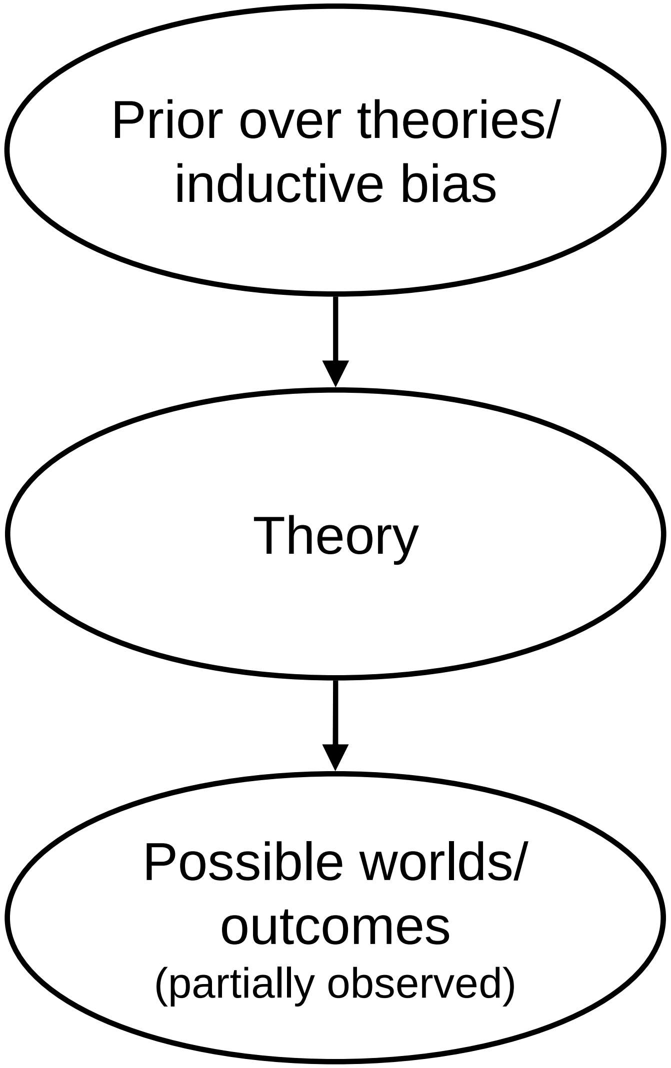
First-Order Probabilistic Models

Brian Milch
http://people.csail.mit.edu/milch

9.66: Computational Cognitive Science

December 7, 2006

Theories


Prior over theories/
inductive bias
Theory
Possible worlds/
outcomes
(partially observed)

How Can Theories be Represented?

Deterministic	Probabilistic
Propositional formulas	Bayesian network
Finite state automaton	N-gram model Hidden Markov model
Context-free grammar	Probabilistic context-free grammar
First-order formulas	First-order probabilistic model

Outline

- Motivation: Why first-order models?
- Models with known objects and relations
  - Representation with probabilistic relational models (PRMs)
  - Inference (not much to say)
  - Learning by local search
- Models with unknown objects and relations
  - Representation with Bayesian logic (BLOG)
  - Inference by likelihood weighting and MCMC
  - Learning (not much to say)

Propositional Theory (Deterministic)

- Scenario with students, courses, profs

  Dr. Pavlov teaches CS1 and CS120
  Matt takes CS1
  Judy takes CS1 and CS120

- Propositional theory

$PavlovDemanding \rightarrow CS1Hard$

$PavlovDemanding \rightarrow CS120Hard$

$CS1Hard \rightarrow MattTired$

$CS1Hard \rightarrow JudyTired$

$CS120Hard \rightarrow JudyTired$

$\neg CS1Hard \rightarrow MattGetsAInCS1$

$\neg CS1Hard \rightarrow JudyGetsAInCS1$

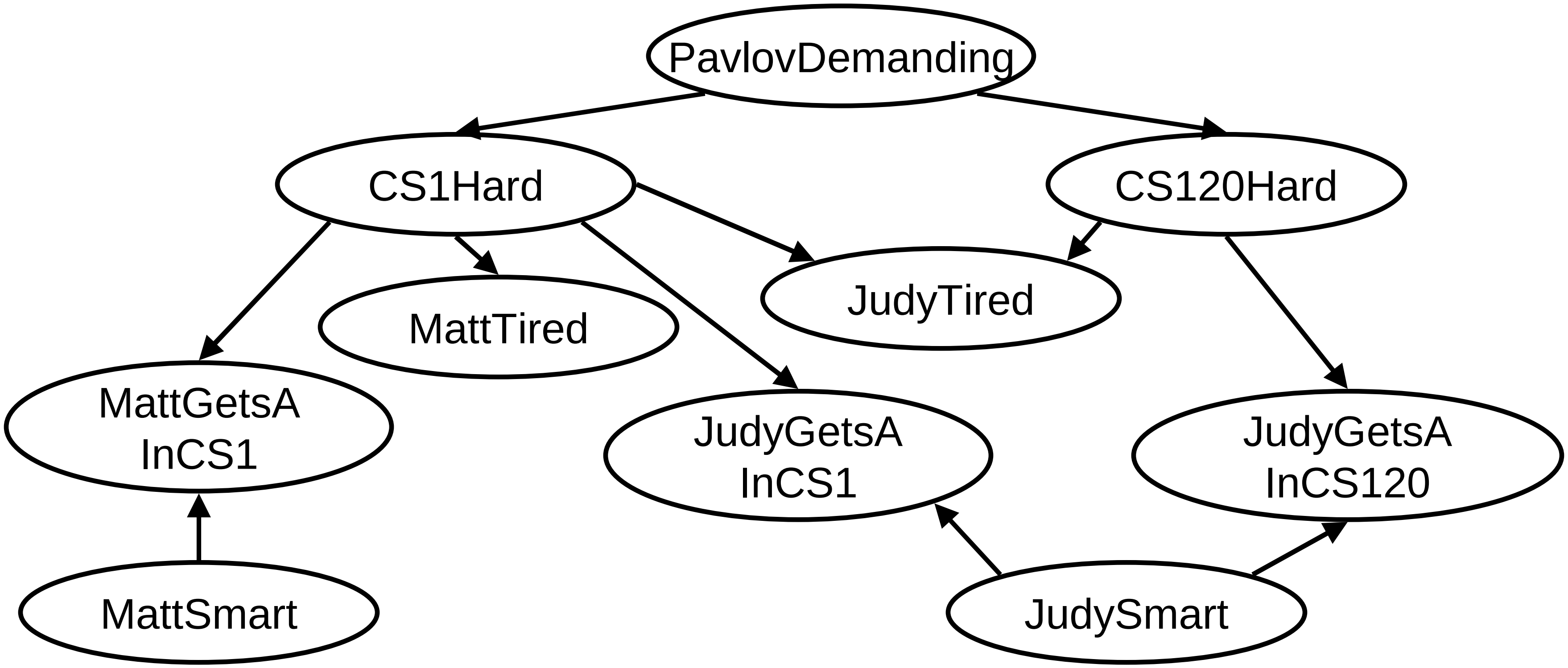
$\neg CS120Hard \rightarrow JudyGetsAInCS120$

$MattSmart \wedge CS1Hard \rightarrow MattGetsAInCS1$

$JudySmart \wedge CS1Hard \rightarrow JudyGetsAInCS1$

$JudySmart \wedge CS120Hard \rightarrow JudyGetsAInCS120$

Propositional Theory (Probabilistic)



- Specific to particular scenario (who takes what, etc.)
- No generalization of knowledge across objects

First-Order Theory

- General theory:

  $\forall$ p $\forall$ c [Teaches(p, c) $\wedge$ Demanding(p) $\rightarrow$ Hard(c)]

  $\forall$ s $\forall$ c [Takes(s, c) $\wedge$ Hard(c) $\rightarrow$ Tired(s, c)]

  $\forall$ s $\forall$ c [Takes(s, c) $\wedge$ Easy(c) $\rightarrow$ GetsA(s, c)]

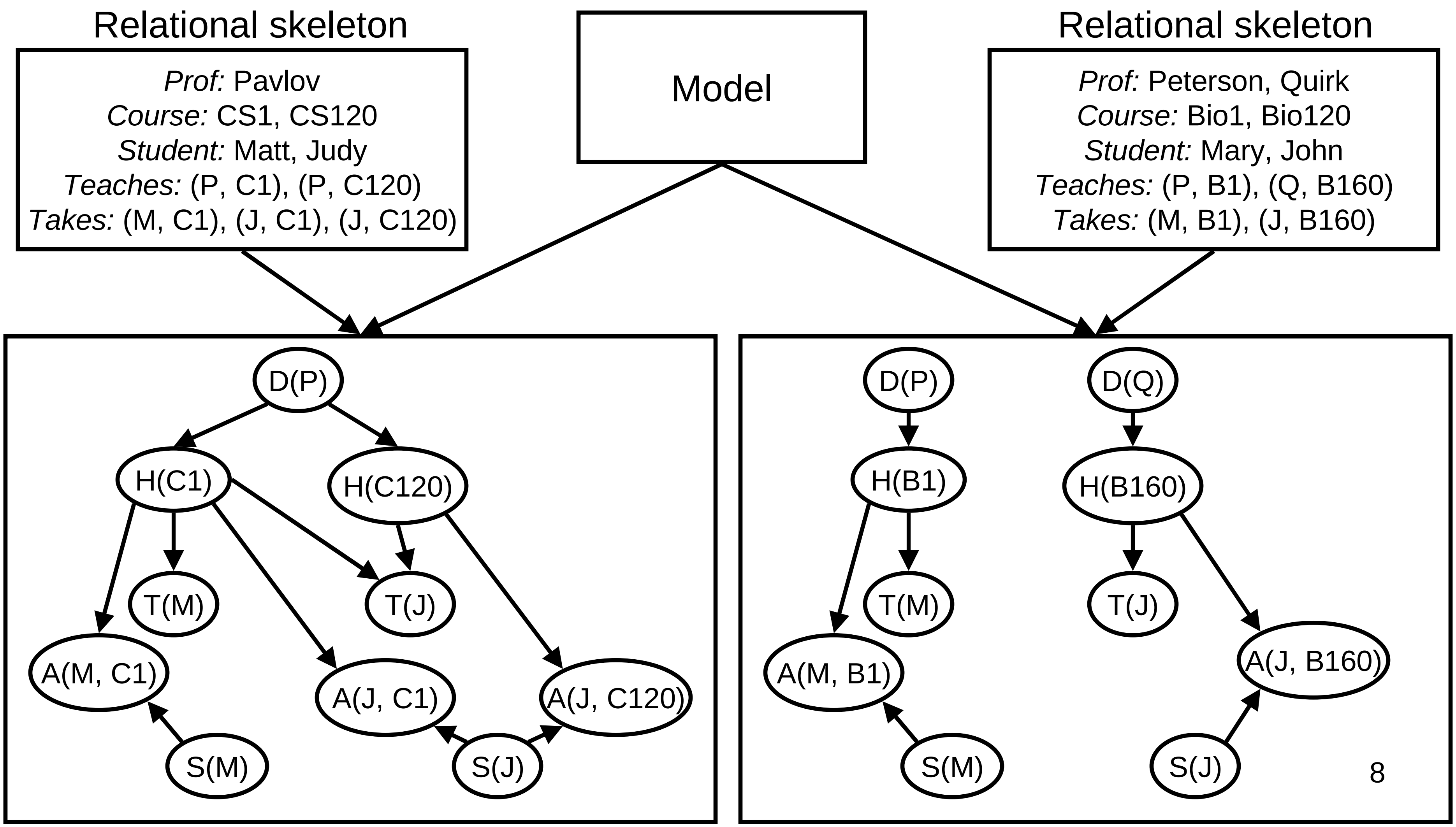
  $\forall$ s $\forall$ c [Takes(s, c) $\wedge$ Hard(c) $\wedge$ Smart(s) $\rightarrow$ GetsA(s, c)]

- Relational skeleton:

Teaches(Pavlov, CS1)	Teaches(Pavlov, CS120)
Takes(Matt, CS1)	
Takes(Judy, CS1)	Takes(Judy, CS120)

- Compact, generalizes across scenarios and objects
- But deterministic

Task for First-Order Probabilistic Model



First-Order Probabilistic Models with Known Skeleton

- Random functions become indexed families of random variables

  Demanding(p)   Hard(c)   Tired(s)   Smart(s)   GetsA(s, c)

- For each family of RVs, specify:
  - How to determine parents from relations
  - CPD that can handle varying numbers of parents
- One way to do this:
  probabilistic relational models (PRMs)
  [Koller & Pfeffer 1998; Friedman, Getoor, Koller & Pfeffer 1999]

Probabilistic Relational Models

- Functions/relations treated as slots on objects
  - Simple slots (random)
    p.Demanding, c.Hard, s.Smart, s.Tired
  - Reference slots (nonrandom; value may be a set)
    p.Teaches, c.TaughtBy
- Specify parents with slot chains
  c.Hard $\leftarrow$ {c.TaughtBy.Demanding}
- Introduce link objects for non-unary functions
  - new type: Registration
  - reference slots: r.Student, r.Course, c.RegisteredIn
  - simple slots: r.GetsA

PRM for Academic Example

p.Demanding $\leftarrow$ {}

c.Hard $\leftarrow$ {c.TaughtBy.Demanding}

s.Smart $\leftarrow$ {}

r.GetsA $\leftarrow$ {r.Course.Hard, r.Student.Smart}

s.Tired $\leftarrow$ {**`#True`**(c.RegisteredIn.Course.Hard)}

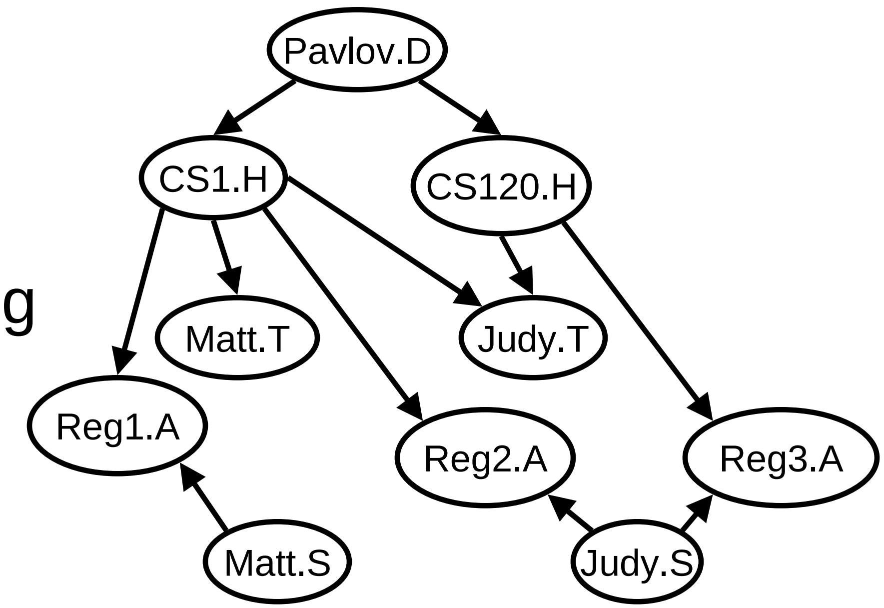
Aggregation function: takes multiset of slot chain values, returns single value

⇩

CPDs always get one parent value per slot chain

Inference in PRMs

- Construct ground BN
  - Node for each simple slot on each object
  - Edges found by following parent slot chains
- Run a BN inference algorithm
  - Exact (variable elimination)
  - Gibbs sampling
  - Loopy belief propagation



Warning: May be intractable

[Although see Pfeffer et al. (1999) paper on SPOOK for smarter method]

Learning PRMs

- Learn structure: for each simple slot, a set of parent slot chains with aggregation functions

$$P(S \mid D) \propto \underbrace{P(S)}_{\text{prior}} \underbrace{\int P(D \mid \theta, S) P(\theta \mid S)\, d\theta}_{\text{marginal likelihood}}$$

- Marginal likelihood
  - prefers fitting the data well
  - penalizes having lots of parameters, i.e., lots of parents
- Prior penalizes long slot chains:

$$P(S) \propto \exp\left(- \sum_{F \in \text{slots}} \sum_{C \in \text{Pa}_S(F)} \text{length}(C) \right)$$

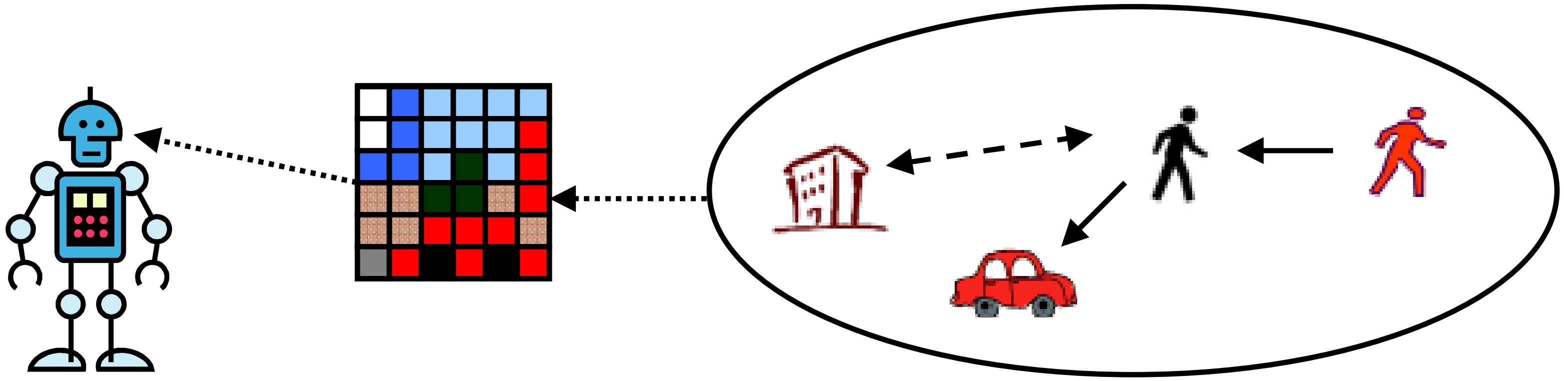
PRM Learning Algorithm

- Local search over structures
  - Operators add, remove, reverse slot chains
  - Greedy: looks at all possible moves, choose one that increases score the most
- Proceed in phases
  - Increase max slot chain length each time
  - Until no improvement in score

PRM Benefits and Limitations

- Benefits
  - Generalization across objects
    - Models are compact
    - Don't need to learn new theory for each new scenario
  - Learning algorithm is known
- Limitations
  - Slot chains are restrictive, e.g., can't say
    GoodRec(p, s) $\leftarrow$ {GotA(s, c) : TaughtBy(c, p)}
  - Objects and relations have to be specified in skeleton [although see later extensions to PRM language]

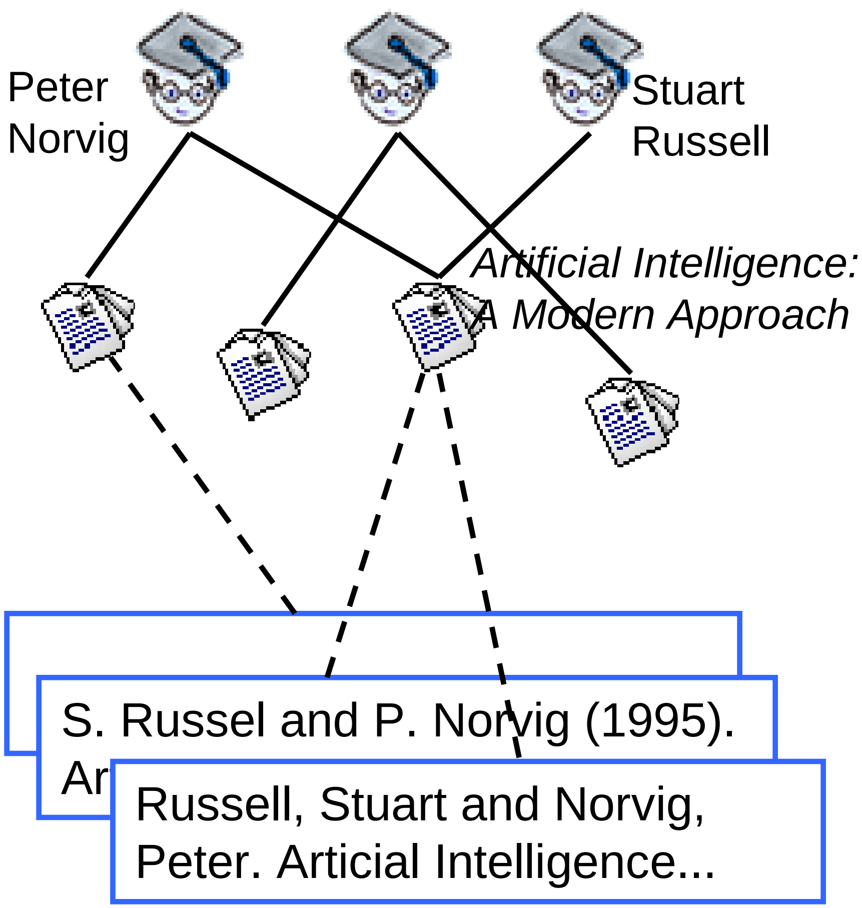
Basic Task for Intelligent Agents



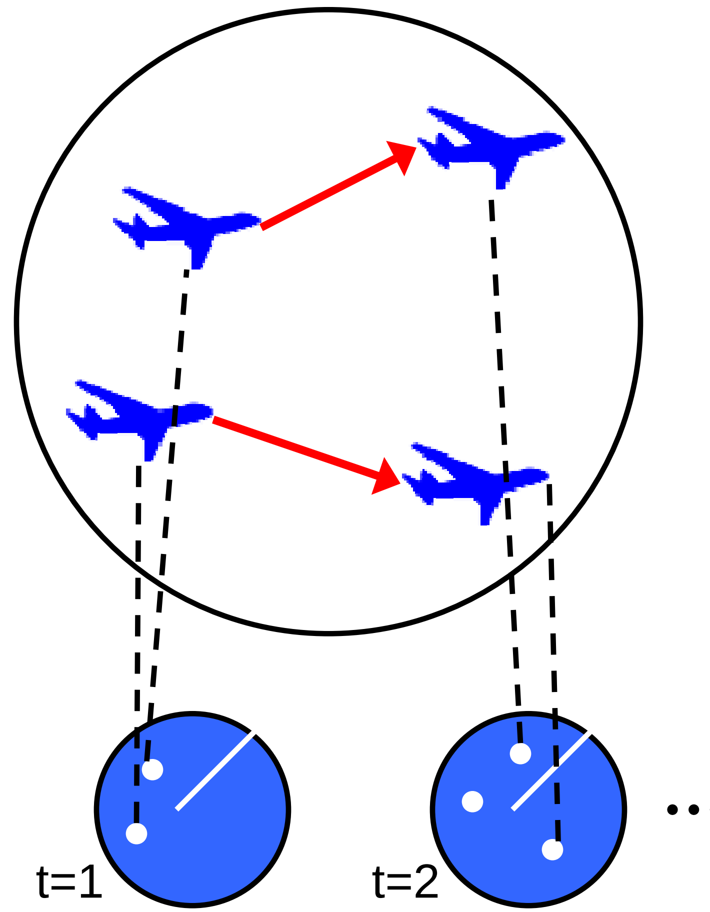
- Given observations, make inferences about underlying objects
- Difficulties:
  - Don’t know list of objects in advance
  - Don’t know when same object observed twice

(identity uncertainty / data association / record linkage)

Unknown Objects: Applications



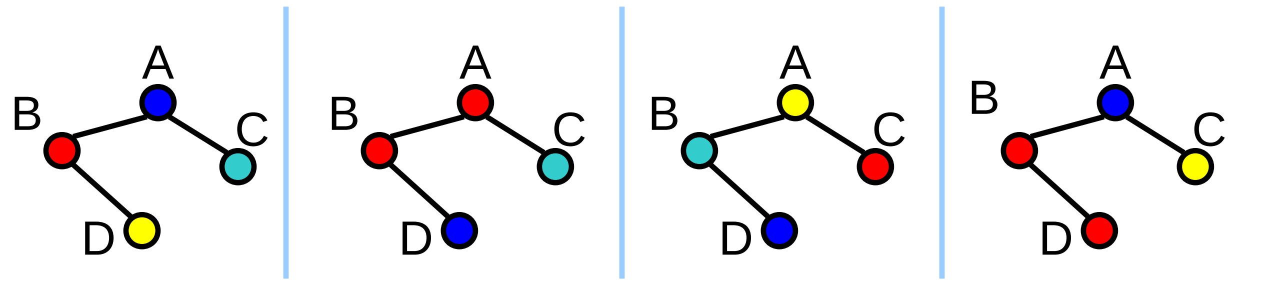
Citation Matching



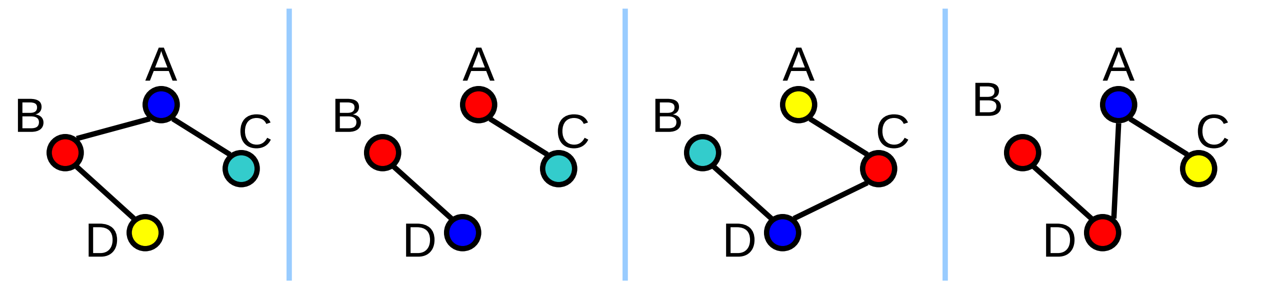
Multi-Target Tracking

Levels of Uncertainty

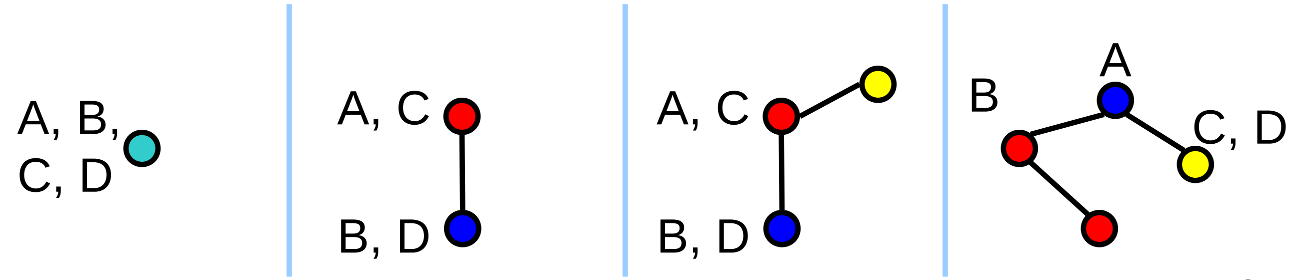
Attribute Uncertainty



Relational Uncertainty



Unknown Objects

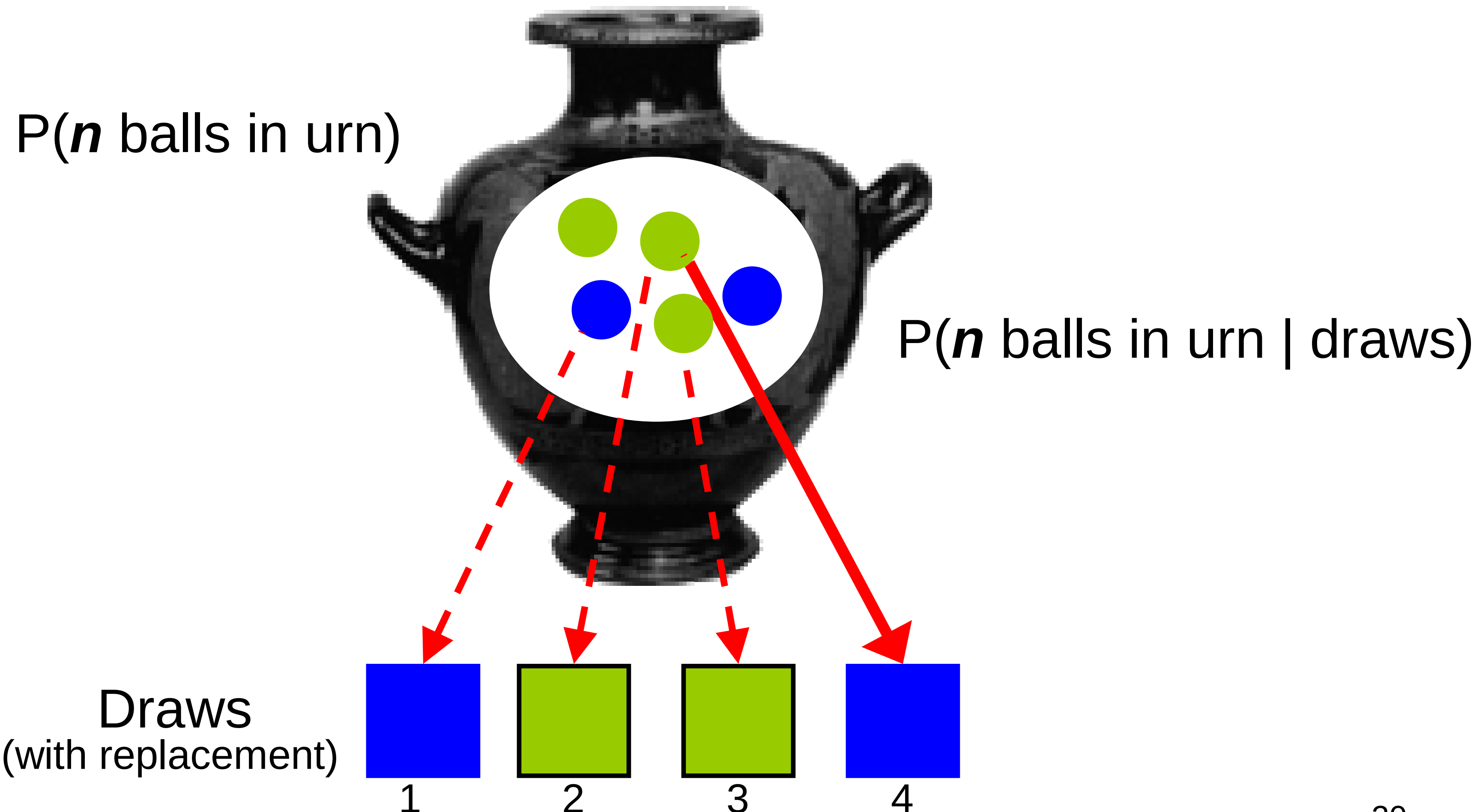


Bayesian Logic (BLOG)

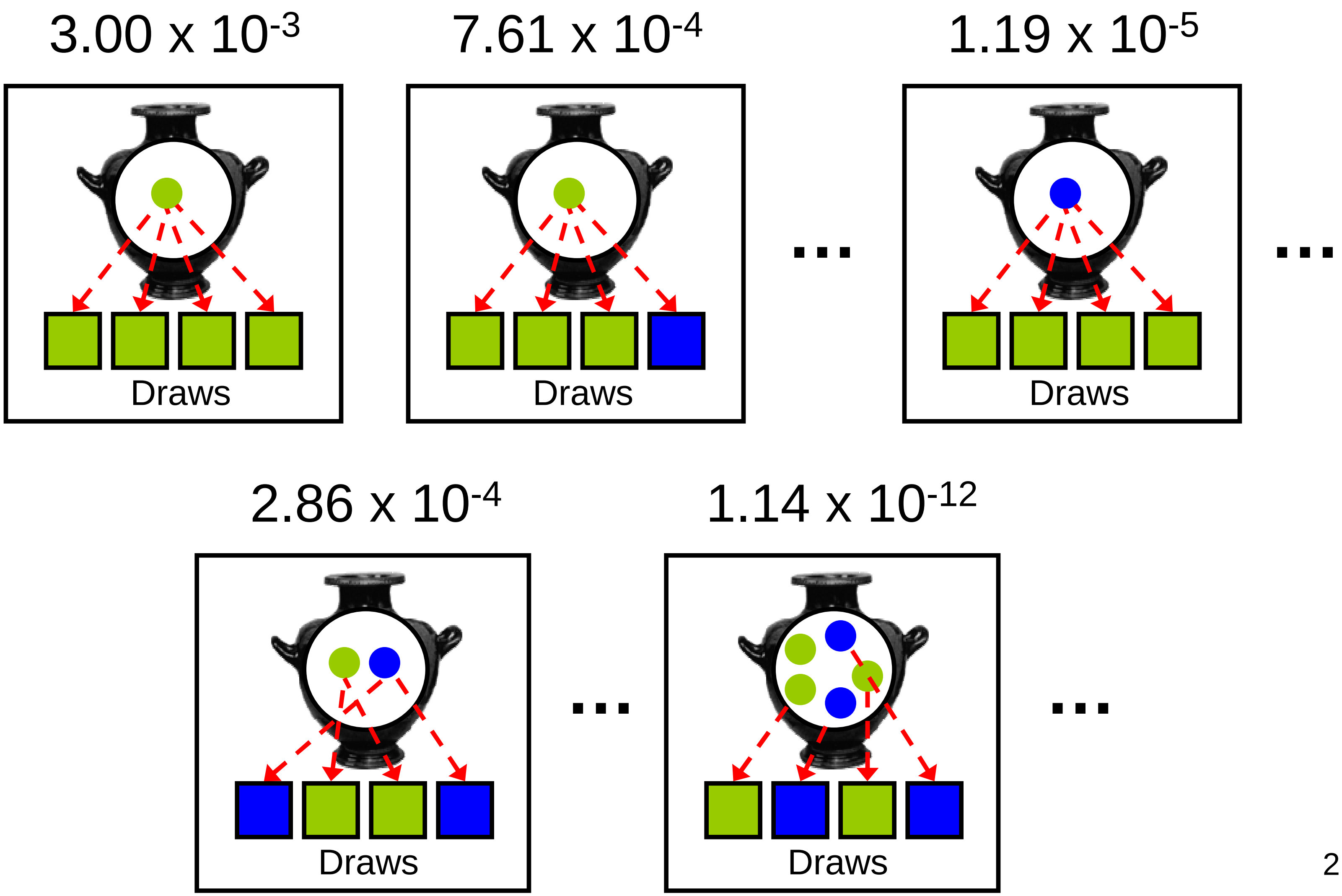
[Milch et al., SRL 2004; IJCAI 2005]

- Defines probability distribution over possible worlds with varying sets of objects
- Intuition: Stochastic generative process with two kinds of steps:
  - Set the value of a function on a tuple of arguments
  - Add some number of objects to the world

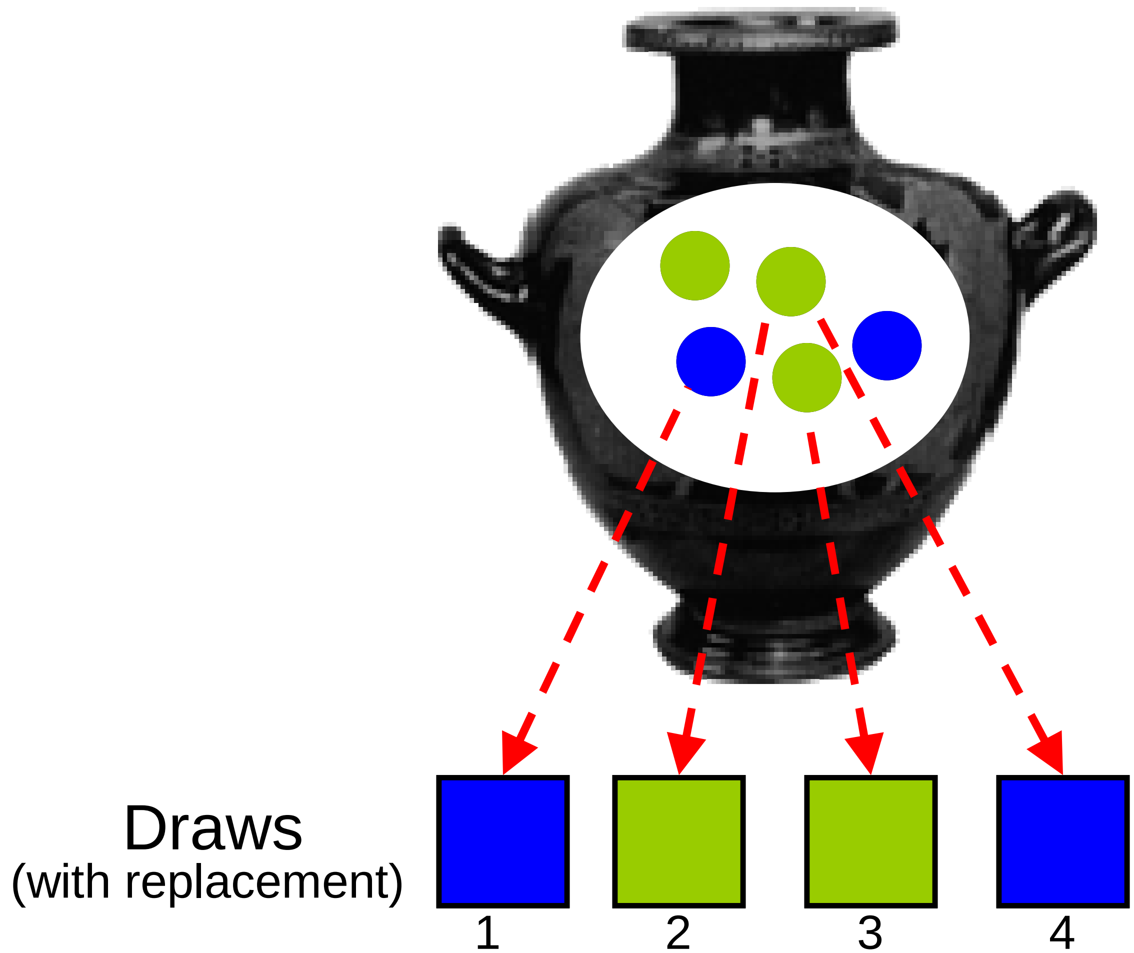
Simple Example: Balls in an Urn



Possible Worlds



Generative Process for Possible Worlds



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) ~ UniformChoice({Ball b});

ObsColor(d)
    if (BallDrawn(d) != null) then
        ~ NoisyCopy(TrueColor(BallDrawn(d)));
```

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) ~ UniformChoice({Ball b});

ObsColor(d)
    if (BallDrawn(d) != null) then
        ~ NoisyCopy(TrueColor(BallDrawn(d)));
```

header

number statement

dependency statements

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;
```

Identity uncertainty: BallDrawn(Draw1) $\stackrel{?}{=}$ BallDrawn(Draw2)

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

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

ObsColor(d)
    if (BallDrawn(d) != null) then
        ~ NoisyCopy(TrueColor(BallDrawn(d)));
```

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) ~ UniformChoice({Ball b});

ObsColor(d)
    if (BallDrawn(d) != null) then
        ~ NoisyCopy(TrueColor(BallDrawn(d)));
```

Arbitrary conditional probability distributions

CPD arguments

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) ~ UniformChoice({Ball b});

ObsColor(d)
    if (BallDrawn(d) != null) then
        ~ NoisyCopy(TrueColor(BallDrawn(d)));
```

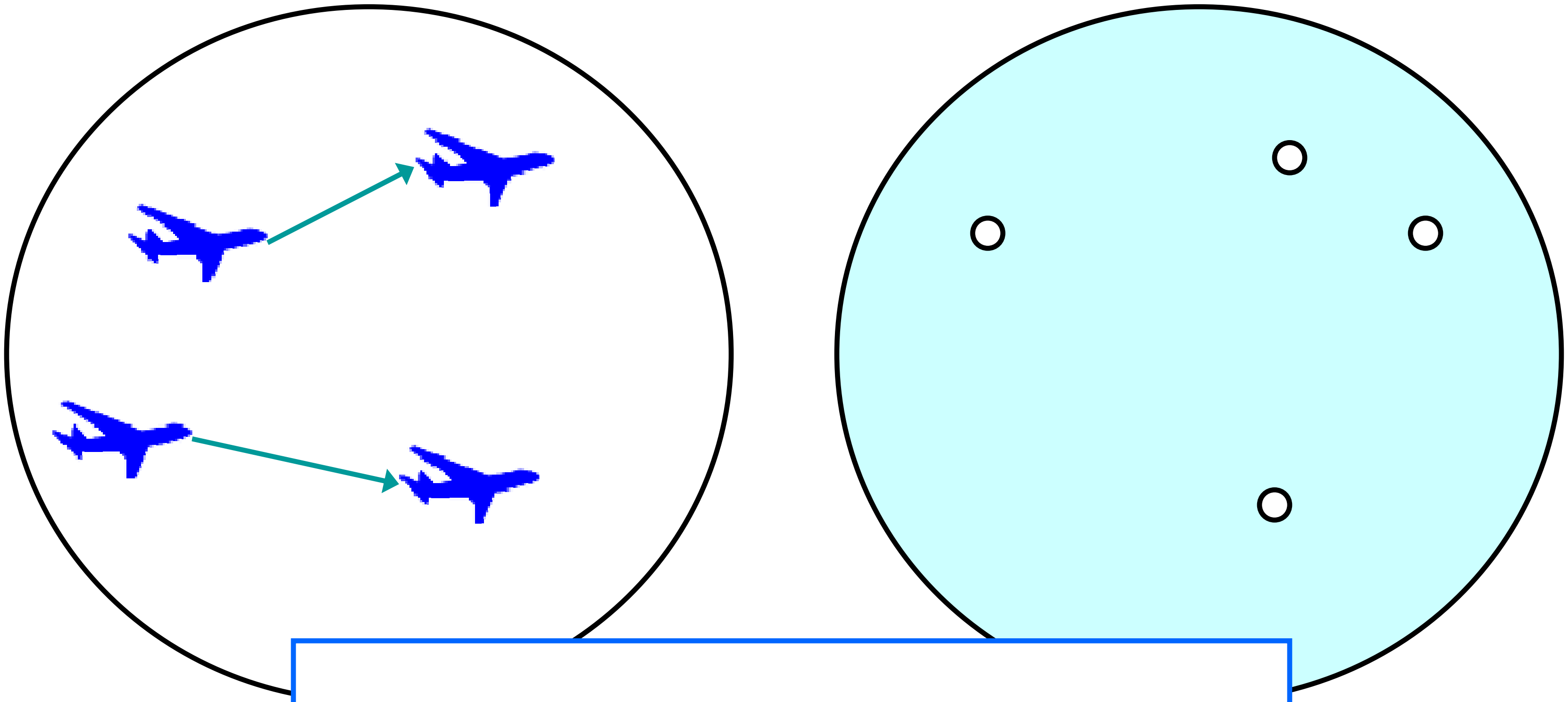
Context-specific dependence

Syntax of Dependency Statements

RetType Function(*ArgType*$_1$ x$_1$, ..., *ArgType*$_k$ x$_k$)
if *Cond*$_1$ then ~ *ElemCPD*$_1$(*Arg*$_{1,1}$, ..., *Arg*$_{1,m}$)
elseif *Cond*$_2$ then ~ *ElemCPD*$_2$(*Arg*$_{2,1}$, ..., *Arg*$_{2,m}$)
...
else ~ *ElemCPD*$_n$(*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
- Number statements: same except that their headers have the form #<*Type*>

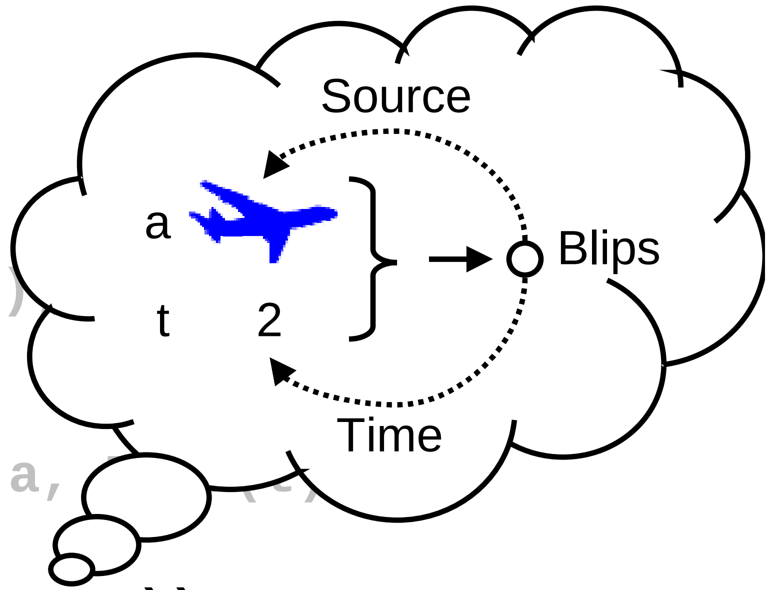
Generative Process for Aircraft Tracking

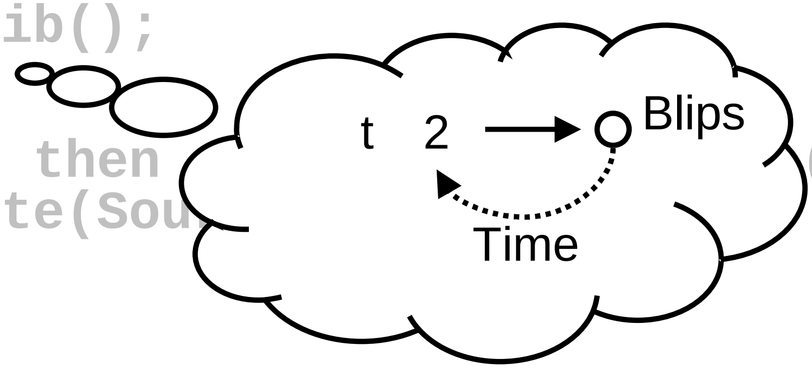


Existence of radar blips depends on existence and locations of aircraft

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,

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

#Blip(Time = t)
  ~ NumFalseAlarmsDistrib();

ApparentPos(r)
  if (Source(r) = null) then                ()
  else ~ ObsDistrib(State(Sou
```





Declarative Semantics

- What is the set of possible worlds?
- What is the probability distribution over worlds?

What Exactly Are the Objects?

- Objects are tuples that encode generation history
- Aircraft: (Aircraft, 1), (Aircraft, 2), …
- Blip from (Aircraft, 2) at time 8:

  (Blip, (Source, (Aircraft, 2)), (Time, 8), 1)

Basic Random Variables (RVs)

- For each number statement and tuple of generating objects, have RV for number of objects generated
- For each function symbol and tuple of arguments, have RV for function value
- Lemma: Full instantiation of these RVs uniquely identifies a possible world

Another Look at a BLOG Model

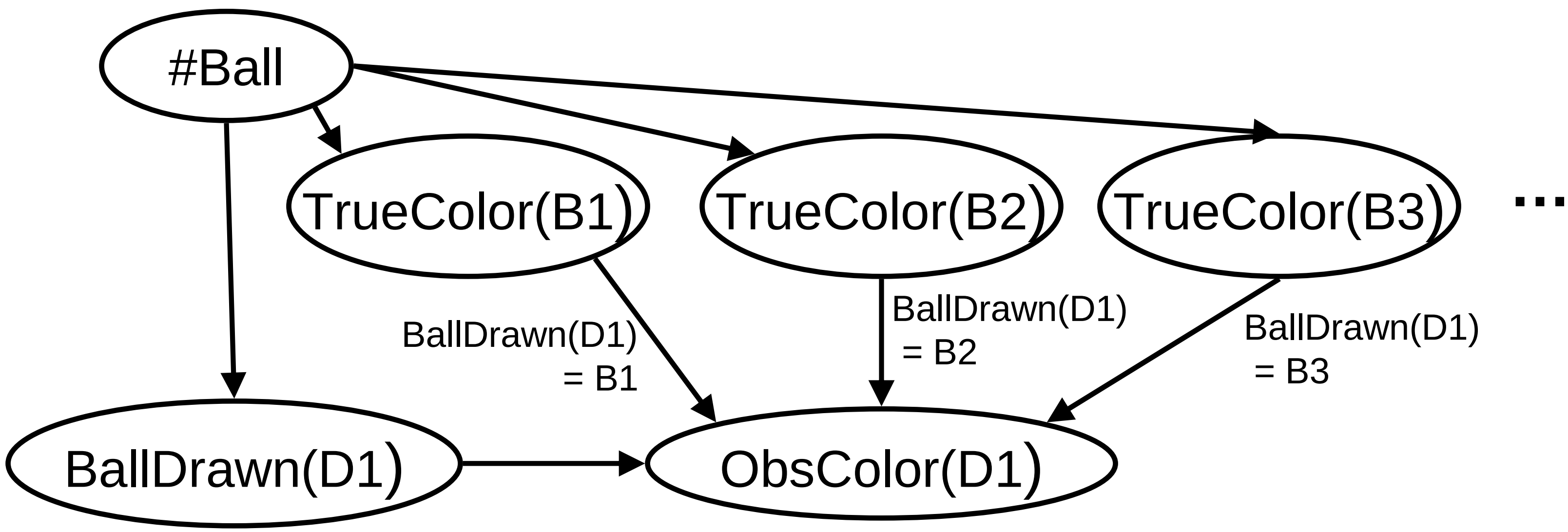
```
…
#Ball ~ Poisson[6]();

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

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

ObsColor(d)
    if !(BallDrawn(d) = null) then
        ~ NoisyCopy(TrueColor(BallDrawn(d)));
```

Dependency and number statements define CPDs for basic RVs

Semantics: Contingent BN

[Milch et al., AI/Stats 2005]

- Each BLOG model defines a contingent BN

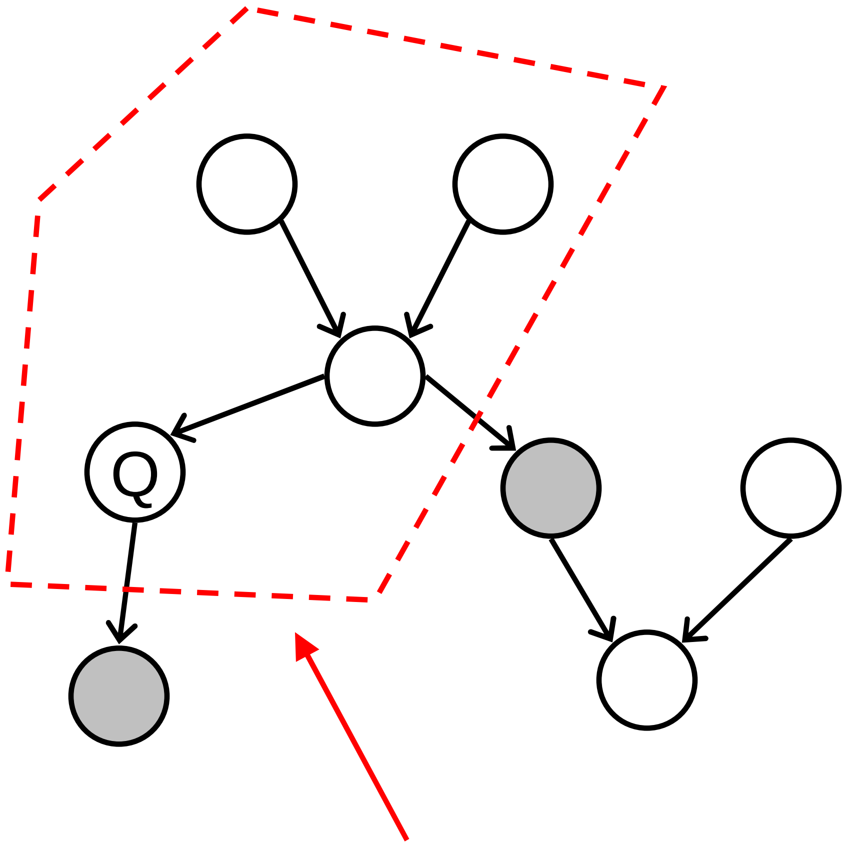


- *Theorem:* Every BLOG model that satisfies certain conditions (analogous to BN acyclicity) fully defines a distribution

Inference on BLOG Models

- Very easy to define models where exact inference is hopeless
- Sampling-based approximation algorithms:
  - Likelihood weighting
  - Markov chain Monte Carlo

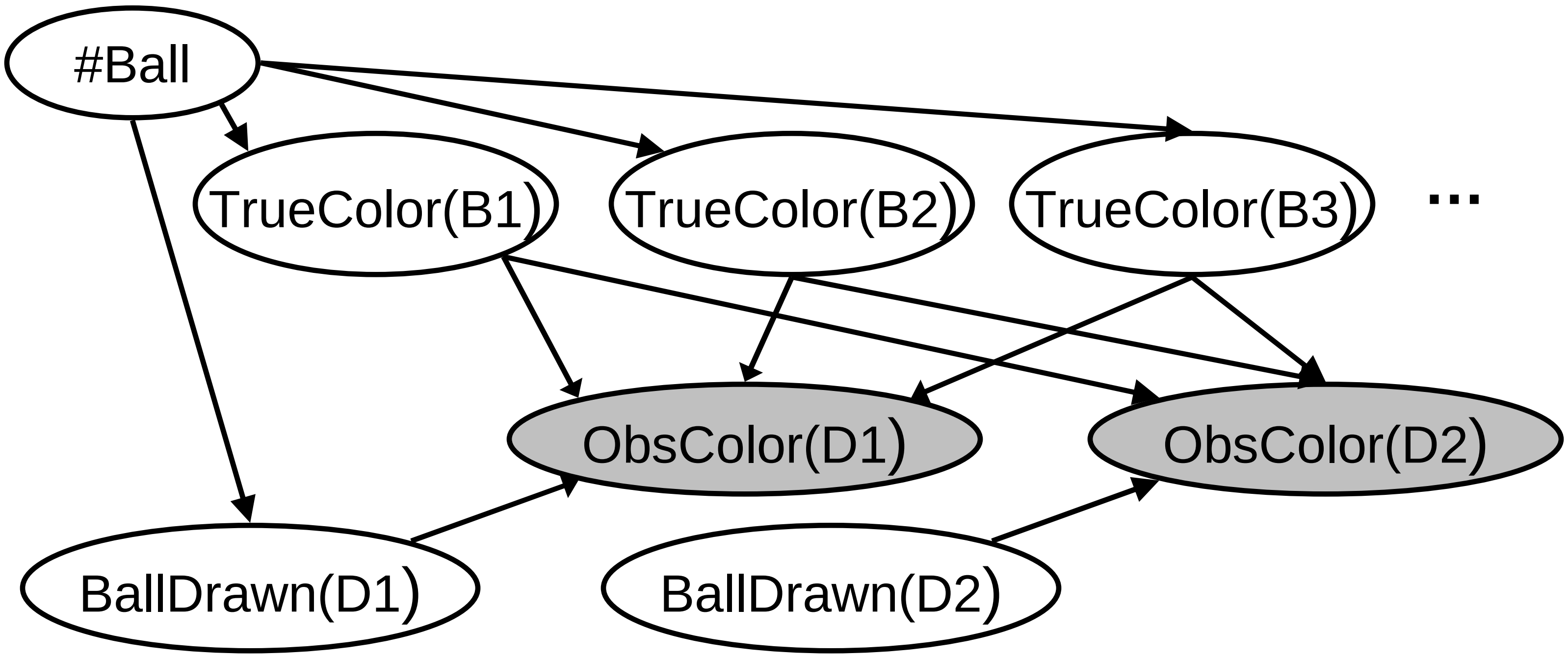
Likelihood Weighting (LW)



- Sample non-evidence nodes top-down
- Weight each sample by probability of observed evidence values given their parents
- Provably converges to correct posterior

Only need to sample ancestors of query and evidence nodes

Application to BLOG



- Given ObsColor variables, get posterior for #Ball
- Until we condition on BallDrawn(***d***), ObsColor(***d***) has infinitely many parents
- Solution: interleave sampling and relevance determination

[Milch *et al.*, AISTATS 2005]

LW for Urn and Balls

Evidence:

✓ ObsColor(Draw1) = Blue;
✓ ObsColor(Draw2) = Green;

Query:

✓ #Ball

Instantiation

#Ball = 7
BallDrawn(Draw1) = (Ball, 3)
TrueColor((Ball, 3)) = Blue
ObsColor(Draw1) = Blue;
BallDrawn(Draw2) = (Ball, 3)
ObsColor(Draw2) = Green;

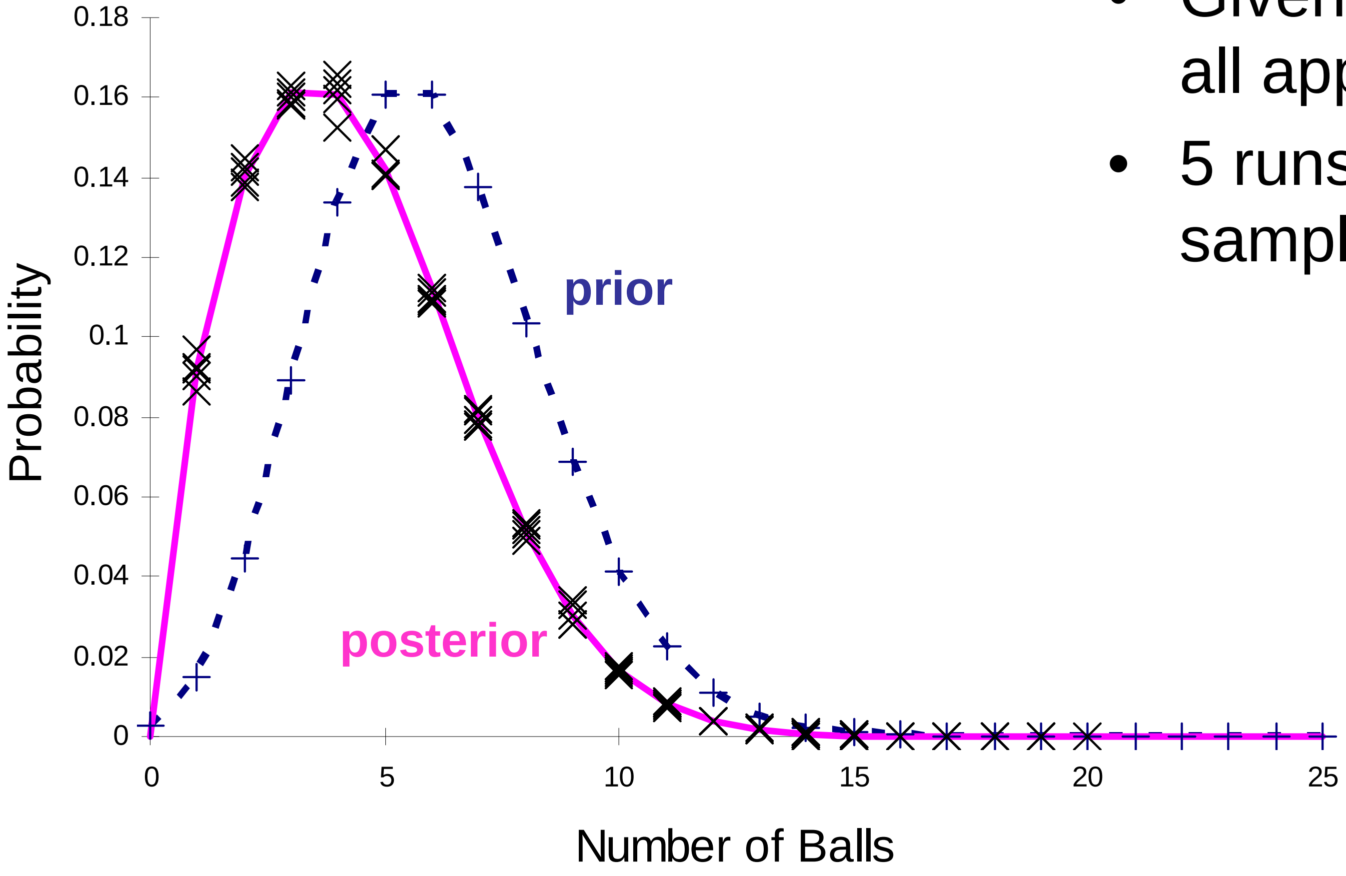
Weight: 1 x 0.8 x 0.2

Stack

TrueColor((Ball, 3))
BallDrawn(Draw2)
ObsColor(Draw2)
#Ball

```
#Ball ~ Poisson();
TrueColor(b) ~ TabularCPD();
BallDrawn(d) ~ UniformChoice({Ball b});
ObsColor(d)
    if !(BallDrawn(d) = null) then
        ~ TabularCPD(TrueColor(BallDrawn(d)));
```

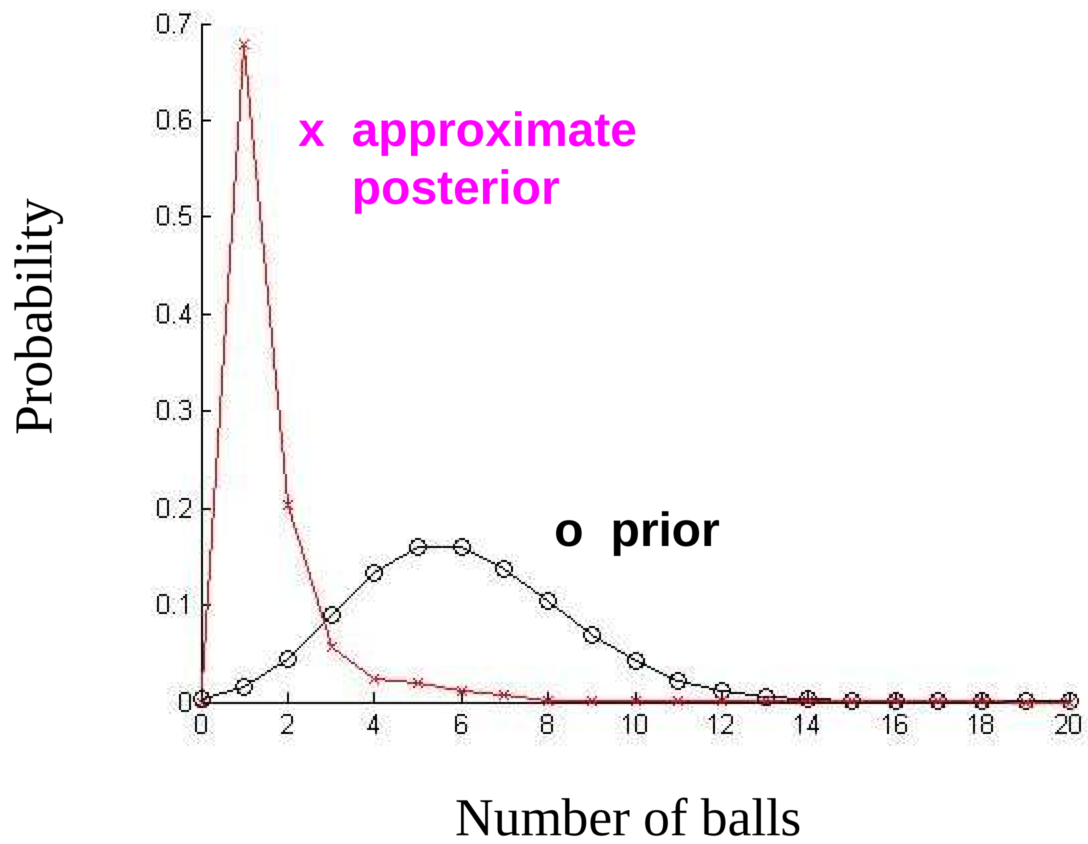
Examples of Inference



- Given 10 draws, all appearing blue
- 5 runs of 100,000 samples each

Examples of inference

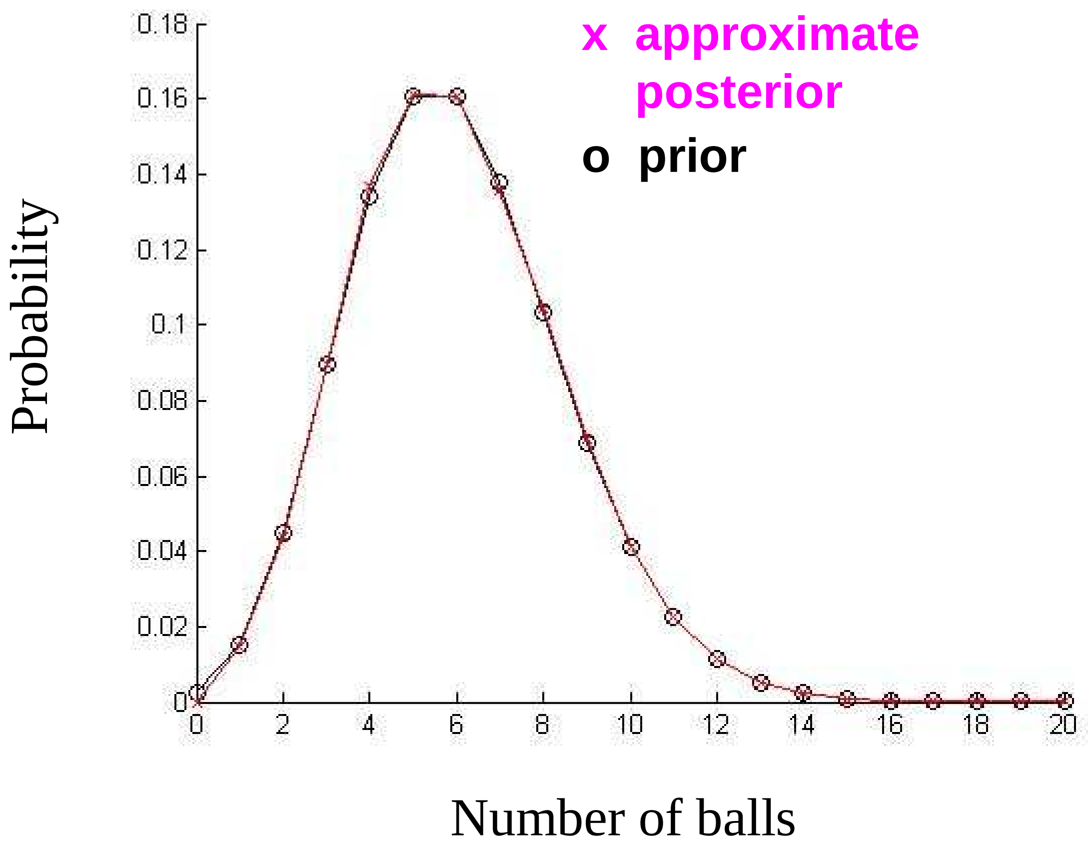
[Courtesy of Josh Tenenbaum]



- Ball colors: {Blue, Green, Red, Orange, Yellow, Purple, Black, White}
- Given 10 draws, all appearing Blue
- Runs of 100,000 samples each

Examples of inference

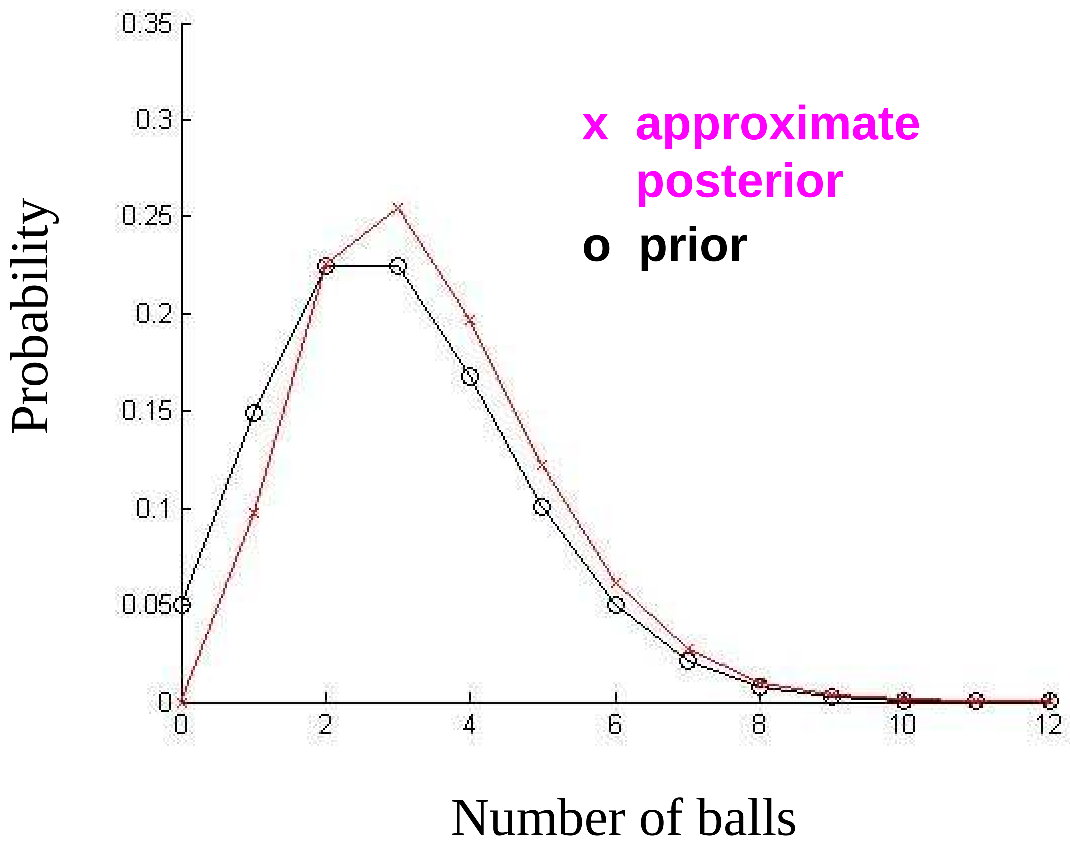
[Courtesy of Josh Tenenbaum]



- Ball colors: {Blue}
- Given 10 draws, all appearing Blue
- Runs of 100,000 samples each

Examples of inference

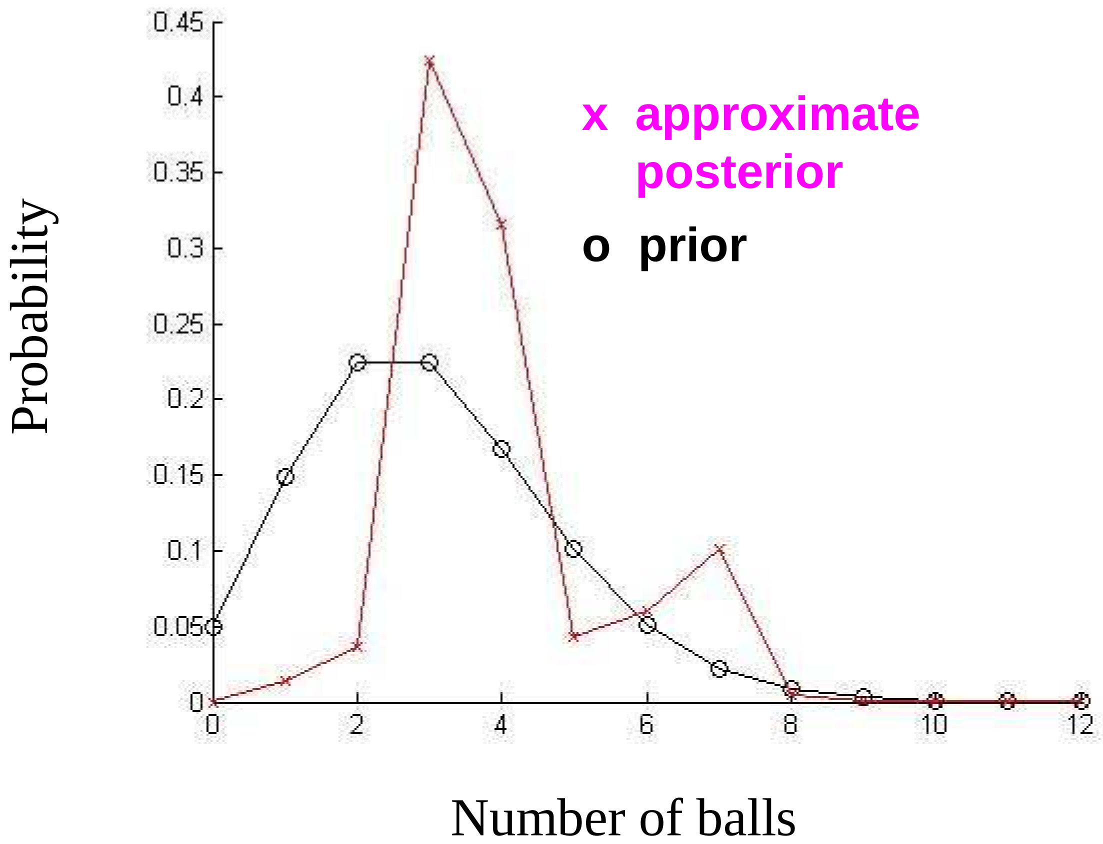
[Courtesy of Josh Tenenbaum]



- Ball colors: {Blue, Green}
- Given 3 draws: 2 appear Blue, 1 appears Green
- Runs of 100,000 samples each

Examples of inference

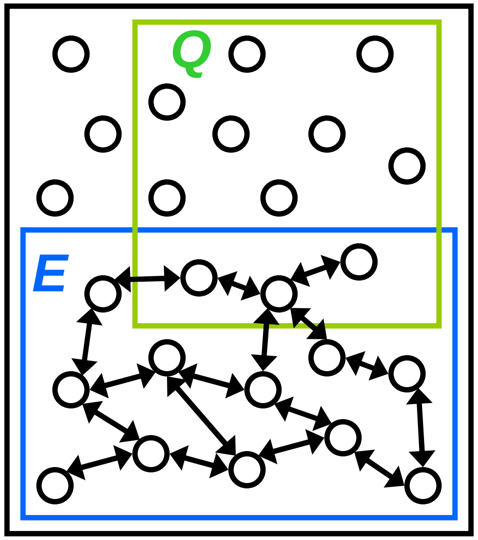
[Courtesy of Josh Tenenbaum]



- Ball colors: {Blue, Green}
- Given 30 draws: 20 appear Blue, 10 appear Green
- Runs of 100,000 samples each

More Practical Inference

- Drawback of likelihood weighting: as number of observations increases,
  - Sample weights become very small
  - A few high-weight samples tend to dominate
- More practical to use MCMC algorithms
  - Random walk over possible worlds
  - Find high-probability areas and stay there



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 $\boldsymbol{s}_{n+1} = \boldsymbol{s}_n$

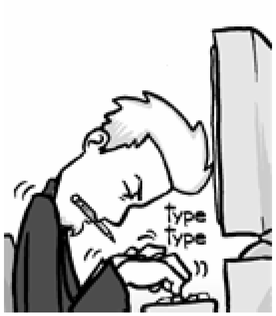
Stationary distribution is proportional to $\boldsymbol{p}(\boldsymbol{s})$

⇩

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

Toward General-Purpose Inference

- Without BLOG, each new application requires new code for:
  - Proposing moves
  - Representing MCMC states
  - Computing acceptance probabilities



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

General MCMC Engine

[Milch & Russell, UAI 2006]

Model
(in declarative language)

- 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?

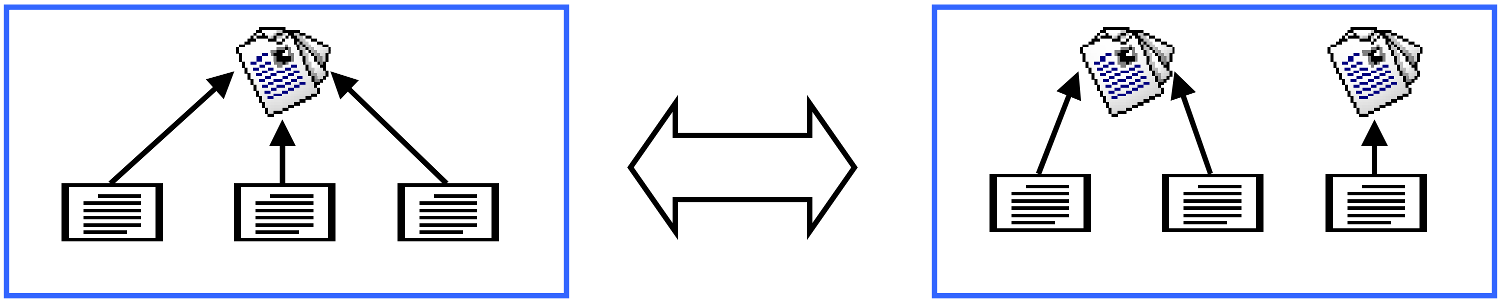
Example: Citation Model

```
guaranteed Citation Cit1, Cit2, Cit3, Cit4;
#Res ~ NumResearchersPrior();
String Name(Res r) ~ NamePrior();
#Pub ~ NumPubsPrior();
Res Author(Pub p) ~ Uniform({Res r});
String Title(Pub p) ~ TitlePrior();
Pub PubCited(Citation c) ~ Uniform({Pub p});
String Text(Citation c) ~ FormatCPD
       (Title(PubCited(c)), Name(Author(PubCited(c))));
```

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

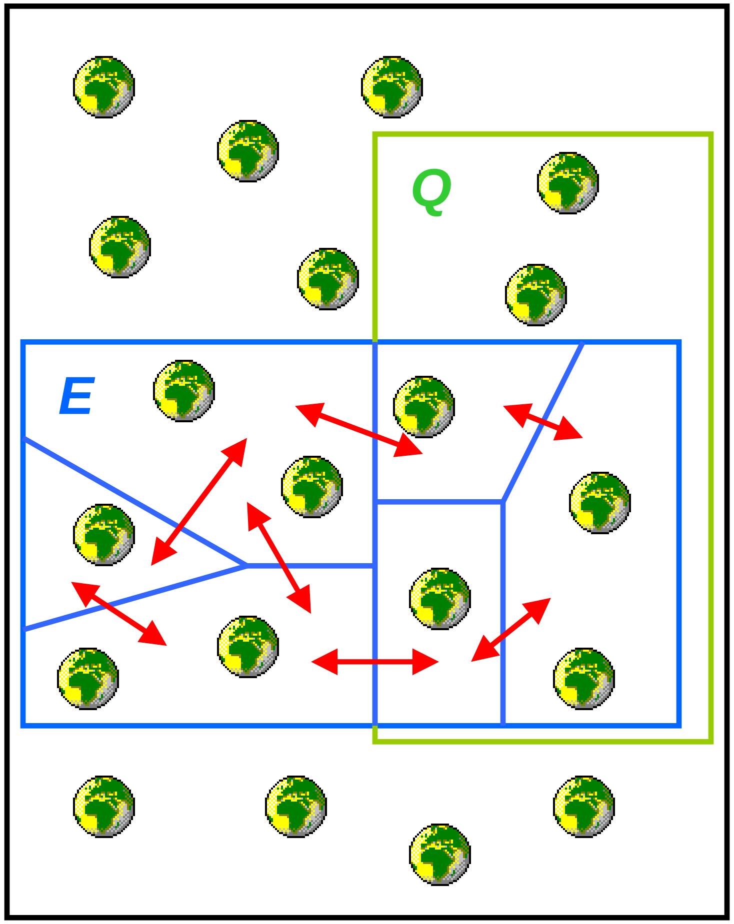
- Not complete instantiations!
  – No titles, author names for uncited publications
- States are partial instantiations of random variables

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

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

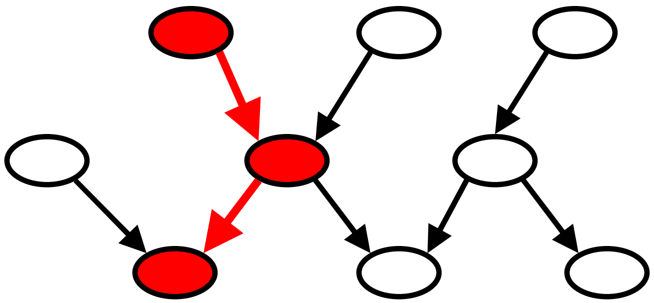
MCMC over Events

- Markov chain over events σ, with stationary distrib. proportional to ***p***(σ)
- *Theorem:* Fraction of visited events in ***Q*** converges to ***p***(***Q***|***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 instantiations that include all active parents of the variables they instantiate



- Then probability is product of CPDs:

$$p(\sigma) = \prod_{X \in \mathrm{vars}(\sigma)} p_X\left(\sigma(X) \mid \sigma(\mathrm{Pa}_\sigma(X))\right)$$

Computing Acceptance Probabilities Efficiently

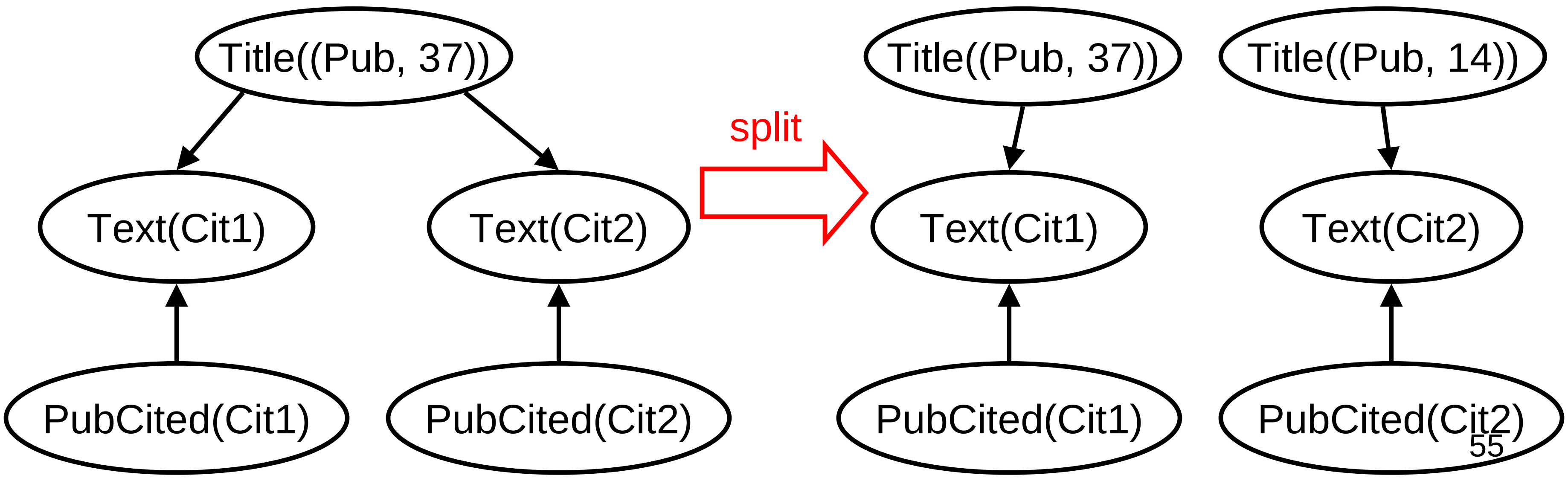
- First part of acceptance probability is:

$$\frac{p(\sigma')}{p(\sigma_n)} = \frac{\prod_{X \in \text{vars}(\sigma')} p_X\left(\sigma'(X) \mid \sigma'\left(\text{Pa}_{\sigma'}(X)\right)\right)}{\prod_{X \in \text{vars}(\sigma_n)} p_X\left(\sigma_n(X) \mid \sigma_n\left(\text{Pa}_{\sigma_\text{n}}(X)\right)\right)}$$

- If moves are local, most factors cancel
- Need to compute factors for $\boldsymbol{X}$ only if proposal changes $\boldsymbol{X}$ or one of $\text{Pa}_{\sigma_n}(X)$

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)

Learning BLOG Models

- Much larger class of dependency structures
  - If-then-else conditions
  - CPD arguments, which can be:
    - terms
    - set expressions, maybe containing conditions
- And we'd like to go further: invent new
  - Random functions, e.g., Colleagues(x, y)
  - Types of objects, e.g., Conferences
- Search space becomes extremely large

Summary

- First-order probabilistic models combine:
  - Probabilistic treatment of uncertainty
  - First-order generalization across objects
- PRMs
  - Define BN for any given relational skeleton
  - Can learn structure by local search
- BLOG
  - Expresses uncertainty about relational skeleton
  - Inference by MCMC over partial world descriptions
  - Learning is open problem