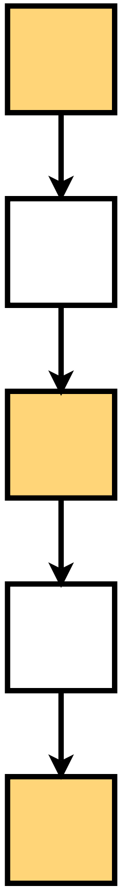
BLOG: Probabilistic Models with Unknown Objects

Milch et. al. 2005

574 Presentation - Brian Ferris

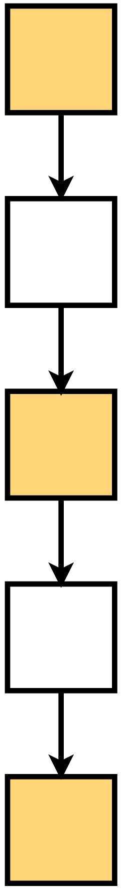


Overview

- Introduction
- Motivating Examples
- BLOG: Bayesian Logic
- Syntax and Semantics
- Inference

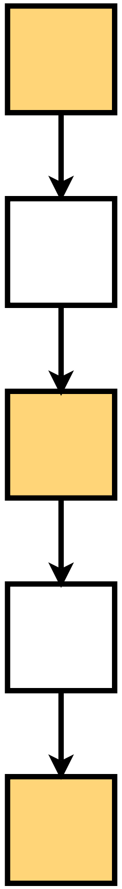
Introduction

- Existing *first-order probabilistic languages* attempt to model objects and relationships between them
- Such languages have difficulty in modeling unknown objects in a flexible way
- There are many interesting problems involving unknown objects



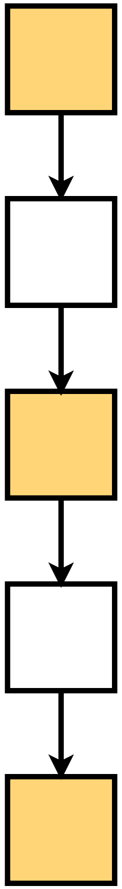
Example I

- An urn contains an unknown number of balls, which are equally likely to be blue or green
- Balls are drawn, observed (with 0.2 observation error), and replaced
- How many balls are in the urn? Was the same ball drawn twice?



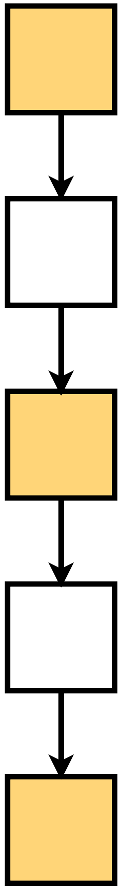
Example II

- An unknown number of aircraft are being tracked on radar
- Each radar blip gives the approximate position of an aircraft, but some blips are false positives, and some aircraft are not detected.
- What aircraft exist, and what are their trajectories?



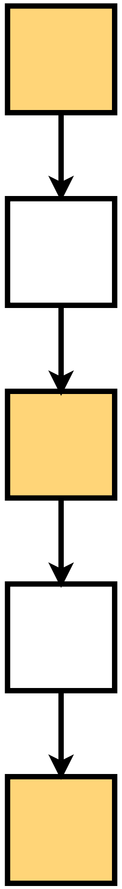
BLOG

- A language for defining probability distributions over outcomes with varying sets of objects
- Syntax similar to First Order Logic
- Describes a stochastic model for generating worlds



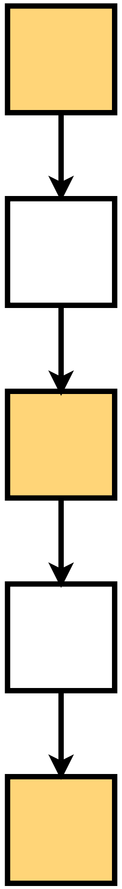
BLOG: Example 1

```
// Blog is typed
type Color; type Ball; type Draw;

// Random functions
random Color TrueColor(Ball);
random Ball BallDrawn(Draw);
random Color ObsColor(Draw);

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



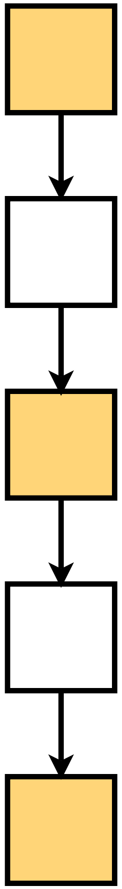
BLOG: Example 1

```
// Number of balls has a Poisson prior
#Ball ~ Poisson[6]();

// Both possible colors of a ball are equally likely
TrueColor(b) ~ TabularCPD[[0.5,0.5]];

// Balls are drawn with uniform probability from the urn
BallDrawn(d) ~ Uniform({Ball b});

// The observed color of a drawn ball is wrong with P=0.2
ObsColor(d)
    if( BallDrawn(d) != null ) then
        ~TabularCPD[[0.8,0.2][0.2,0.8]]
            (TrueColor(BallDrawn(d))
```



Syntax and Semantics

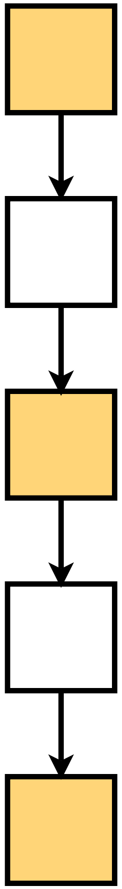
- In BLOG, everything is treaded as a *function*: constants are just functions that return true
- Functions are typed $(\tau 0, \tau 1 \ldots \tau k)$ where $\tau 0$ is the return type and $\tau 1 \ldots \tau k$ are the argument types



S&S: Types

```
// Object types
type Aircraft; type Blip;

// Built-in types for strings, numbers, and tuples
type String; type R5Vector;
```

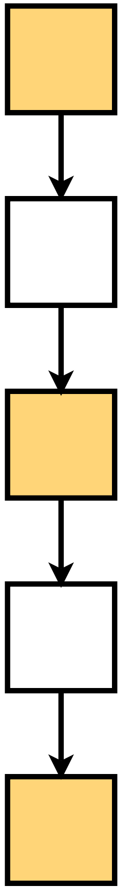
- The *type* keyword introduces the various types for a given model



S&S: Random Functions

```
// Random functions
random R6Vector State(Aircraft,NaturalNum);
random R3Vector ApparentPos(Blip);
```

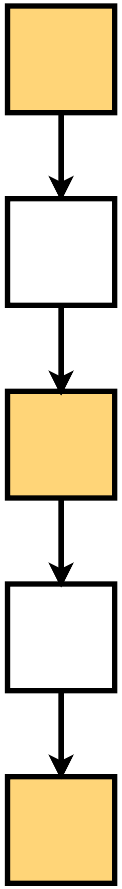
- The *random* keyword introduces a random function of the form τ0 f(τ1...τk) where “f” is the function name, τ1...τk are the argument types, and τ0 is the return type



S&S: Non-Random

```
// Non-Random functions
nonrandom NaturalNum Pred(NaturalNum);
```

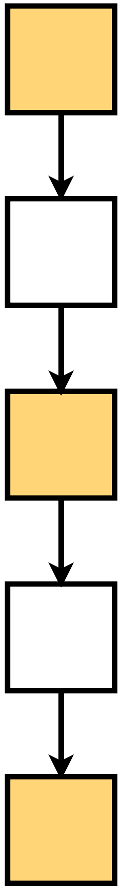
- The *nonrandom* keyword introduces a function whose interpretation is fixed in all possible world.
- Typing syntax is similar to random functions.



S&S: Dependencies

```
// Dependent function
State(a,t)
if t=0
        then ~ InitState()
        else ~ StateTransition(State(a,Pred(t)))
```

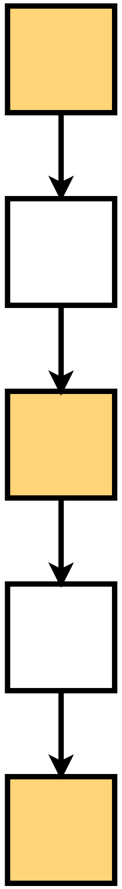
- Allows a level of flow control for functions
- Given example establishes an initial state and subsequent states as transitions from the preceding state



S&S: Generation

```
// Generator functions
generating Aircraft Source(Blip)
generating NaturalNum Time(Blip)

#Aircraft ~ NumAircraftDistrib();
#Blip: (Source,Time) ⇒ (a,t) ~ Detection(State(a,t))
```

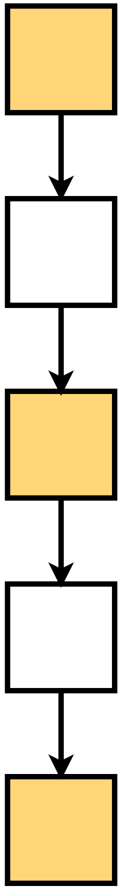
- Most powerful construct that specifies the generation of new objects
- A combination of Generator functions and Number functions



S&S: Generation

```
// Generator functions
generating Aircraft Source(Blip)
generating NaturalNum Time(Blip)

#Aircraft ~ NumAircraftDistrib();
#Blip: (Source,Time) ⇒ (a,t) ~ Detection(State(a,t))
```

- #τ : (g1..gk) ⇒ (x1..xk) ~ DistFunc(x)
- g1..gk are functions who accept objects of type τ.
- An object of type τ is with P determined by DistFunc(x) when objects o1..ok for g1..gk

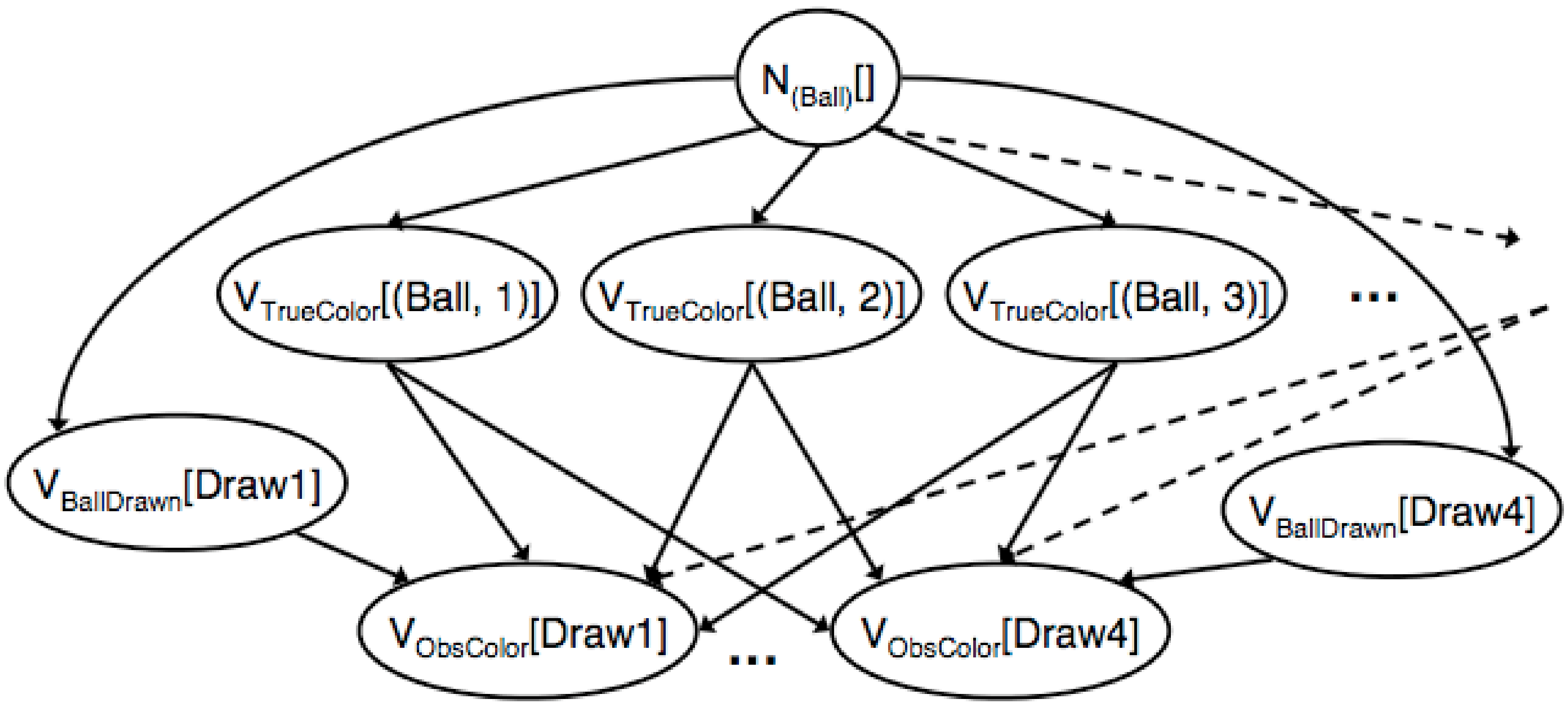


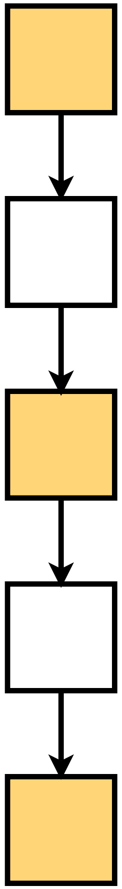
Inference

- For a given random variable (random function), we consider an instantiation σ over a set of RV vars(σ)
- $P(\sigma) = \prod X \in vars(\sigma) \quad px(\sigma x \mid \sigma pa(X))$’
  - px is the CPD for X
  - σpa is σ restricted to parents of X

Inference

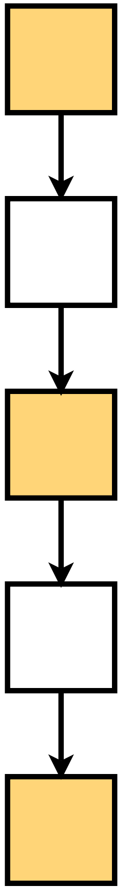
- In a Bayes Net for a BLOG model, the parent set is often infinite in size





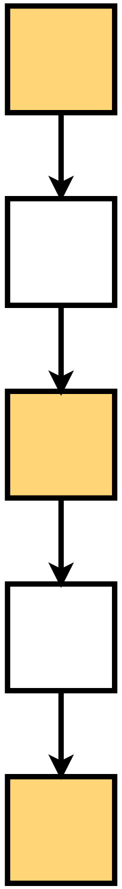
Inference

- Self-supporting instantiation
  - While the parent set may be infinite, not all entries are needed to calculate the CPD
  - If a given instantiation can be ordered such that Xn depends on only X1..Xn-1 for all n ≤ N, then... self-supporting
- See [Milch et al. 2005b] for proof regarding self-supporting instantiations with countably infinite random variables*



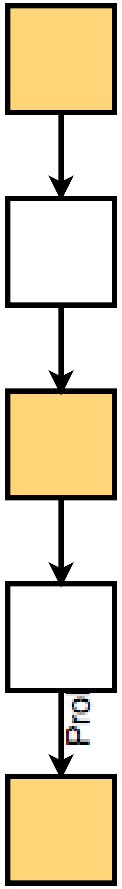
Inference

- Is this even decidable?
  - Yes, using rejection sampling
  - Very slow, but decidable
  - See termination criteria proof in the chapter
- Faster algorithm using likelihood weighting algorithm with backward chaining from the query and evidence nodes to avoid unneeded sampling

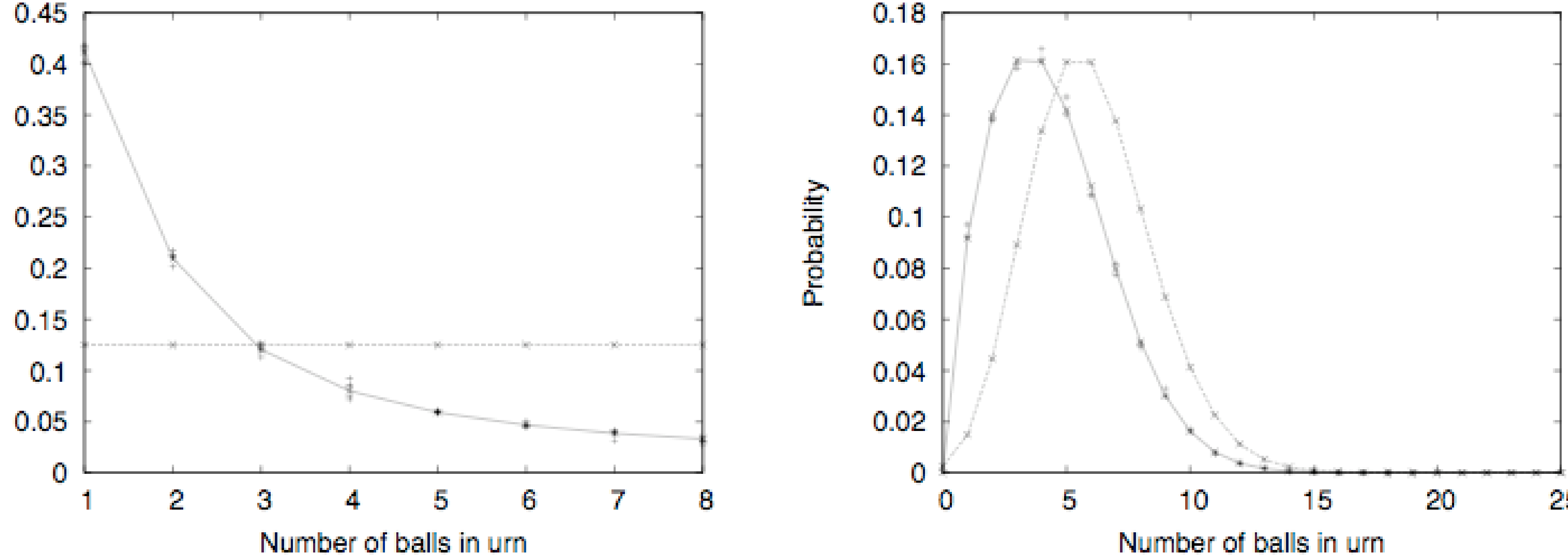


Inference

- Rejection Sampling
  - Start with initially empty σ
  - Augment as function dependencies are met
  - Continue until all query and evidence variables have been sampled
  - If consistent, increment Nq
- P(Q=q|e) is Nq/N



Results



Balls in urn example:
10 balls drawn, all blue,
with uniform (a) and poisson (b) priors