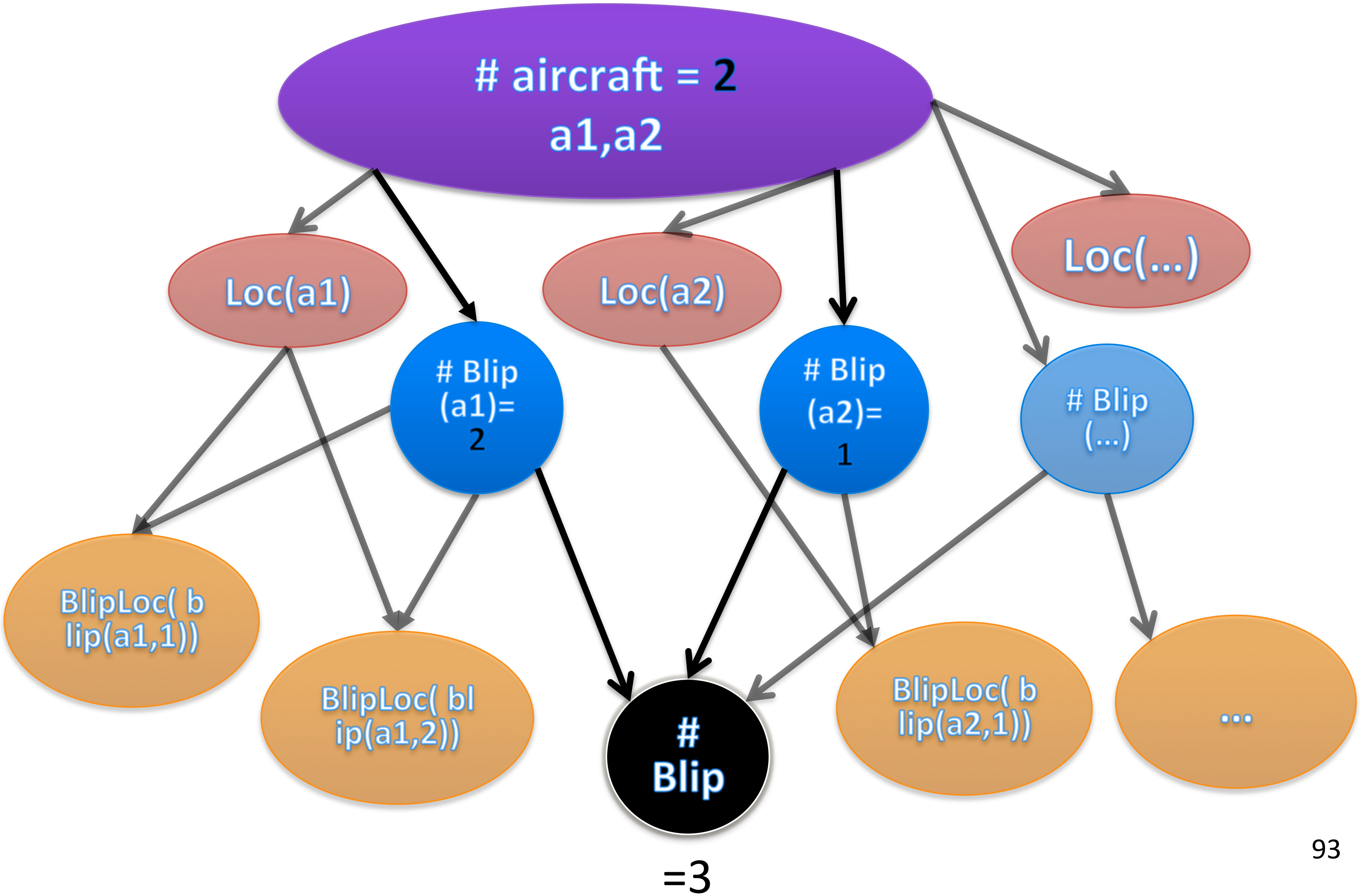
Likelihood Weighting



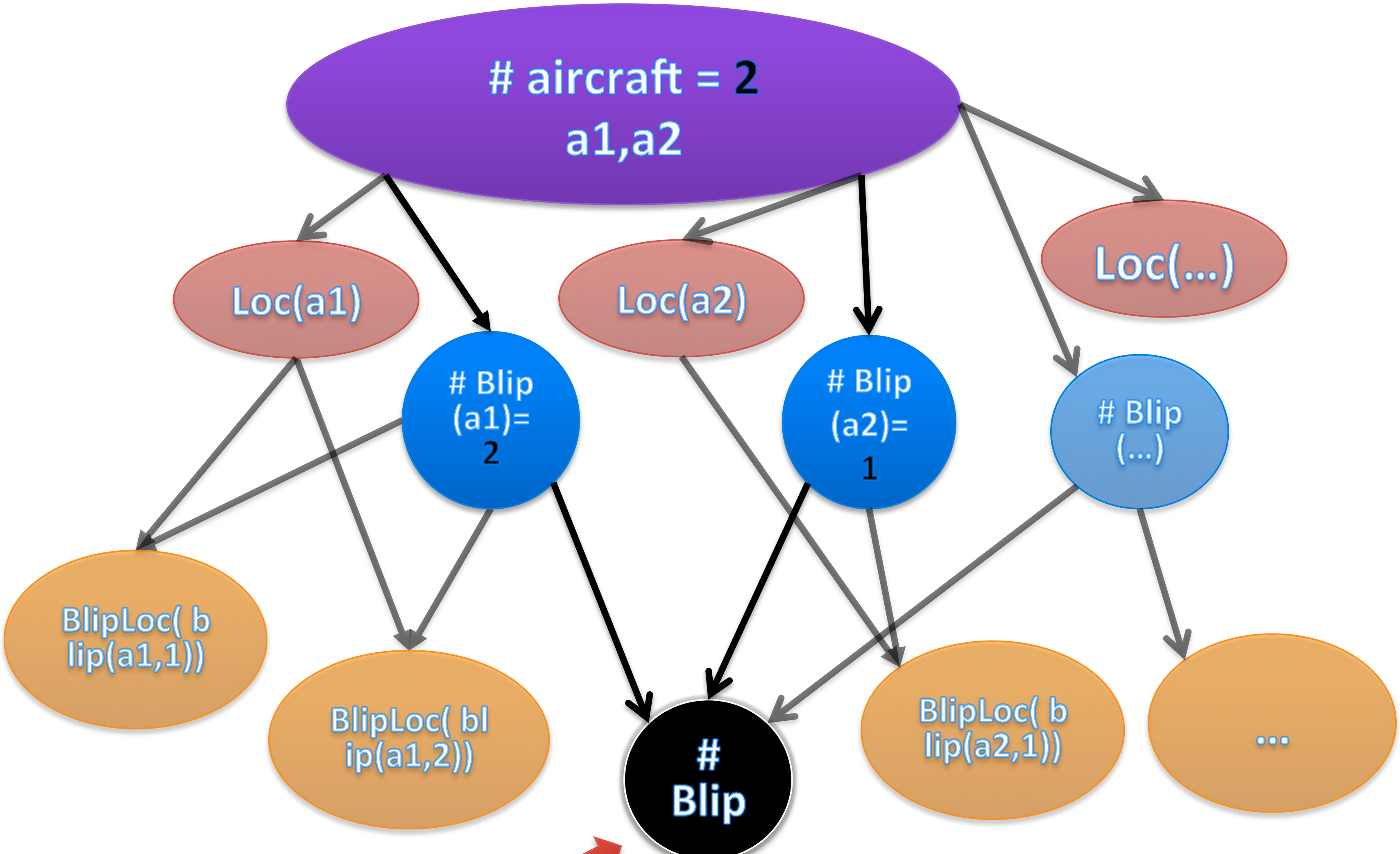
Likelihood Weighting



Likelihood Weighting



Likelihood Weighting



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 $\pi(x')q(x|x') / \pi(x)q(x'|x)$; everything cancels except local changes
- Query evaluation on states is also incrementalizable (cf DB systems)

Switch variable

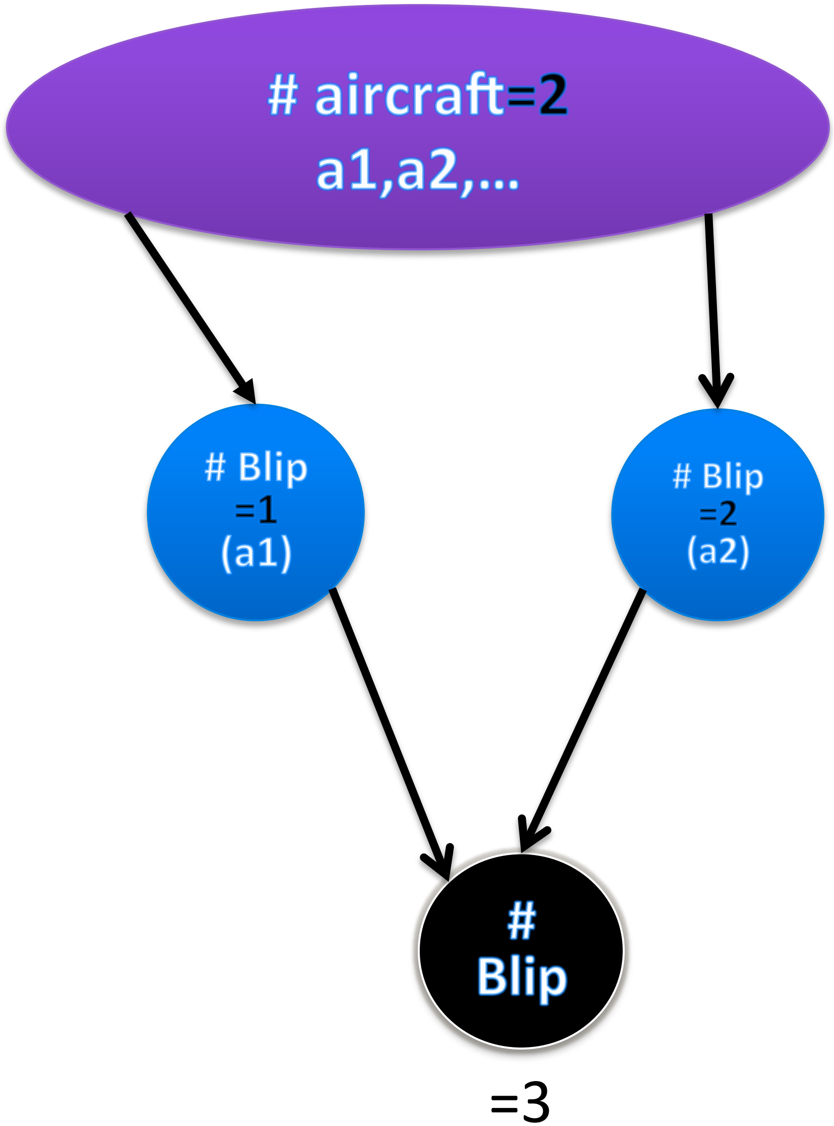
- variable that is either
  - number variable; or
  - function app var appearing in condition of if-then-else, or case-in
  - e.g. #Aircraft

```
random Real pulling_power(Person p, Match m) ~
  if lazy(p, m) then strength(p) / 2.0
  else strength(p);
```

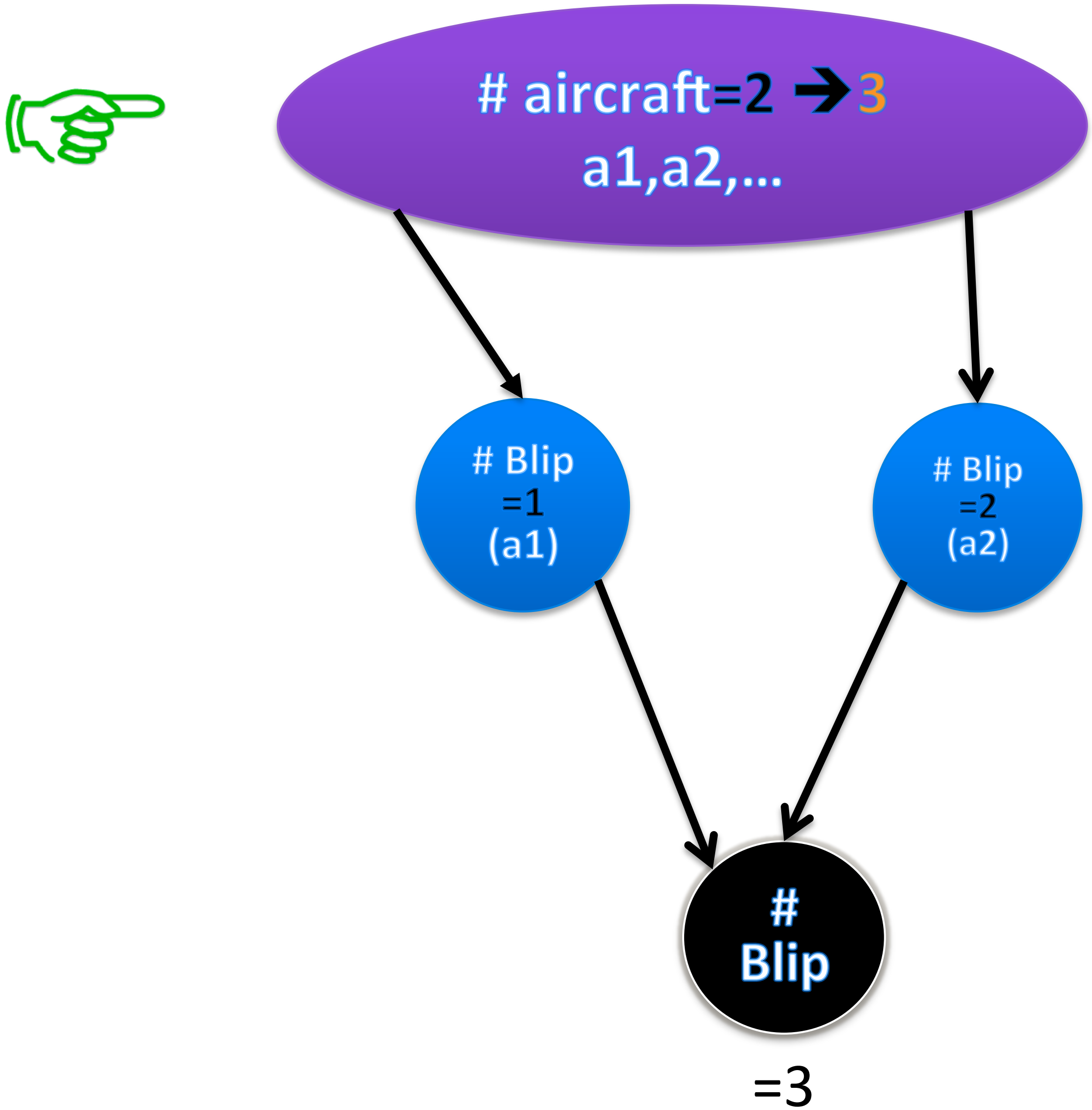
MCMC for BLOG (Milch et al UAI 2006)

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 $\pi(x')q(x|x') / \pi(x)q(x'|x)$

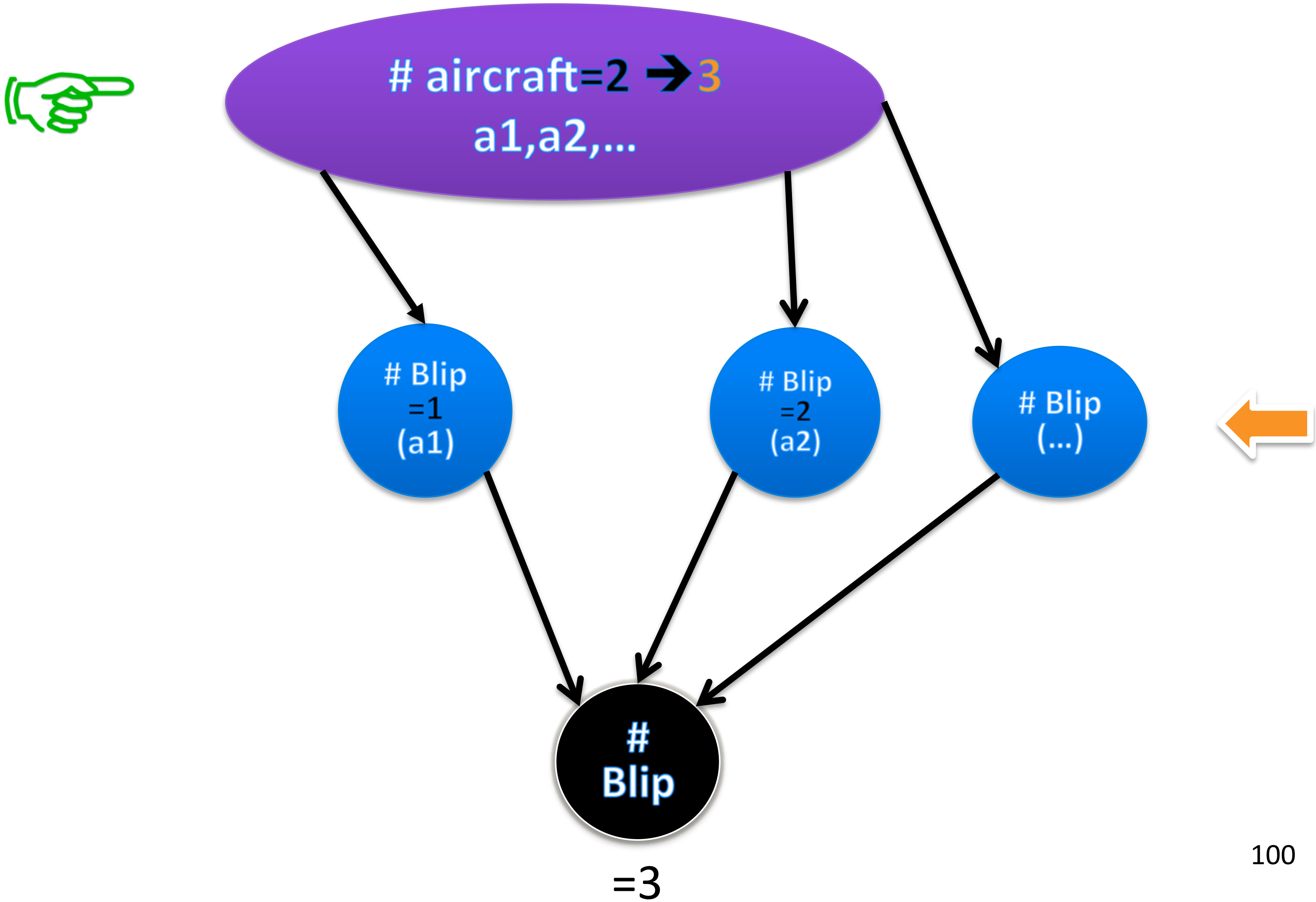
MCMC for BLOG



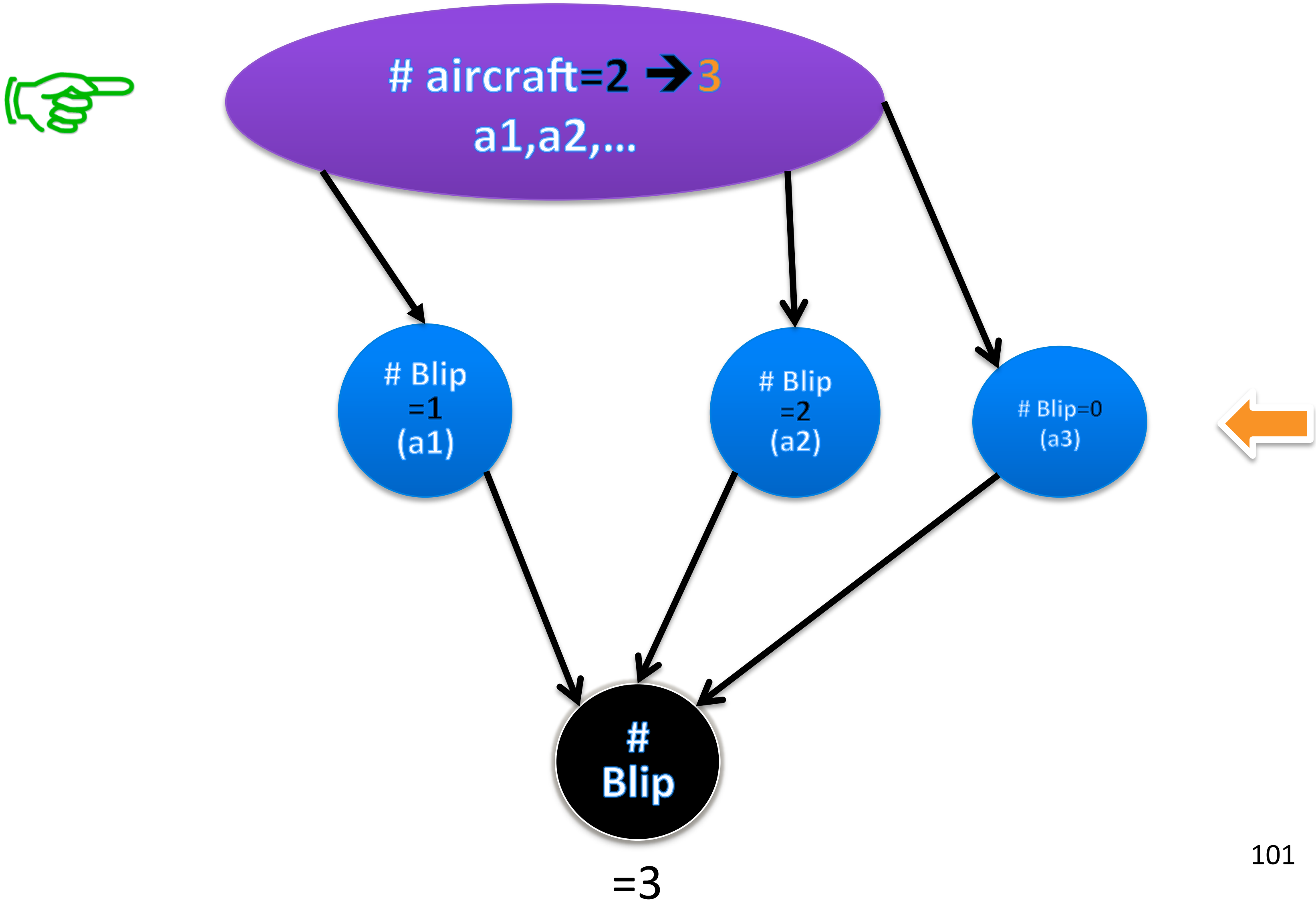
MCMC for BLOG



MCMC for BLOG



MCMC for BLOG



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

  http://bayesianlogic.github.io/download/blog-langref.pdf
- BLOG User manual:

  http://bayesianlogic.github.io/pages/users-manual.html
- 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

- http://bayesianlogic.github.io/pages/interactive-shell-and-debugging-blog-models.html
- 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