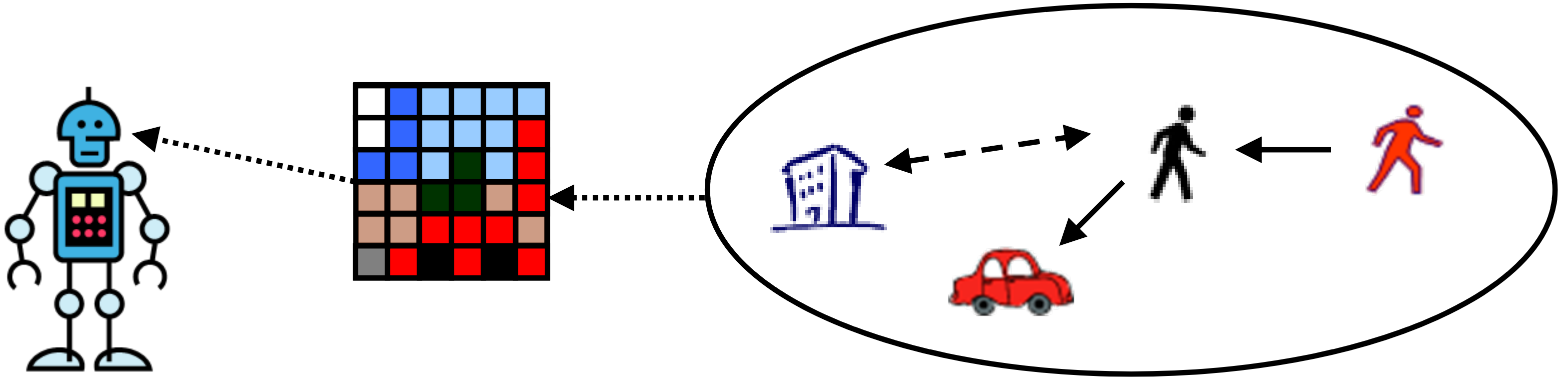
BLOG: Probabilistic Models with Unknown Objects

Brian Milch

Harvard CS 282

November 29, 2007

Handling Unknown Objects

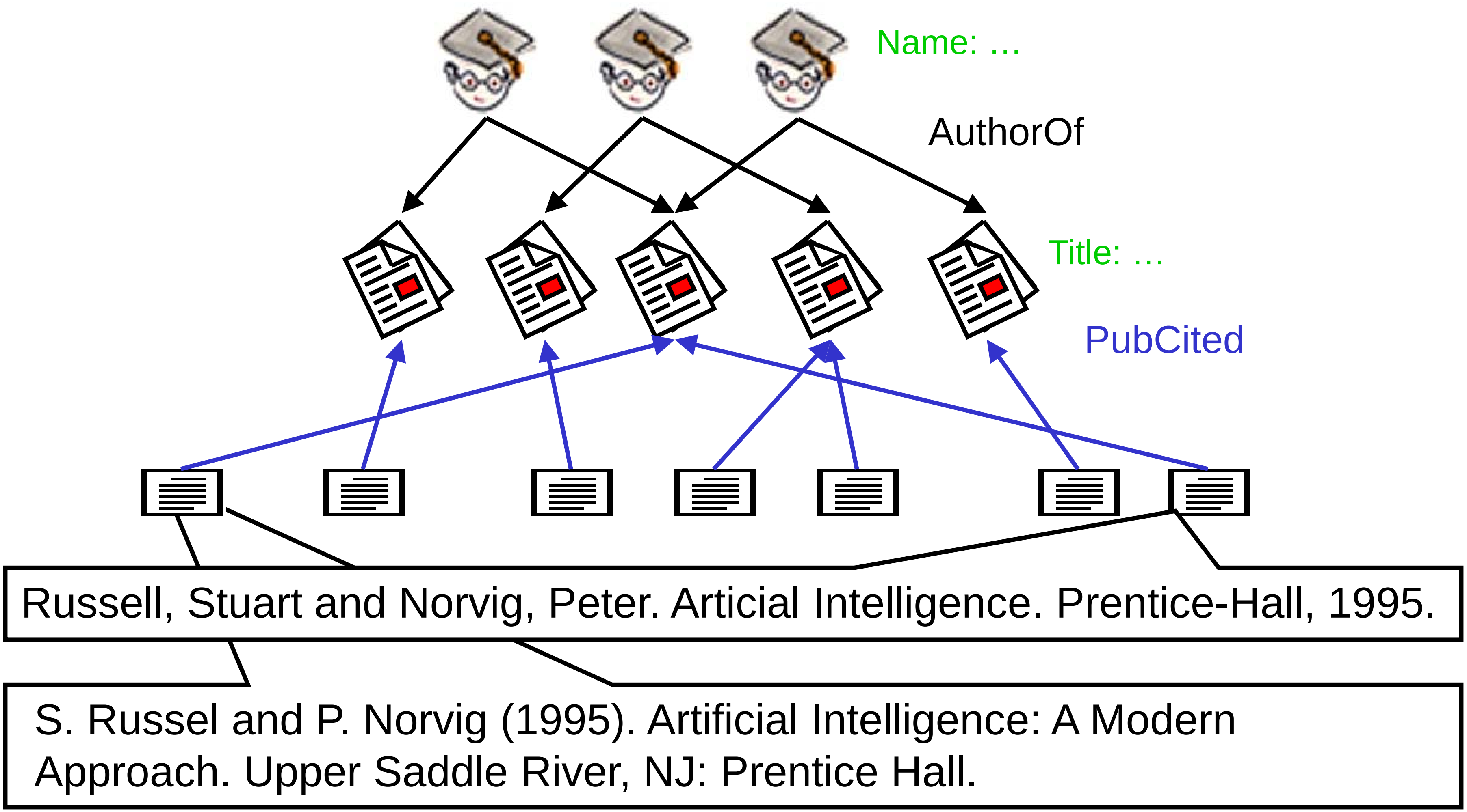


- Fundamental task: given observations, make inferences about initially unknown objects
- But most probabilistic modeling languages assume set of objects is fixed and known
- Bayesian logic (BLOG) lifts this assumption

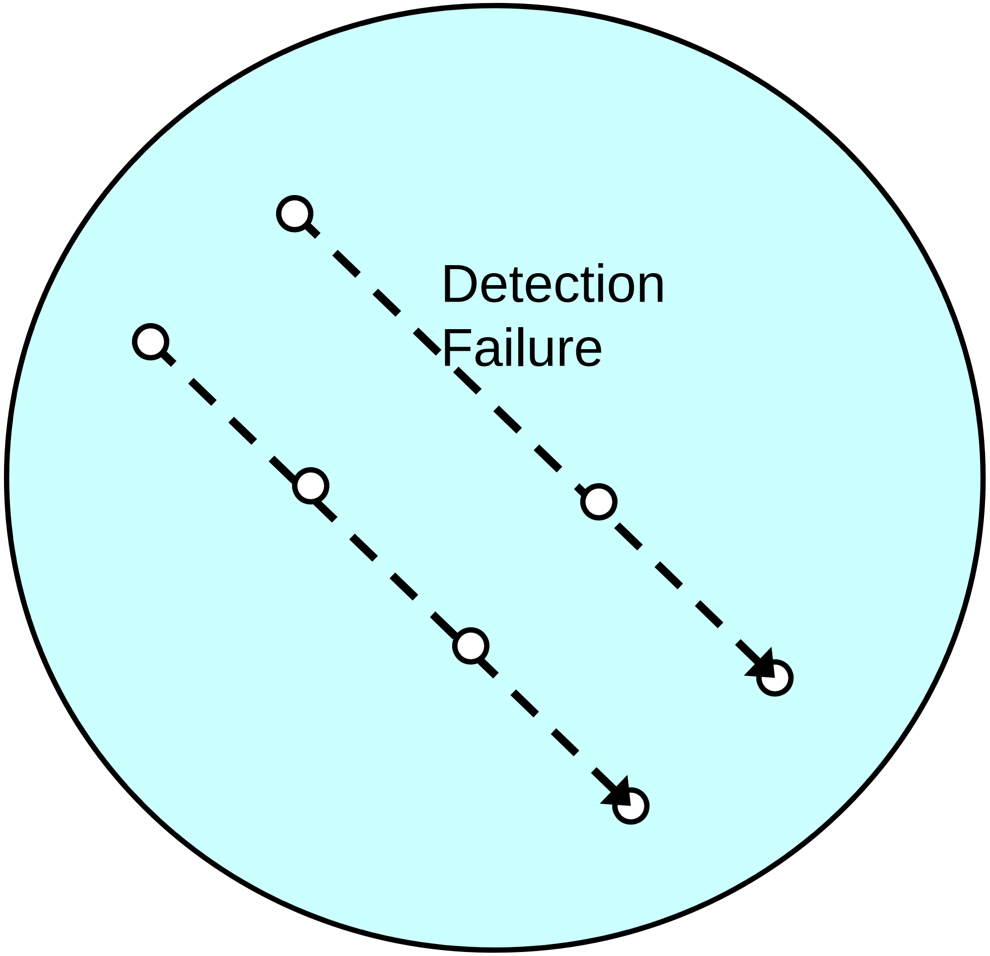
Outline

- Motivating examples
- Bayesian logic (BLOG)
  - Syntax
  - Semantics
- Inference on BLOG models using MCMC

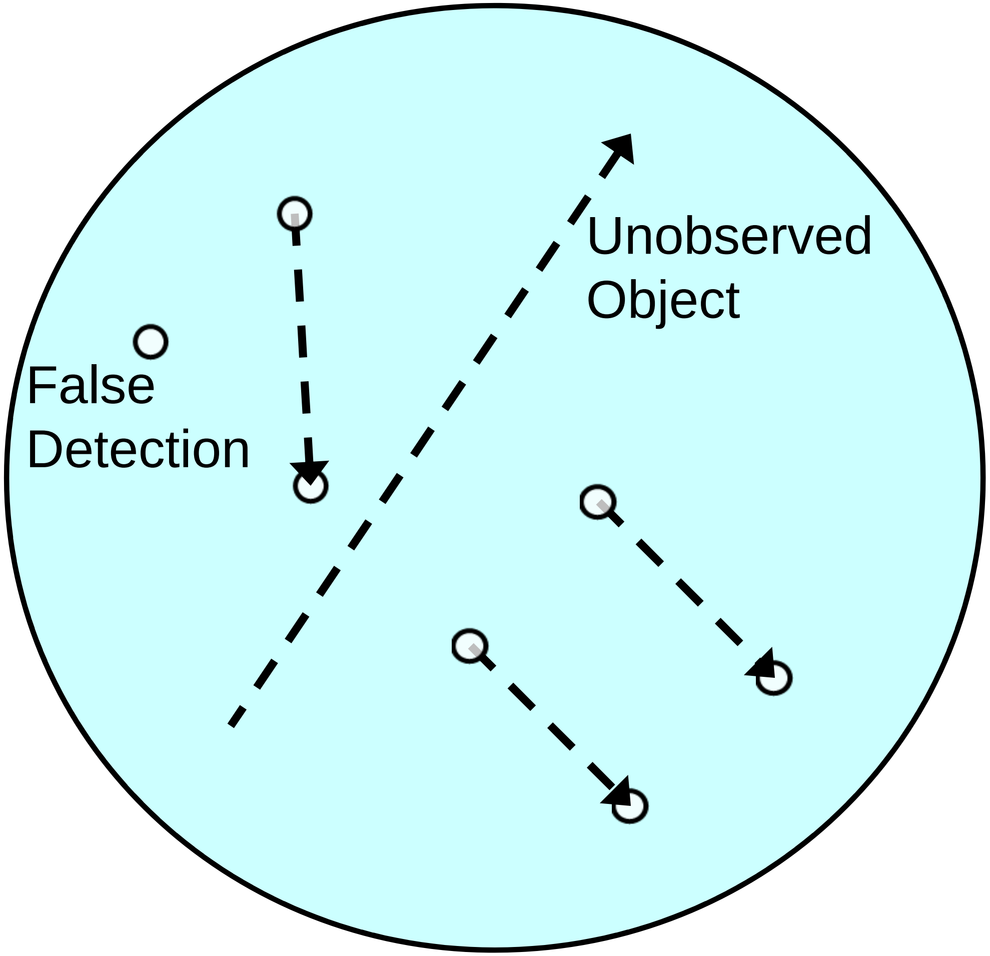
Example 1: Bibliographies



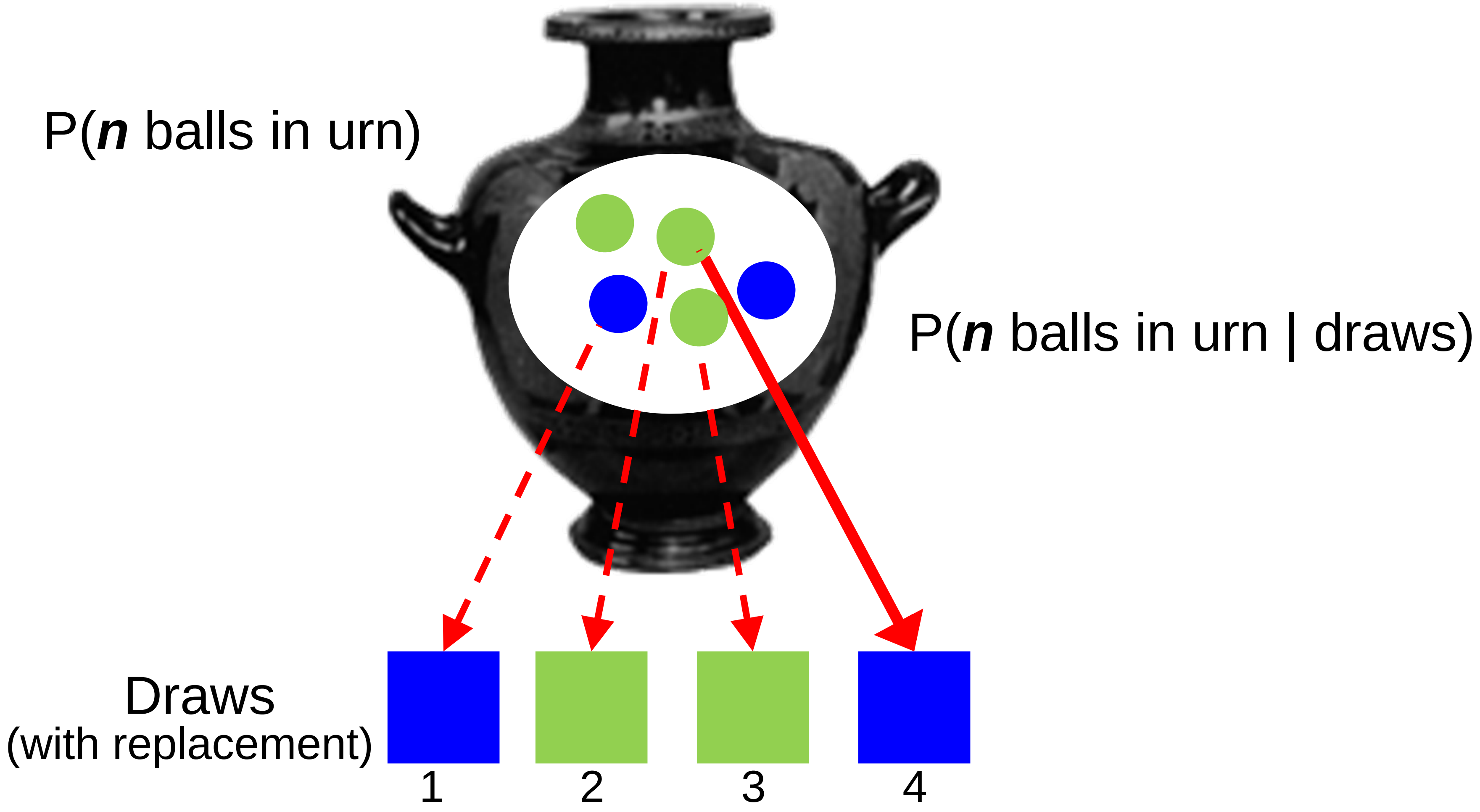
Example 2: Aircraft Tracking



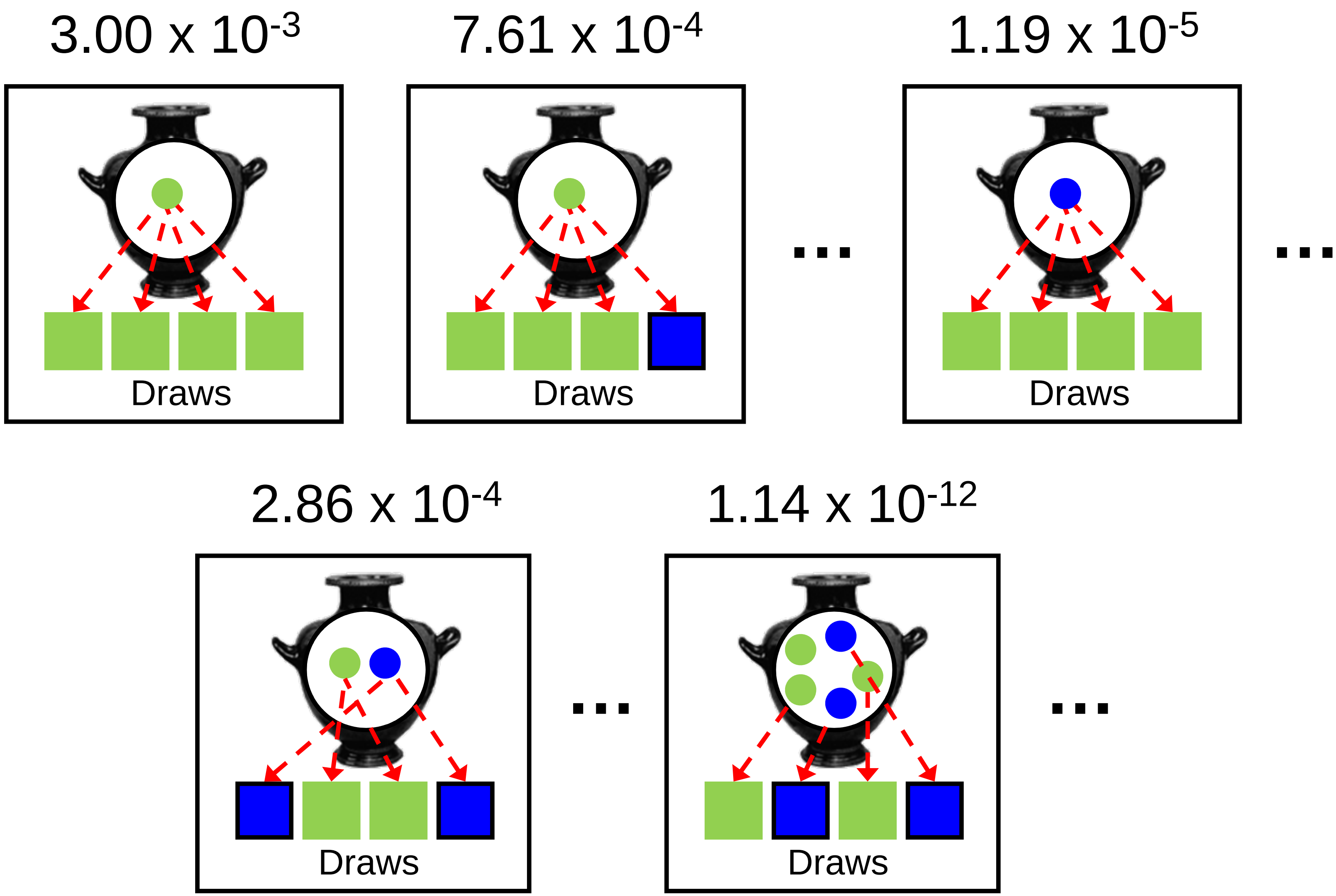
Example 2: Aircraft Tracking



Simple Example: Balls in an Urn



Possible Worlds



Typed First-Order Language

- Types:

  Ball, Draw, Color

  (Built-in types: Boolean, NaturalNum, Real, R*k*Vector, String)

- Function symbols:

  TrueColor: (Ball) → Color
  BallDrawn: (Draw) → Ball
  ObsColor: (Draw) → Color

  Blue: () → Color
  Green: () → Color

  Draw1: () → Draw
  Draw2: () → Draw
  Draw3: () → Draw

  constant symbols

First-Order Structures

- A structure for a typed first-order language maps…
  - Each type $\rightarrow$ a set of objects
  - Each function symbol
    $\rightarrow$ a function on those objects
- A BLOG model defines:
  - A typed first-order language
  - A probability distribution over structures of that language

BLOG Model for Urn and Balls: Header

```
type Color;
type Ball;
type Draw;
```

type declarations

```
random Color TrueColor(Ball);
random Ball BallDrawn(Draw);
random Color ObsColor(Draw);
```

function declarations

```
guaranteed Color Blue, Green;
guaranteed Draw Draw1, Draw2, Draw3, Draw4;
```

guaranteed object statements:
introduce constant symbols,
assert that they denote *distinct* objects

Defining the Distribution: Known Objects

- Suppose only guaranteed objects exist
- Then possible world is fully specified by values for basic random variables

$$\underbrace{V_f}_{\text{random function}}[\underbrace{o_1, \ldots, o_k}_{\text{objects of } f\text{'s argument types}}]$$

- Model will define conditional distributions for these variables

Dependency Statements

Elementary CPD

CPD parameters

```
TrueColor(b) ~ TabularCPD[[0.5, 0.5]]();

BallDrawn(d) ~ Uniform({Ball b});

ObsColor(d)
    if (BallDrawn(d) != null) then
        ~ TabularCPD[[0.8, 0.2],
                     [0.2, 0.8]]
          (TrueColor(BallDrawn(d)));
```

CPD arguments

Syntax of Dependency Statements

$Function(x_1, ..., x_k)$
    if $Cond_1$ then ~ $ElemCPD_1[params](Arg_{1,1}, ..., Arg_{1,m})$
    elseif $Cond_2$ then ~ $ElemCPD_2[params](Arg_{2,1}, ..., Arg_{2,m})$
    ...
    else ~ $ElemCPD_n[params](Arg_{n,1}, ..., Arg_{n,m})$;

- Conditions are arbitrary first-order formulas
- Elementary CPDs are names of Java classes
- Arguments can be terms or set expressions

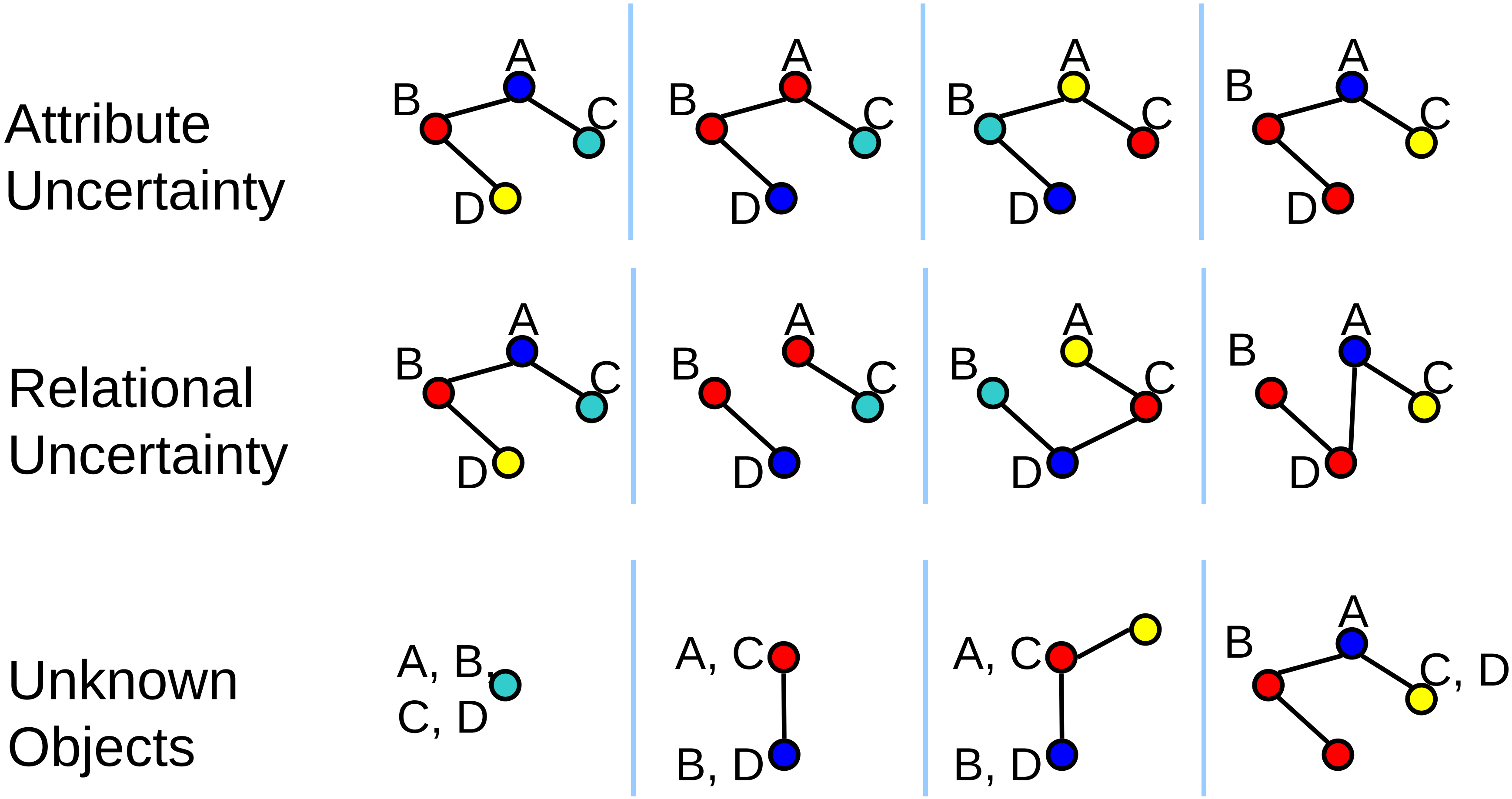
BLOG Model So Far

```
type Color; type Ball;  type Draw;

random Color TrueColor(Ball);
random Ball BallDrawn(Draw);
random Color ObsColor(Draw);

guaranteed Color Blue, Green;
guaranteed Draw Draw1, Draw2, Draw3, Draw4;
```

??? Distribution over what balls exist?

```
TrueColor(b) ~ TabularCPD[[0.5, 0.5]]();

BallDrawn(d) ~ Uniform({Ball b});

ObsColor(d)
    if (BallDrawn(d) != null) then
        ~ TabularCPD[[0.8, 0.2], [0.2, 0.8]]
           (TrueColor(BallDrawn(d)));
```

Challenge of Unknown Objects



Number Statements

```
#Ball ~ Poisson[6]();
```

- Define conditional distributions for basic RVs called number variables, e.g., N_{Ball}
- Can have same syntax as dependency statements:

```
#Candies
  if Unopened(Bag)
    then ~ RoundedNormal[10]
               (MeanCount(Manuf(Bag)))
  else ~ Poisson[50];
```

Full BLOG Model for Urn and Balls

```
type Color; type Ball;  type Draw;

random Color TrueColor(Ball);
random Ball BallDrawn(Draw);
random Color ObsColor(Draw);

guaranteed Color Blue, Green;
guaranteed Draw Draw1, Draw2, Draw3, Draw4;

#Ball ~ Poisson[6]();

TrueColor(b) ~ TabularCPD[[0.5, 0.5]]();

BallDrawn(d) ~ Uniform({Ball b});

ObsColor(d)
    if (BallDrawn(d) != null) then
        ~ TabularCPD[[0.8, 0.2], [0.2, 0.8]]
           (TrueColor(BallDrawn(d)));
```

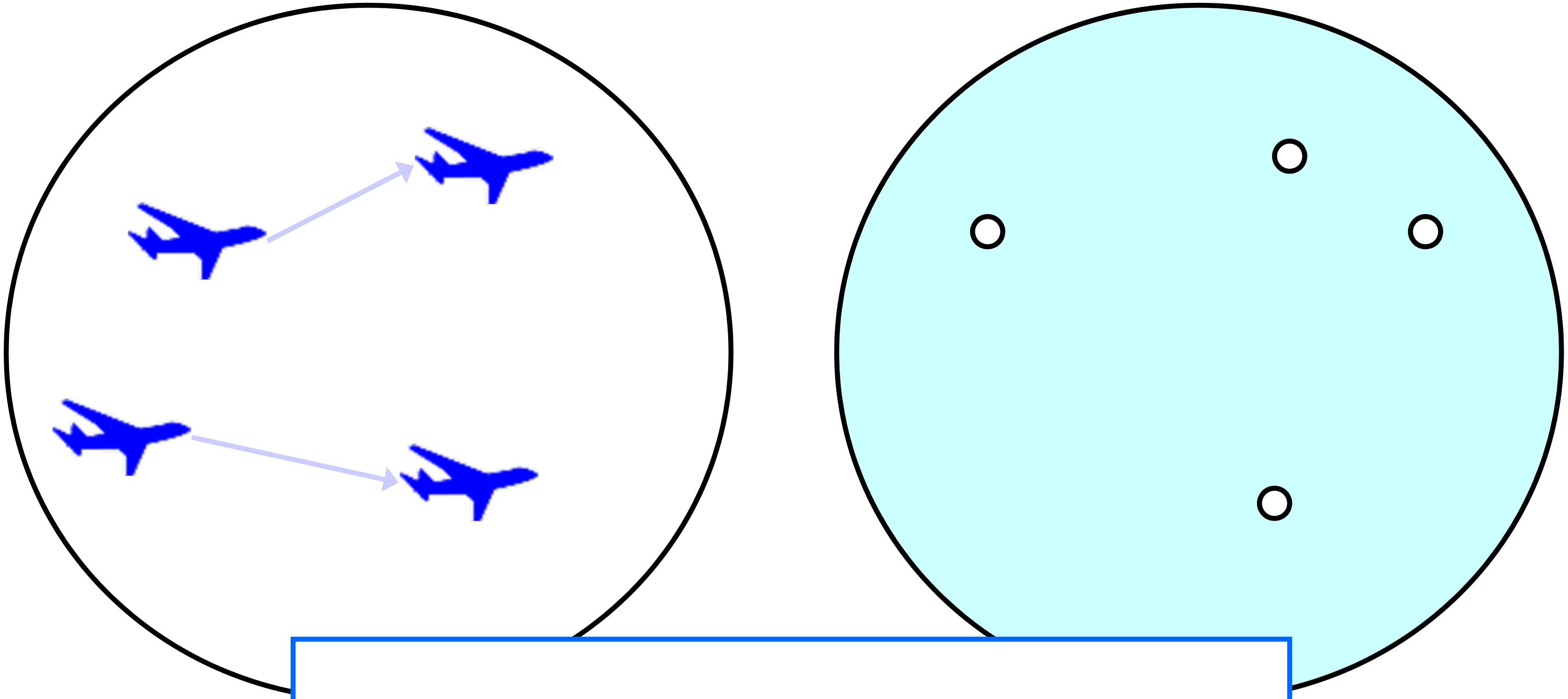
Model for Citations: Header

```
type Res;
type Pub;
type Cit;

random String Name(Res);

random NaturalNum NumAuthors(Pub);
random Res NthAuthor(Pub, NaturalNum);
random String Title(Pub);

random Pub PubCited(Cit);
random String Text(Cit);


guaranteed Citation Cit1, Cit2, Cit3, Cit4;
```

Model for Citations: Body

```
#Res ~ NumResearchersPrior();

Name(r) ~ NamePrior();

#Pub ~ NumPubsPrior();

NumAuthors(p) ~ NumAuthorsPrior();

NthAuthor(p, n)
    if (n < NumAuthors(p)) then ~ Uniform({Res r});

Title(p) ~ TitlePrior();

PubCited(c) ~ Uniform({Pub p});

Text(c) ~ FormatCPD
       (Title(PubCited(c)),
        {n, Name(NthAuthor(PubCited(c), n)) for
           NaturalNum n : n < NumAuthors(PubCited(c))});
```

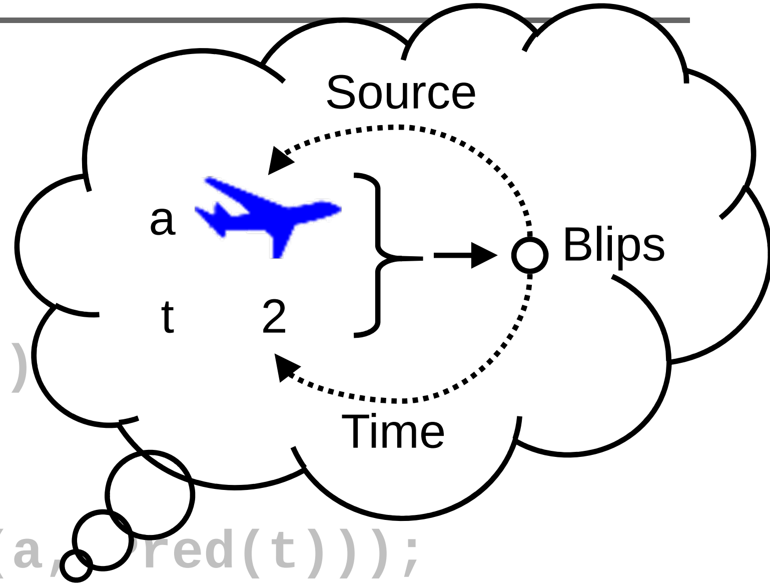
Probability Model for Aircraft Tracking

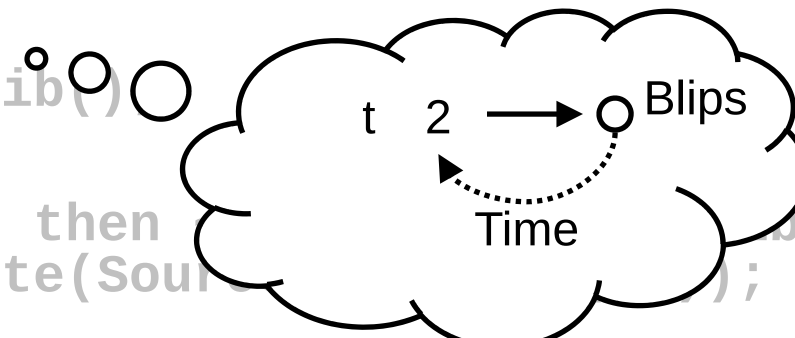


Existence of radar blips depends on existence and locations of aircraft

BLOG Model for Aircraft Tracking

```
origin Aircraft Source(Blip);
origin NaturalNum Time(Blip);
…
#Aircraft ~ NumAircraftDistrib();
State(a, t)
  if t = 0 then ~ InitState()
  else ~ StateTransition(State(a, Pred(t)));
#Blip(Source = a, Time = t)
  ~ NumDetectionsDistrib(State(a, t));
#Blip(Time = t)
  ~ NumFalseAlarmsDistrib();
ApparentPos(r)
  if (Source(r) = null) then ~ ...b()
  else ~ ObsDistrib(State(Sour...));
```





Families of Number Variables

```
#Blip(Source = a, Time = t)
  ~ NumDetectionsDistrib(State(a, t));
```

- Defines family of number variables

N_{blip}[Source = o_s, Time = o_t]

Object of type Aircraft

Object of type NaturalNum

- Note: no dependency statements for origin functions

Outline

- Motivating examples
- Bayesian logic (BLOG)
  - Syntax
  - Semantics
- Inference on BLOG models using MCMC

Declarative Semantics

- What is the set of possible worlds?
  - They’re first-order structures, but with what objects?
- What is the probability distribution over worlds?

What Exactly Are the Objects?

- Potential objects are tuples that encode generation history
  - Aircraft: (Aircraft, 1), (Aircraft, 2), ...
  - Blips from (Aircraft, 2) at time 8:

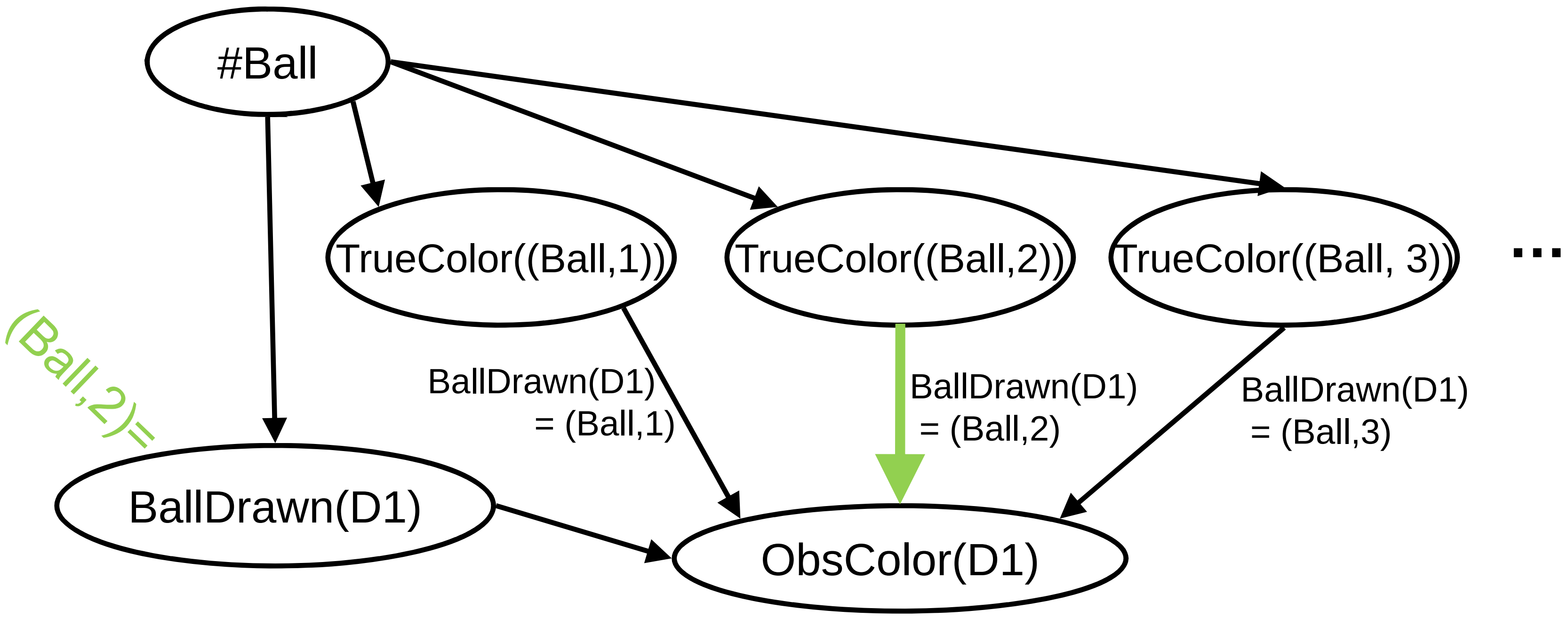
    (Blip, (Source, (Aircraft, 2)), (Time, 8), 1)
    (Blip, (Source, (Aircraft, 2)), (Time, 8), 2)
    ...
- Point: If we specify value for number variable
  N_{blip}[Source=(Aircraft, 2), Time=8]
  there's no ambiguity about *which* blips have this source and time

Worlds and Random Variables

- Recall basic random variables:
  - One for each random function on each tuple of potential arguments
  - One for each number statement and each tuple of potential generating objects
- *Lemma*: Full instantiation of basic RVs uniquely identifies a possible world
- Caveat: Infinitely many potential objects $\rightarrow$ infinitely many basic RVs

Contingent Bayesian Network

- Each BLOG model defines contingent Bayesian network (CBN) over basic RVs
  - Edges active only under certain conditions



[Milch et al., AI/Stats 2005]

BN Semantics

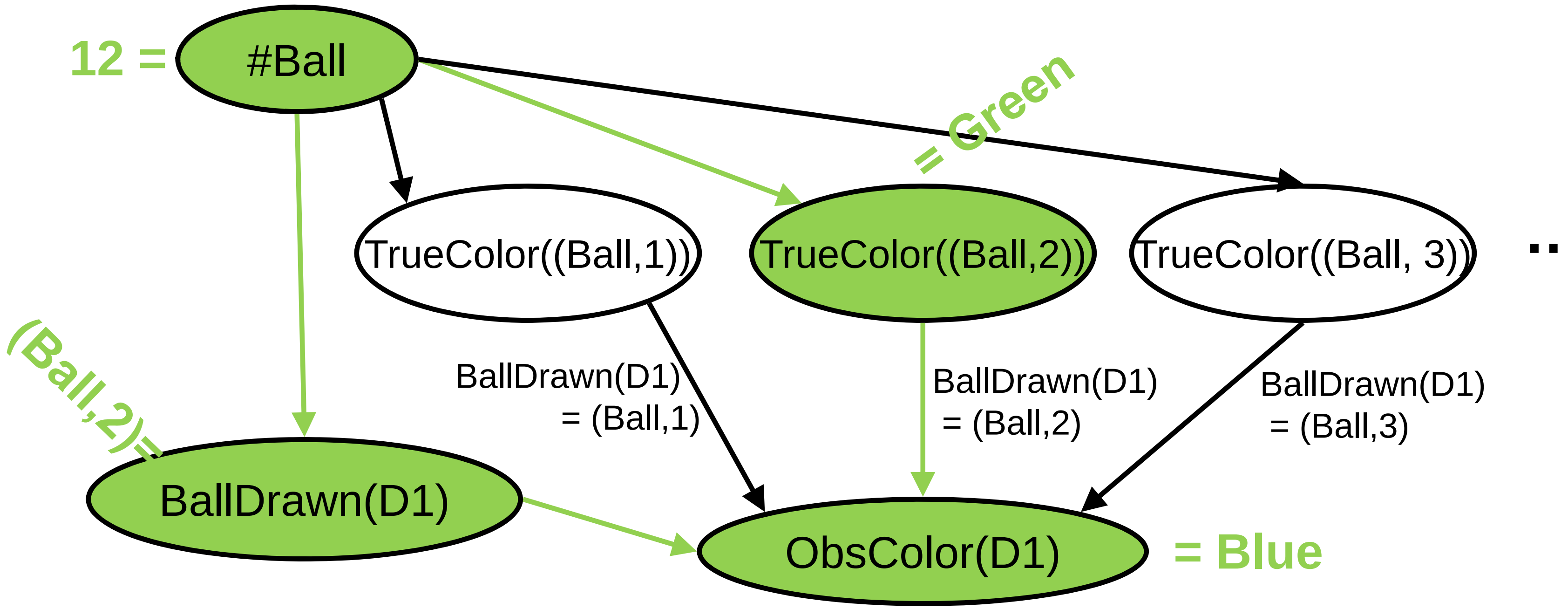
- Usual semantics for BN with *N* nodes:

$$p(x_1,\ldots,x_N) = \prod_{i=1}^{N} p_i(x_i \mid x_{\mathrm{Pa}(i)})$$

- If BN is infinite but has topological numbering X_1, X_2, …, then suffices to make same assertion for each finite prefix of this numbering

But CBN may fail to have topological numbering!

Self-Supporting Instantiations



- $x_1, \ldots, x_n$ is self-supporting if for all $i < n$:
  - $x_1, \ldots, x_{(i-1)}$ determines which parents of X_i are active
  - These active parents are all in $X_1,\ldots,X_{(i-1)}$

Semantics for CBNs and BLOG

- CBN asserts that for each self-supporting instantiation $x_1,\ldots,x_n$:

$$p(x_1,\ldots,x_n) = \prod_{i=1}^{n} p_i(x_i \mid x_{\mathrm{Pa}(i|x_1,\ldots,x_{(i-1)})})$$

- *Theorem*: If CBN satisfies certain conditions (analogous to BN acyclicity), these constraints fully define distribution
- So by earlier lemma, BLOG model fully defines distribution over possible worlds

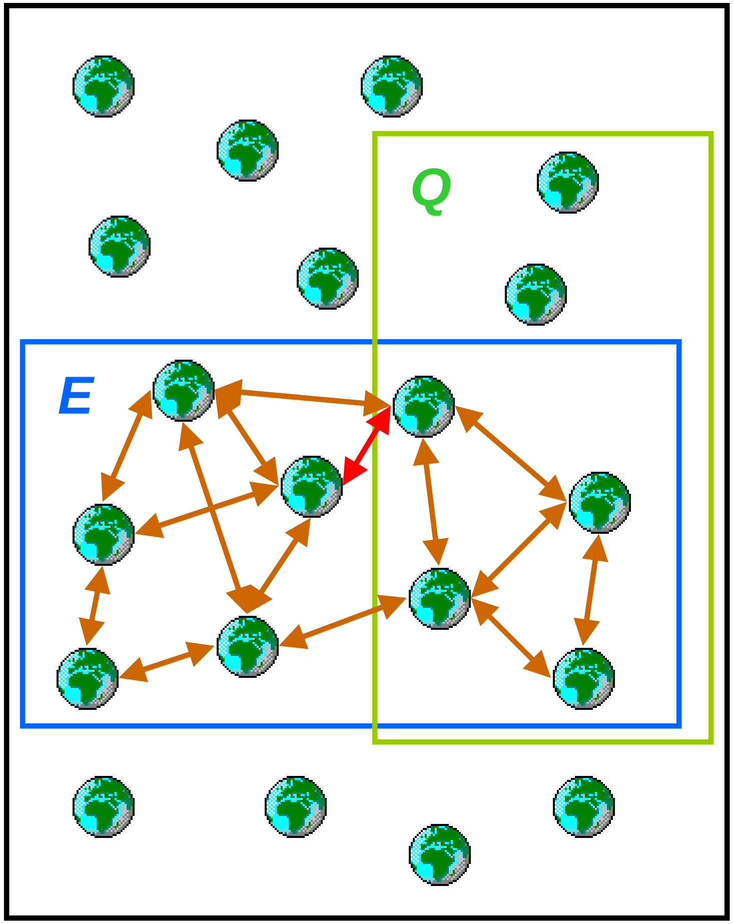
[Milch *et al.*, IJCAI 2005]

Outline

- Motivating examples
- Bayesian logic (BLOG)
  - Syntax
  - Semantics
- Inference on BLOG models using MCMC

Review: Markov Chain Monte Carlo

- Markov chain $\boldsymbol{s}_1$, $\boldsymbol{s}_2$, ... over outcomes in $\boldsymbol{E}$
- Designed so unique stationary distribution is proportional to $\boldsymbol{p}(\boldsymbol{s})$
- Fraction of $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_N$ in query event $\boldsymbol{Q}$ converges to $\boldsymbol{p}(\boldsymbol{Q}|\boldsymbol{E})$ as $\boldsymbol{N} \to \infty$



Metropolis-Hastings MCMC

- Let $\boldsymbol{s}_1$ be arbitrary state in $\boldsymbol{E}$
- For $\boldsymbol{n}$ = 1 to $\boldsymbol{N}$
  - Sample $\boldsymbol{s}' \in \boldsymbol{E}$ from proposal distribution $\boldsymbol{q}(\boldsymbol{s}' \mid \boldsymbol{s}_n)$
  - Compute acceptance probability

$$\alpha = \max\left(1, \frac{p(s')\, q(s_n \mid s')}{p(s_n)\, q(s' \mid s_n)}\right)$$

  - With probability α, let $\boldsymbol{s}_{n+1} = \boldsymbol{s}'$;
    else let $\boldsymbol{s}_{n+1} = \boldsymbol{s}_n$

Stationary distribution is proportional to $\boldsymbol{p}(\boldsymbol{s})$

$\Downarrow$

Fraction of visited states in $\boldsymbol{Q}$ converges to $\boldsymbol{p}(\boldsymbol{Q}|\boldsymbol{E})$

Toward General-Purpose Inference

- Successful applications of MCMC with domain-specific proposal distributions:
  - Citation matching [Pasula et al., 2003]
  - Multi-target tracking [Oh et al., 2004]
- But each application requires new code for:
  - Proposing moves
  - Representing MCMC states
  - Computing acceptance probabilities

- Goal:
  - User specifies model and proposal distribution
  - General-purpose code does the rest

General MCMC Engine

[Milch *et al*., UAI 2006]

Model
(in BLOG)

- Define $\boldsymbol{p}(\boldsymbol{s})$

1. What are the MCMC states?

Custom proposal distribution
(Java class)

- Propose MCMC state $\boldsymbol{s}'$ given $\boldsymbol{s}_n$
- Compute ratio $\boldsymbol{q}(\boldsymbol{s}_n \mid \boldsymbol{s}') / \boldsymbol{q}(\boldsymbol{s}' \mid \boldsymbol{s}_n)$

- Compute acceptance probability based on model
- Set $\boldsymbol{s}_{n+1}$

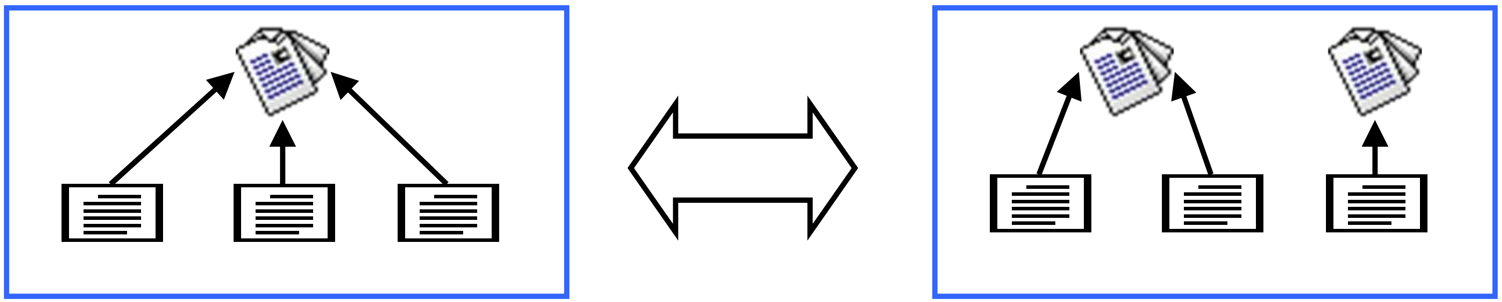
General-purpose engine
(Java code)

2. How does the engine handle arbitrary proposals efficiently?

Proposer for Citations

[Pasula *et al.*, NIPS 2002]

- Split-merge moves:



  - Propose titles and author names for affected publications based on citation strings
- Other moves change total number of publications

MCMC States

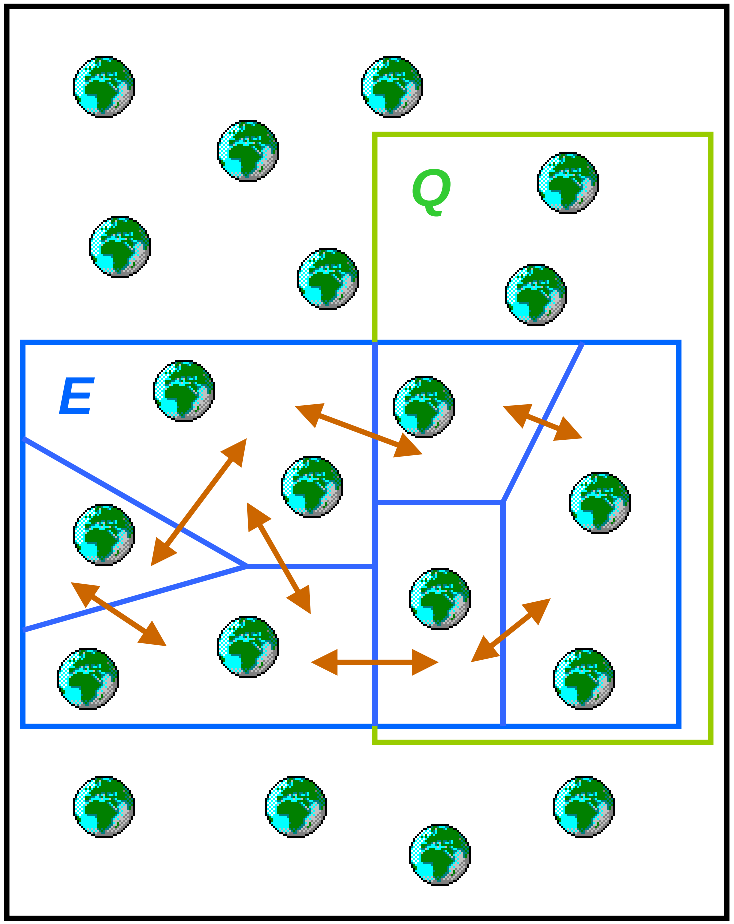
- Not complete instantiations!
  - No titles, author names for uncited publications
- States are partial instantiations of random variables

  #Pub = 100, PubCited(Cit1) = (Pub, 37), Title((Pub, 37)) = “Calculus”

  - Each state corresponds to an event: set of outcomes satisfying description

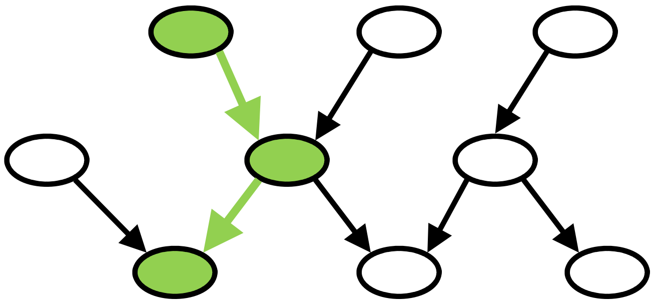
MCMC over Events

- Markov chain over events σ, with stationary distrib. proportional to $\boldsymbol{p}(\sigma)$
- *Theorem:* Fraction of visited events in ***Q*** converges to $\boldsymbol{p}(\boldsymbol{Q}|\boldsymbol{E})$ if:
  – Each σ is either subset of ***Q*** or disjoint from ***Q***
  – Events form partition of ***E***



Computing Probabilities of Events

- Engine needs to compute $\boldsymbol{p}(\sigma') \,/\, \boldsymbol{p}(\sigma_{\boldsymbol{n}})$ efficiently (without summations)
- Use self-supporting instantiations
- Then probability is product of CPDs:



$$p(\sigma) = p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p_i(x_i \mid x_{\mathrm{Pa}(i \mid x_1, \ldots, x_{(i-1)})})$$

States That Are Even More Abstract

- Typical partial instantiation:

#Pub = 100, PubCited(Cit1) = (Pub, 37), Title((Pub, 37)) = "Calculus",
PubCited(Cit2) = (Pub, 14), Title((Pub, 14)) = "Psych"

  - Specifies *particular* publications, even though publications are interchangeable

- Let states be abstract partial instantiations:

$\exists\, \boldsymbol{x}\, \exists\, \boldsymbol{y} \neq \boldsymbol{x}$ [#Pub = 100, PubCited(Cit1) = ***x***, Title(***x***) = "Calculus",
PubCited(Cit2) = ***y***, Title(***y***) = "Psych"]

- There are conditions under which we can compute probabilities of such events

Computing Acceptance Probabilities Efficiently

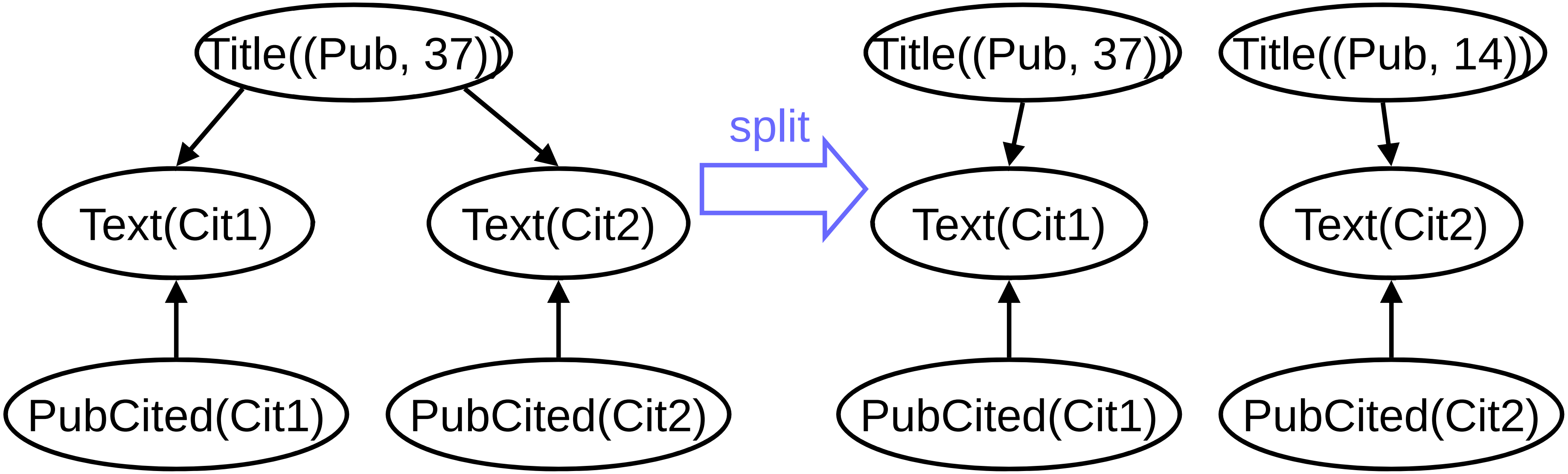
- First part of acceptance probability is:

$$\frac{p(\sigma')}{p(\sigma_n)} = \frac{\prod_{i \in \mathrm{vars}(\sigma')} p_i\left(x'_i \mid x'_{\mathrm{Pa}(i|\sigma')}\right)}{\prod_{i \in \mathrm{vars}(\sigma_n)} p_i\left(x_i \mid x_{\mathrm{Pa}(i|\sigma_n)}\right)}$$

- If moves are local, most factors cancel
- Need to compute factors for X_i only if proposal changes X_i or one of $\mathrm{Pa}(i \mid \sigma_n)$

Identifying Factors to Compute

- Maintain list of changed variables
- To find children of changed variables, use context-specific BN
- Update context-specific BN as active dependencies change



Results on Citation Matching

		Face (349 cits)	Reinforce (406 cits)	Reasoning (514 cits)	Constraint (295 cits)
Hand-coded	Acc:	95.1%	81.8%	88.6%	91.7%
	Time:	14.3 s	19.4 s	19.0 s	12.1 s
BLOG engine	Acc:	95.6%	78.0%	88.7%	90.7%
	Time:	69.7 s	99.0 s	99.4 s	59.9 s

- Hand-coded version uses:
  - Domain-specific data structures to represent MCMC state
  - Proposer-specific code to compute acceptance probabilities
- BLOG engine takes 5x as long to run
- But it’s faster than hand-coded version was in 2003!
  (hand-coded version took 120 secs on old hardware and JVM)

BLOG Software

- Bayesian Logic inference engine available:

  http://people.csail.mit.edu/milch/blog

Summary

- Modeling unknown objects is essential
- BLOG models define probability distributions over possible worlds with
  - Varying sets of objects
  - Varying mappings from observations to objects
- Can do inference on BLOG models using MCMC over partial worlds