2ND SCHOOL ON FOPSS: LOGIC AND LEARNING

A tutorial on Bayesian learning

Daniel M. Roy

University of Toronto | Vector Institute

The theory of probability is at bottom nothing but common sense reduced to calculus; it makes one appreciate with exactness that which accurate minds feel with a sort of instinct, often without being able to account for it. — Pierre Simon Laplace

Based on

Freer, Roy, and Tenenbaum (2012). Towards common-sense reasoning via conditional simulation: Legacies of Turing in Artificial Intelligence. Turing's Legacy (ASL Lecture Notes in Logic).

The goal of this tutorial is to give a “Bayesian perspective” on “learning”.

- What is a “Bayesian perspective”?
- What is “learning”?

The Bayesian perspective

- Probability is no longer just a limiting relative frequency associated with independent identical experiments.
- Instead, probability represents *degree of belief*.
  - $\mathbb{P}\{$ I will fly to Vilnius tomorrow $\} = 0.97$.
  - $\mathbb{P}\{$ I will fly to Canada tomorrow $\} = 0.001$.
  - $\mathbb{P}\{$ I will fly to Canada and Vilnius tomorrow $\} = 0$.
  - $\mathbb{P}\{$ I will fly to Canada or Vilnius tomorrow $\} = 0.971$.
  - $\mathbb{P}\{$ Tom has flu $|$ Tom has fever and muscle aches in Winter $\} = 0.6$.
  - $\mathbb{P}\{$ Tom has flu $|$ Tom has fever and muscle aches in Summer $\} = 0.1$.
- Probabilities are *personal*.
- The key structures are *joint distributions* of multiple random variables.
  - $X =$ Patient has flu, $Y =$ has fever, $Z =$ has muscle aches, $S =$ It is Summer, $W =$ It is Winter.
  - A joint distribution is a specification of probabilities $\mathbb{P}\{X = x, Y = y, Z = z, S = s, W = w\}$ for every possible "event".

What is learning?

- If we do not know $\mathbb{P}$, we can attempt to "learn" it from data. This necessarily requires some strong assumptions and relates to the problem of induction.
  - *Learning to classify images*: Learn $\mathbb{P}\{\text{Label} \mid \text{Image}\}$ from dataset of labelled images.
  - *Learning to diagnosis*: Learn $\mathbb{P}\{\text{Disease, Symptoms}\}$ from past patient data, in order to infer diseases from symptoms in future patients.
- The distributions $\mathbb{P}$ above are generally assumed to come from a parametric family $\{\mathbb{P}_\theta\}$, e.g., $\theta \in \mathbb{R}^d$. The ("true" or "best") parameter θ^* is assumed unknown.
- An approach is "fully Bayesian" if one uses probability distributions to express uncertainty for all unknown quantities (such as θ^*), modeling them as random variables with prior distributions.
- In a "fully Bayesian" approach, learning is probabilistic inference, and thus everything is probabilistic inference.
- In contrast, in a frequentist approach, one would develop estimators for θ^* with good frequentist (sampling) properties.

UNIVERSAL STOCHASTIC INFERENCE

- The Bayesian framework is conceptually simple:
  - represent all knowledge by distributions
  - evidence incorporated by conditioning
- I will introduce the Bayesian framework using the computational framework of *universal stochastic inference* wherein
  - distributions are represented by *programs*
  - evidence is represented by *predicates*
  - conditioning is a higher-order *procedure*
- Programs can represent distributions by being *simulators*
  - a program $\mathtt{P}$ represents a distribution μ by simulating μ, i.e., generating a random output X whose distribution is μ

Related frameworks

This perspective is closely related to

- Probabilistic programming
- Approximate Bayesian Computation (ABC)
- Implicit Generative Models

STRUCTURE OF THE TUTORIAL

The tutorial centers around a higher-order procedure, called `QUERY`, which implements a simple, though generic, form of probabilistic conditioning.

- `QUERY` operates on complex probabilistic models that are themselves represented by programs
- By using `QUERY` appropriately, one can describe various forms of inference, learning, and decision-making.
- We will introduce inference, learning, and decision making through an extended example of medical diagnosis, a classic AI problem.
- Elusive "common-sense" behavior arises *implicitly* from past experience and models of causal structure and goals, rather than explicitly via rules or purely deductive reasoning.

Example: simple probabilistic Python program

```
1 def binomial(n, p):
2     return sum( [bernoulli(p) for i in range(n)] )
```

- *returns* a **random integer** in $\{0, \ldots, n\}$.
- *defines* a **family of distributions** on $\{0, \ldots, n\}$, in particular, the *Binomial family*.
- *represents* a **statistical model** of

  the # of successes among
  n *independent and identical experiments*

Example: simple probabilistic Python program

```
1 def binomial(n, p):
2     return sum( [bernoulli(p) for i in range(n)] )
```

- *returns* a **random integer** in $\{0, \ldots, n\}$.
- *defines* a **family of distributions** on $\{0, \ldots, n\}$, in particular, the *Binomial family*.
- *represents* a **statistical model** of

  the # of successes among
  n *independent and identical experiments*

Example: simple probabilistic Python program

```
1 def binomial(n, p):
2     return sum( [bernoulli(p) for i in range(n)] )
```

- *returns* a **random integer** in $\{0, \ldots, n\}$.
- *defines* a **family of distributions** on $\{0, \ldots, n\}$, in particular, the *Binomial family*.
- *represents* a **statistical model** of

  the # of successes among
  n *independent and identical experiments*

Example: simple probabilistic Python program

```
1 def binomial(n, p):
2     return sum( [bernoulli(p) for i in range(n)] )
```

- *returns* a **random integer** in $\{0, \ldots, n\}$.
- *defines* a **family of distributions** on $\{0, \ldots, n\}$, in particular, the *Binomial family*.
- *represents* a **statistical model** of

  the # of successes among
  n *independent and identical experiments*

Example: simple probabilistic Python program

```
1  def binomial(n, p):
2      return sum( [bernoulli(p) for i in range(n)] )
```

Example: simple probabilistic Python program

```
1  def binomial(n, p):
2      return sum( [bernoulli(p) for i in range(n)] )
3  def randomized_trial()
4
5
6      return ( binomial(100, p_control),
7               binomial(10,  p_treatment) )
```

EXAMPLE: SIMPLE PROBABILISTIC PYTHON PROGRAM

```
1  def binomial(n, p):
2      return sum( [bernoulli(p) for i in range(n)] )

3  def randomized_trial()
4      p_control   = uniform(0,1)   # prior
5      p_treatment = uniform(0,1)   # prior
6      return ( binomial(100, p_control),
7               binomial(10,  p_treatment) )
```

EXAMPLE: SIMPLE PROBABILISTIC PYTHON PROGRAM

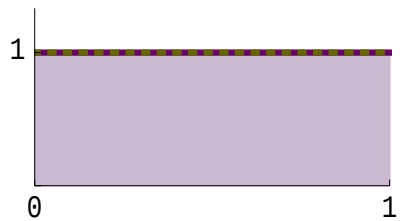
```
1  def binomial(n, p):
2      return sum( [bernoulli(p) for i in range(n)] )

3  def randomized_trial()
4      p_control   = uniform(0,1)  # prior
5      p_treatment = uniform(0,1)  # prior
6      return ( binomial(100, p_control),
7               binomial(10,  p_treatment) )
```

represents a **Bayesian model** of a randomized trial.

EXAMPLE: SIMPLE PROBABILISTIC PYTHON PROGRAM

```
1  def binomial(n, p):
2      return sum( [bernoulli(p) for i in range(n)] )

3  def randomized_trial()
4      p_control   = uniform(0,1)   # prior
5      p_treatment = uniform(0,1)   # prior
6      return ( binomial(100, p_control),
7               binomial(10,  p_treatment) )
```

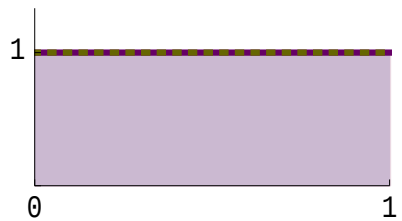
represents a **Bayesian model** of a randomized trial.



EXAMPLE: SIMPLE PROBABILISTIC PYTHON PROGRAM

```
1  def binomial(n, p):
2      return sum( [bernoulli(p) for i in range(n)] )

3  def randomized_trial()
4      p_control   = uniform(0,1)   # prior
5      p_treatment = uniform(0,1)   # prior
6      return ( binomial(100, p_control),
7               binomial(10,  p_treatment) )
```

represents a **Bayesian model** of a randomized trial.

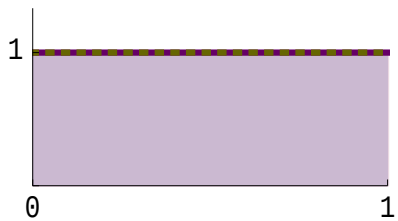


simulation $\longrightarrow$

Example: simple probabilistic Python program

```
1  def binomial(n, p):
2      return sum( [bernoulli(p) for i in range(n)] )

3  def randomized_trial()
4      p_control   = uniform(0,1)   # prior
5      p_treatment = uniform(0,1)   # prior
6      return ( binomial(100, p_control),
7               binomial(10,  p_treatment) )
```

represents a **Bayesian model** of a randomized trial.

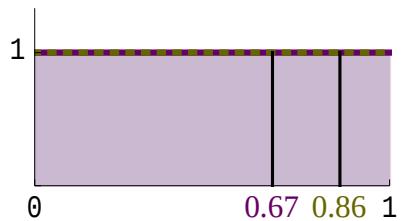


simulation $\longrightarrow$ $(71, 9)$

Example: simple probabilistic Python program

```
1  def binomial(n, p):
2      return sum( [bernoulli(p) for i in range(n)] )

3  def randomized_trial()
4      p_control   = uniform(0,1)   # prior
5      p_treatment = uniform(0,1)   # prior
6      return ( binomial(100, p_control),
7               binomial(10,  p_treatment) )
```

represents a **Bayesian model** of a randomized trial.

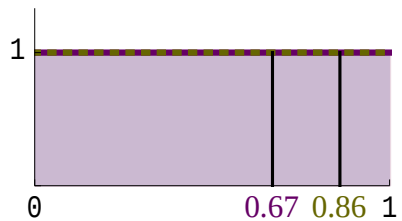


simulation $\longrightarrow$ $(71, 9)$

Example: simple probabilistic Python program

```
1  def binomial(n, p):
2      return sum( [bernoulli(p) for i in range(n)] )

3  def randomized_trial()
4      p_control   = uniform(0,1)   # prior
5      p_treatment = uniform(0,1)   # prior
6      return ( binomial(100, p_control),
7               binomial(10,  p_treatment) )
```

represents a **Bayesian model** of a randomized trial.



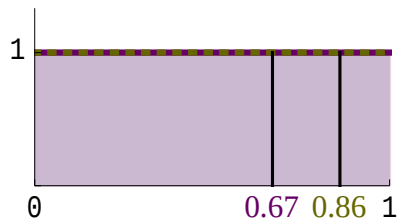
simulation $\longrightarrow$ $(71, 9)$

$(71, 9)$

Example: simple probabilistic Python program

```
1  def binomial(n, p):
2      return sum( [bernoulli(p) for i in range(n)] )

3  def randomized_trial()
4      p_control   = uniform(0,1)   # prior
5      p_treatment = uniform(0,1)   # prior
6      return ( binomial(100, p_control),
7               binomial(10,  p_treatment) )
```

represents a **Bayesian model** of a randomized trial.



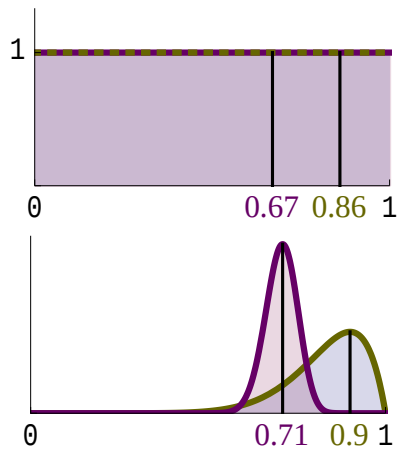
simulation $\longrightarrow$ $(71, 9)$

inference $\longleftarrow$ $(71, 9)$

Example: simple probabilistic Python program

```
1  def binomial(n, p):
2      return sum( [bernoulli(p) for i in range(n)] )

3  def randomized_trial()
4      p_control   = uniform(0,1)   # prior
5      p_treatment = uniform(0,1)   # prior
6      return ( binomial(100, p_control),
7               binomial(10,  p_treatment) )
```

represents a **Bayesian model** of a randomized trial.



simulation $\longrightarrow$ $(71, 9)$

inference $\longleftarrow$ $(71, 9)$

INTRODUCING QUERY

```
1  def QUERY(guesser,checker):
2      # guesser: Unit -> S
3      # predicate : S -> Boolean
4      accept = False
5      while (not accept)
6          guess = guesser()
7          accept = checker(guess)
8      return guess
```

UNDERSTANDING QUERY

```
1  def QUERY(guesser,checker):
2      # guesser: Unit -> S
3      # predicate : S -> Boolean
4      accept = False
5      while (not accept)
6          guess = guesser()
7          accept = checker(guess)
8      return guess
```

UNDERSTANDING QUERY

```
1  def QUERY(guesser,checker):
2      # guesser: Unit -> S
3      # predicate : S -> Boolean
4      accept = False
5      while (not accept)
6          guess = guesser()
7          accept = checker(guess)
8      return guess
```

As a first step towards understanding QUERY, consider the trivial predicate

```
lambda (_): True
```

Then guesser() has the same meaning (distributional semantics) as

```
QUERY(guesser, lambda (_): True)
```

UNDERSTANDING QUERY

```
1  def QUERY(guesser,checker):
2      # guesser: Unit -> S
3      # predicate : S -> Boolean
4      accept = False
5      while (not accept)
6          guess = guesser()
7          accept = checker(guess)
8      return guess
```

UNDERSTANDING QUERY

```
1  def QUERY(guesser,checker):
2      # guesser: Unit -> S
3      # predicate : S -> Boolean
4      accept = False
5      while (not accept)
6          guess = guesser()
7          accept = checker(guess)
8      return guess
```

Consider a slightly more interesting example:

```
def N():
    return uniformInt(range(1,180))
```

and consider the predicate

```
def div235(n):
    return isDivBy(n,2) or isDivBy(n,3) or isDivBy(n,5)
```

What is the meaning of `QUERY(N, div235)` ?

CONDITIONING AS A HIGHER-ORDER PROCEDURE

```
1  def QUERY(guesser,checker):
2      # guesser: Unit -> S
3      # predicate : S -> Boolean
4      accept = False
5      while (not accept)
6          guess = guesser()
7          accept = checker(guess)
8      return guess
```

represents the higher order operation of **conditioning**. When `checker` is deterministic, then `QUERY` denotes the map

$$(P, \mathbf{1}_A) \mapsto P(\,\cdot\,|A) := \frac{P(\,\cdot\,\cap A)}{P(A)}. \tag{1}$$

`QUERY` halts with probability 1 provided $P(A) > 0$.

The stochastic inference problem

THE STOCHASTIC INFERENCE PROBLEM

INPUT: `guesser` and `checker` probabilistic programs.

THE STOCHASTIC INFERENCE PROBLEM

INPUT: `guesser` and `checker` probabilistic programs.

OUTPUT: a sample from the same distribution as the program

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

THE STOCHASTIC INFERENCE PROBLEM

INPUT: `guesser` and `checker` probabilistic programs.

OUTPUT: a sample from the same distribution as the program

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

This computation captures **Bayesian statistical inference**.

THE STOCHASTIC INFERENCE PROBLEM

INPUT: `guesser` and `checker` probabilistic programs.

OUTPUT: a sample from the same distribution as the program

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

This computation captures **Bayesian statistical inference**.

"prior" distribution $\longleftrightarrow$ distribution of `guesser()`

"likelihood(g)" $\longleftrightarrow$ $\mathbb{P}\{$ `checker(`g`)` is `True` $\}$

"posterior" distribution $\longleftrightarrow$ distribution of return value

Example: inferring bias of a coin

EXAMPLE: INFERRING BIAS OF A COIN

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

EXAMPLE: INFERRING BIAS OF A COIN

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Given $x_1, \ldots, x_n \in \{0, 1\}$,
report probability of $x_{n+1} = 1$?

EXAMPLE: INFERRING BIAS OF A COIN

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Given $x_1, \ldots, x_n \in \{0, 1\}$,
report probability of $x_{n+1} = 1$? E.g., $0, 0, 1, 0, 0$

EXAMPLE: INFERRING BIAS OF A COIN

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Given $x_1, \ldots, x_n \in \{0, 1\}$,
report probability of $x_{n+1} = 1$? E.g., $0, 0, 1, 0, 0$

```
def guesser():
  p = uniform()
  return p
```

EXAMPLE: INFERRING BIAS OF A COIN

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Given $x_1, \ldots, x_n \in \{0, 1\}$,
report probability of $x_{n+1} = 1$? E.g., $0, 0, 1, 0, 0$

```
def guesser():
  p = uniform()
  return p

def checker(p):
  return [0,0,1,0,0] == bernoulli(p,5)
```

EXAMPLE: INFERRING BIAS OF A COIN

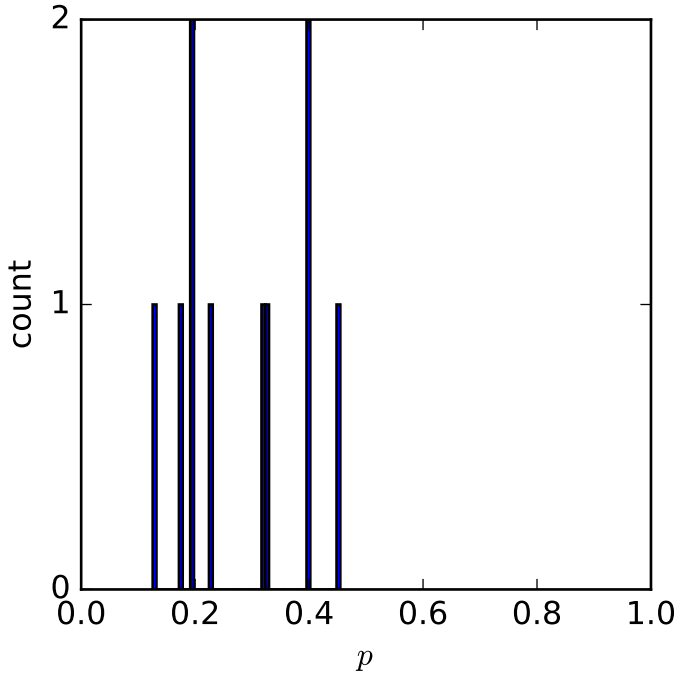
```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Given $x_1, \ldots, x_n \in \{0, 1\}$,
report probability of $x_{n+1} = 1$? E.g., $0, 0, 1, 0, 0$

```
def guesser():
  p = uniform()
  return p

def checker(p):
  return [0,0,1,0,0] == bernoulli(p,5)
```

EXAMPLE: INFERRING BIAS OF A COIN

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

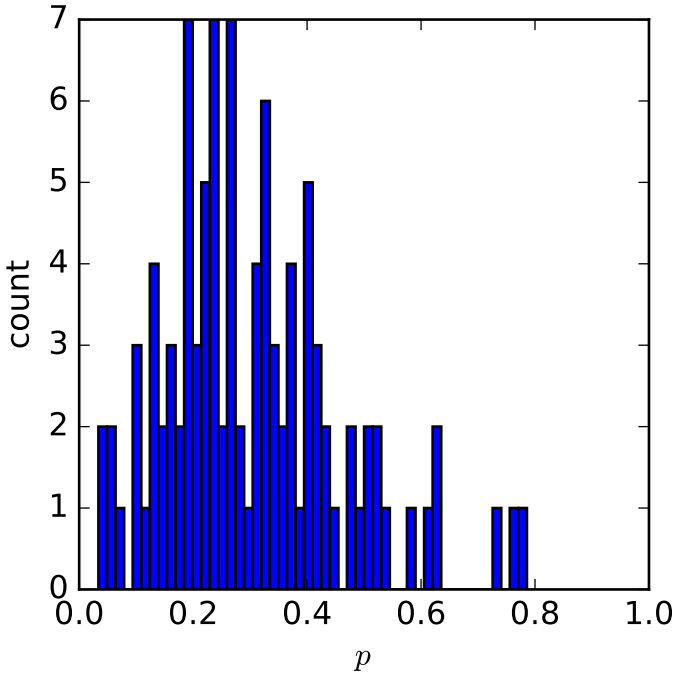


Given $x_1, \ldots, x_n \in \{0, 1\}$,
report probability of $x_{n+1} = 1$? E.g., $0, 0, 1, 0, 0$

```
def guesser():
  p = uniform()
  return p

def checker(p):
  return [0,0,1,0,0] == bernoulli(p,5)
```

EXAMPLE: INFERRING BIAS OF A COIN

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

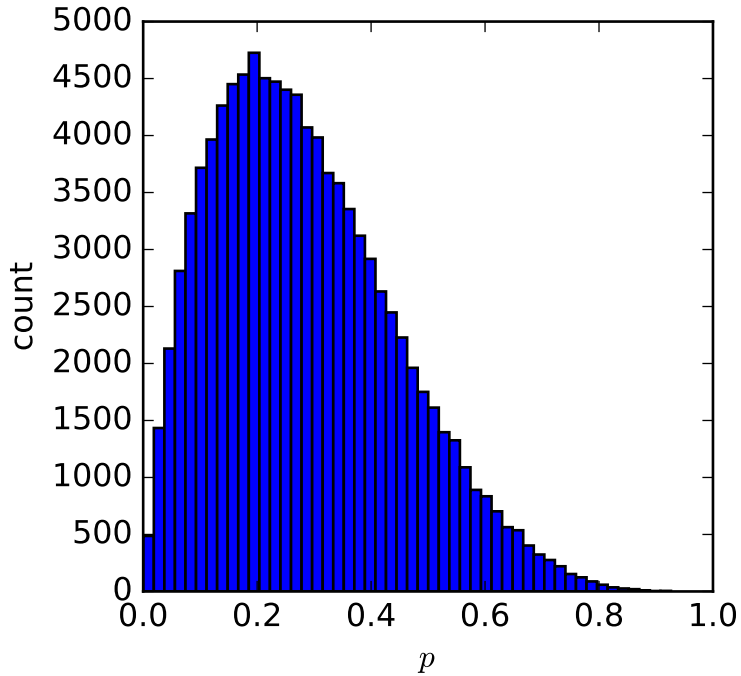


Given $x_1, \ldots, x_n \in \{0, 1\}$,
report probability of $x_{n+1} = 1$? E.g., $0, 0, 1, 0, 0$

```
def guesser():
  p = uniform()
  return p

def checker(p):
  return [0,0,1,0,0] == bernoulli(p,5)
```

EXAMPLE: INFERRING BIAS OF A COIN

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

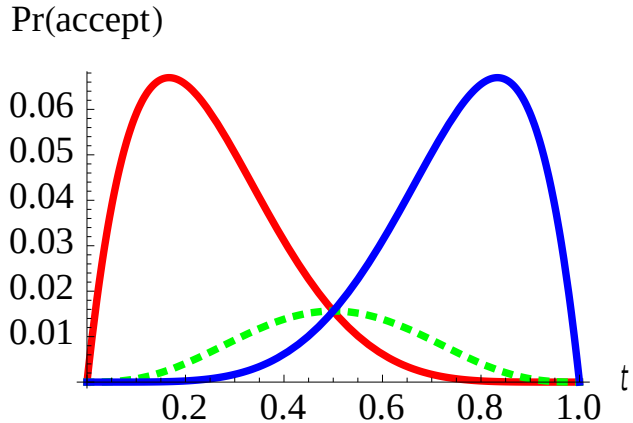


Given $x_1, \ldots, x_n \in \{0, 1\}$,
report probability of $x_{n+1} = 1$? E.g., $0, 0, 1, 0, 0$

```
def guesser():
  p = uniform()
  return p

def checker(p):
  return [0,0,1,0,0] == bernoulli(p,5)
```

Let $s = x_1 + \cdots + x_n$ and let U be uniformly distributed.
For all $t \in [0,1]$, we have $\mathbb{P}\{U \leq t\} = t$ and

$$\begin{aligned}\mathbb{P}\{\texttt{checker}(t,x) \text{ is } \texttt{True}\} &= \mathbb{P}\{\ \forall i\ (\ U_i \leq t \iff x_i = 1\ )\} \\ &= t^s(1-t)^{n-s}.\end{aligned}$$



$$n = 6,\ s \in \{1,3,5\}.$$

$$\mathbb{P}\{\texttt{checker}(U,x) \text{ is } \texttt{True}\} = \int_0^1 t^s(1-t)^{n-s}\,\mathrm{d}t = \frac{(s)!(n-s)!}{(n+1)!} =: Z(s)$$

Let $p(t)\mathrm{d}t$ be the probability that the accepted $\theta \in [t, t+\mathrm{d}t)$.

$$p(t)\mathrm{d}t \approx t^s(1-t)^{n-s}\mathrm{d}t + \{1 - Z(s)\}p(t)\mathrm{d}t \approx \frac{t^s(1-t)^{n-s}}{Z(s)}\mathrm{d}t$$

Probability that the accepted $X = 1$ is then $\int t\,p(t)\mathrm{d}t = \frac{s+1}{n+2}$.

Example: inferring objects from an image

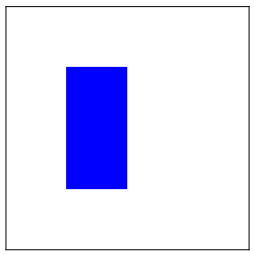
EXAMPLE: INFERRING OBJECTS FROM AN IMAGE

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

EXAMPLE: INFERRING OBJECTS FROM AN IMAGE

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

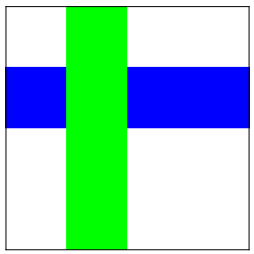
How many objects in this image?



EXAMPLE: INFERRING OBJECTS FROM AN IMAGE

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

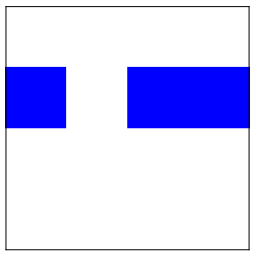
How many objects in this image?



EXAMPLE: INFERRING OBJECTS FROM AN IMAGE

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

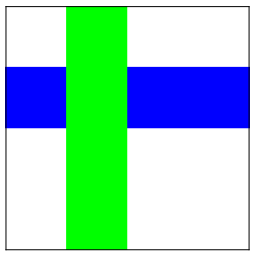
How many objects in this image?



EXAMPLE: INFERRING OBJECTS FROM AN IMAGE

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

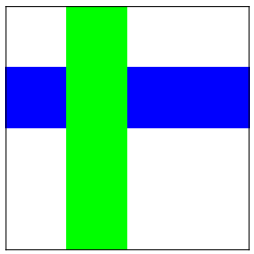
How many objects in this image?



EXAMPLE: INFERRING OBJECTS FROM AN IMAGE

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

How many objects in this image?

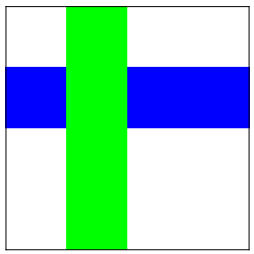


```
def guesser():
  k = geometric()
  blocks = [ randomblock() for _ in range(k) ]
  colors = [ randomcolor() for _ in range(k) ]
  return (k,blocks,colors)
```

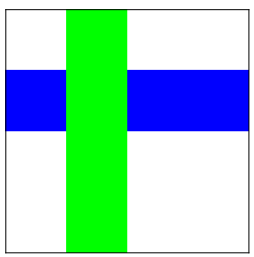
EXAMPLE: INFERRING OBJECTS FROM AN IMAGE

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

How many objects in this image?



```
def guesser():
  k = geometric()
  blocks = [ randomblock() for _ in range(k) ]
  colors = [ randomcolor() for _ in range(k) ]
  return (k,blocks,colors)

def checker(k,blocks,colors):
  return rasterize(blocks,colors) ==
```

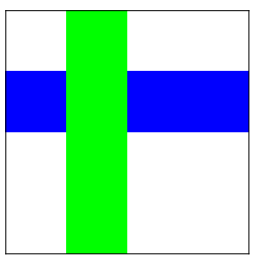


EXAMPLE: INFERRING OBJECTS FROM AN IMAGE

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

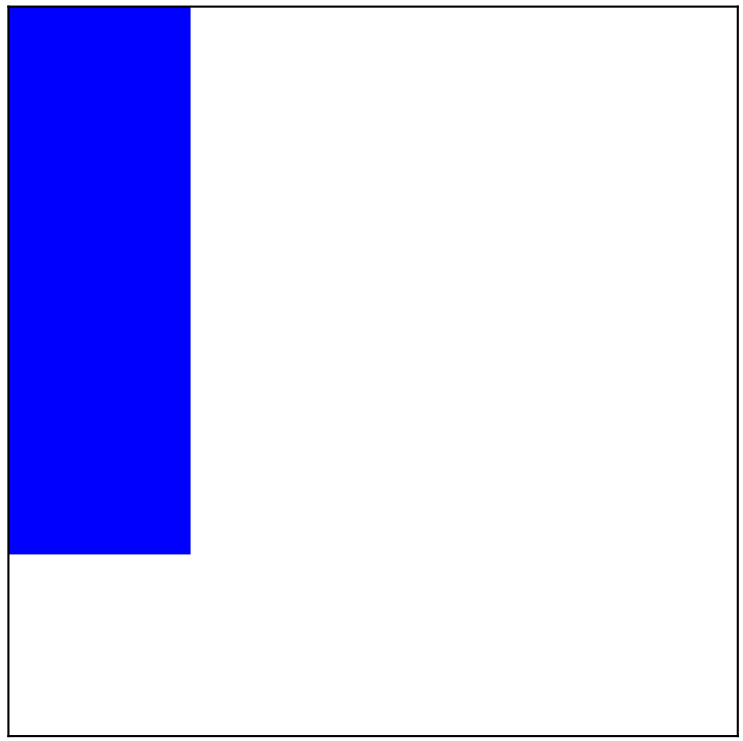
How many objects in this image?

```
def guesser():
  k = geometric()
  blocks = [ randomblock() for _ in range(k) ]
  colors = [ randomcolor() for _ in range(k) ]
  return (k,blocks,colors)

def checker(k,blocks,colors):
  return rasterize(blocks,colors) ==
```



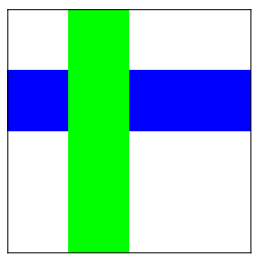
EXAMPLE: INFERRING OBJECTS FROM AN IMAGE

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```



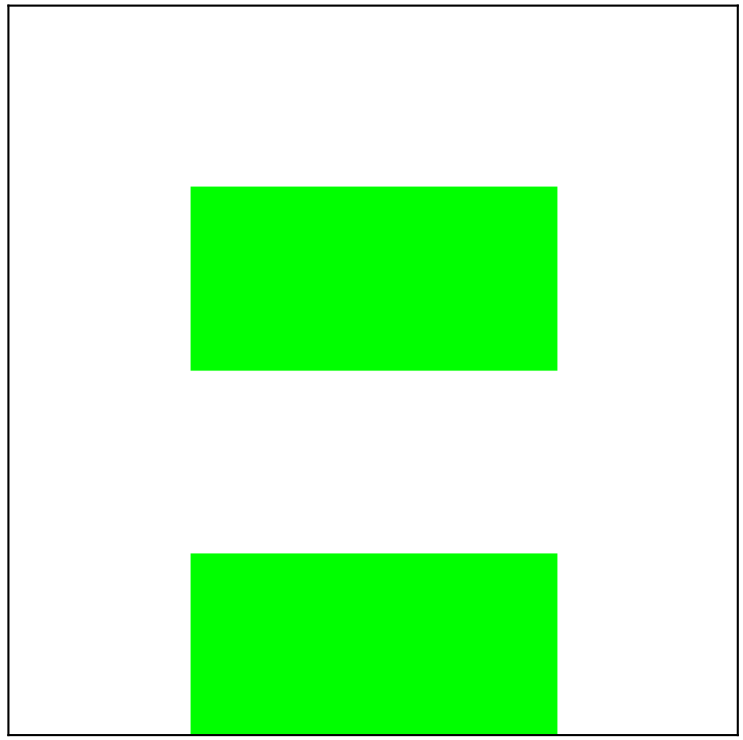
How many objects in this image?

```
def guesser():
  k = geometric()
  blocks = [ randomblock() for _ in range(k) ]
  colors = [ randomcolor() for _ in range(k) ]
  return (k,blocks,colors)

def checker(k,blocks,colors):
  return rasterize(blocks,colors) ==
```



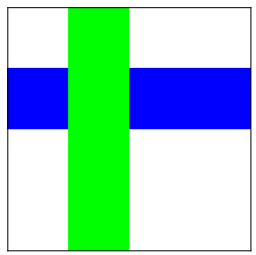
EXAMPLE: INFERRING OBJECTS FROM AN IMAGE

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```



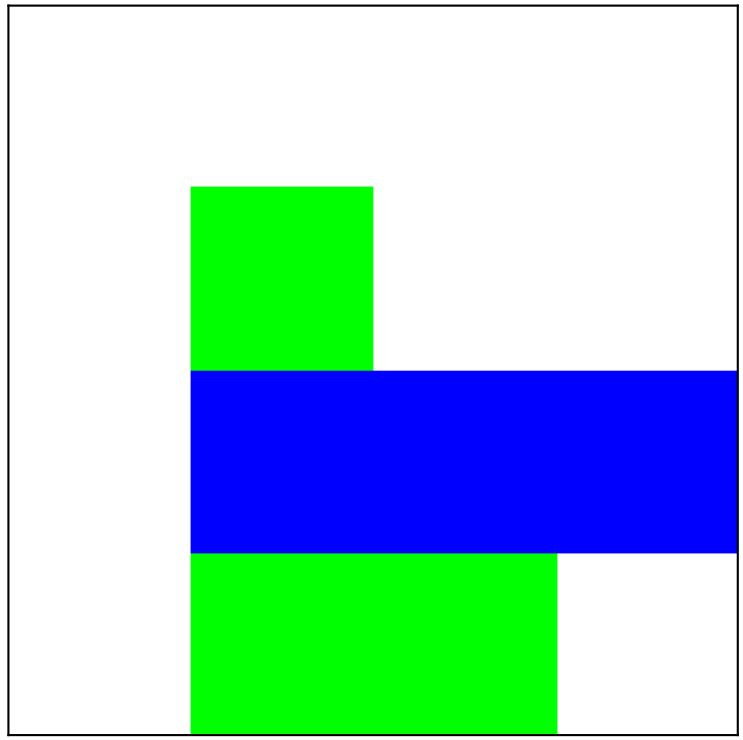
How many objects in this image?

```
def guesser():
  k = geometric()
  blocks = [ randomblock() for _ in range(k) ]
  colors = [ randomcolor() for _ in range(k) ]
  return (k,blocks,colors)

def checker(k,blocks,colors):
  return rasterize(blocks,colors) ==
```



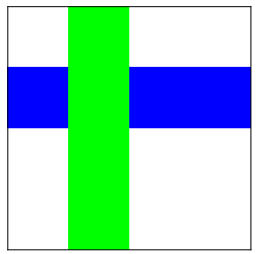
EXAMPLE: INFERRING OBJECTS FROM AN IMAGE

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```



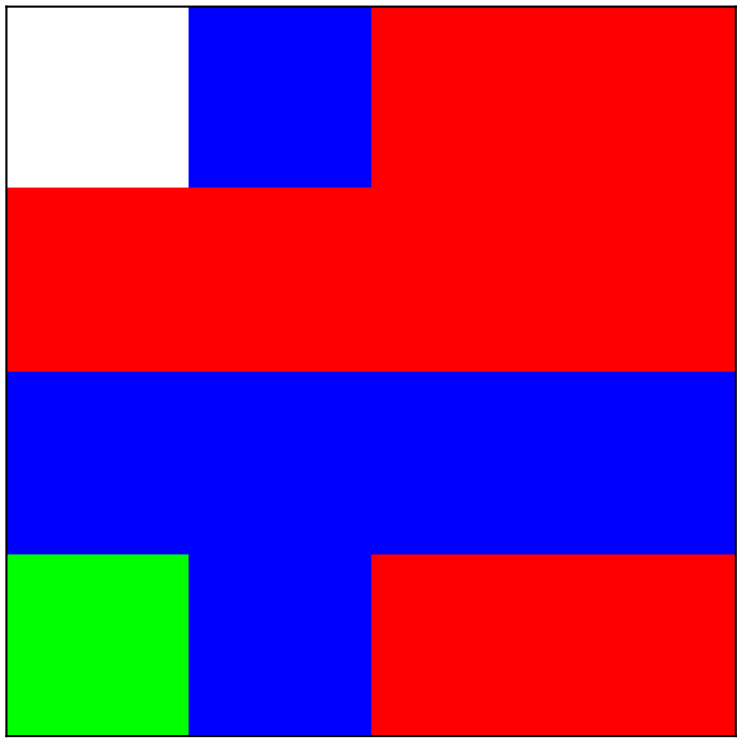
How many objects in this image?

```
def guesser():
  k = geometric()
  blocks = [ randomblock() for _ in range(k) ]
  colors = [ randomcolor() for _ in range(k) ]
  return (k,blocks,colors)

def checker(k,blocks,colors):
  return rasterize(blocks,colors) ==
```



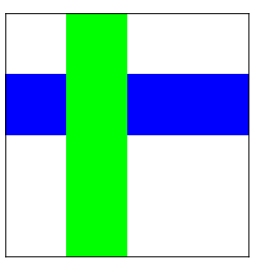
EXAMPLE: INFERRING OBJECTS FROM AN IMAGE

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```



How many objects in this image?

```
def guesser():
  k = geometric()
  blocks = [ randomblock() for _ in range(k) ]
  colors = [ randomcolor() for _ in range(k) ]
  return (k,blocks,colors)

def checker(k,blocks,colors):
  return rasterize(blocks,colors) ==
```

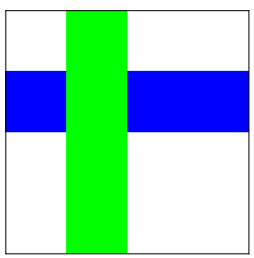


EXAMPLE: INFERRING OBJECTS FROM AN IMAGE

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

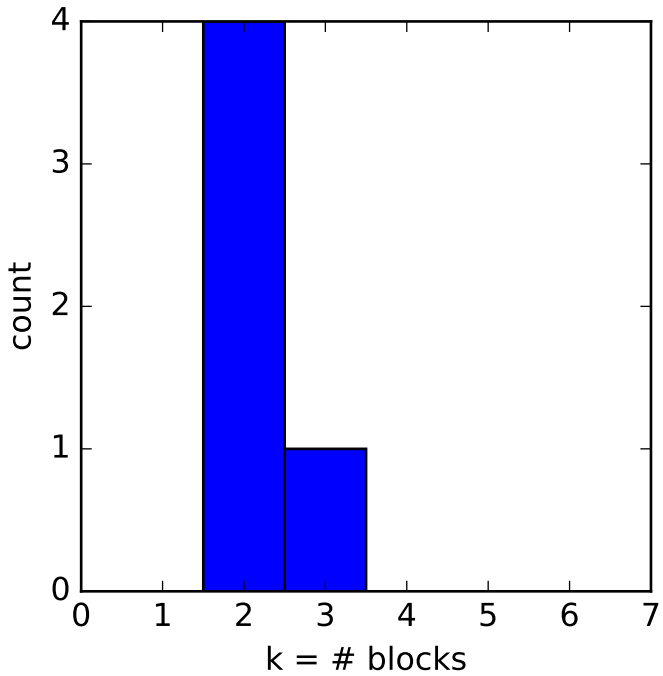
How many objects in this image?

```
def guesser():
  k = geometric()
  blocks = [ randomblock() for _ in range(k) ]
  colors = [ randomcolor() for _ in range(k) ]
  return (k,blocks,colors)

def checker(k,blocks,colors):
  return rasterize(blocks,colors) ==
```



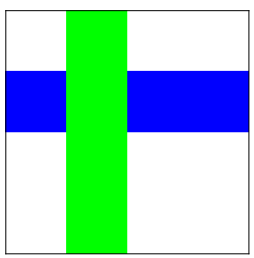
EXAMPLE: INFERRING OBJECTS FROM AN IMAGE

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```



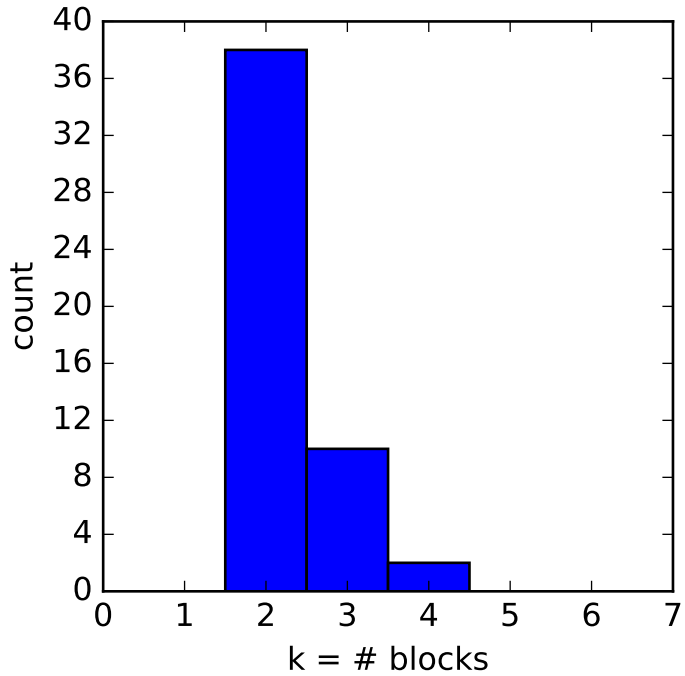
How many objects in this image?

```
def guesser():
  k = geometric()
  blocks = [ randomblock() for _ in range(k) ]
  colors = [ randomcolor() for _ in range(k) ]
  return (k,blocks,colors)

def checker(k,blocks,colors):
  return rasterize(blocks,colors) ==
```



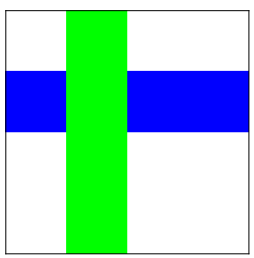
EXAMPLE: INFERRING OBJECTS FROM AN IMAGE

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```



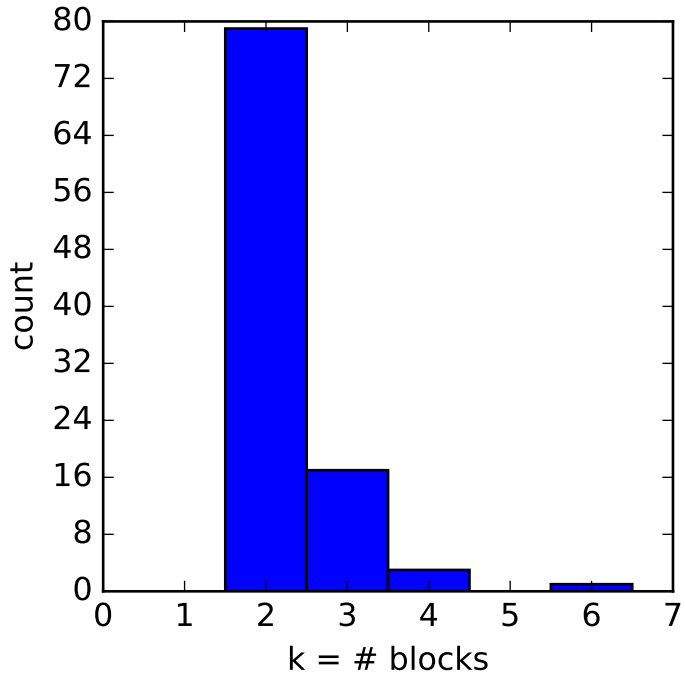
How many objects in this image?

```
def guesser():
  k = geometric()
  blocks = [ randomblock() for _ in range(k) ]
  colors = [ randomcolor() for _ in range(k) ]
  return (k,blocks,colors)

def checker(k,blocks,colors):
  return rasterize(blocks,colors) ==
```



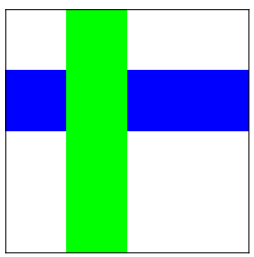
EXAMPLE: INFERRING OBJECTS FROM AN IMAGE

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```



How many objects in this image?

```
def guesser():
  k = geometric()
  blocks = [ randomblock() for _ in range(k) ]
  colors = [ randomcolor() for _ in range(k) ]
  return (k,blocks,colors)

def checker(k,blocks,colors):
  return rasterize(blocks,colors) ==
```



Example: Extracting 3D structure from images

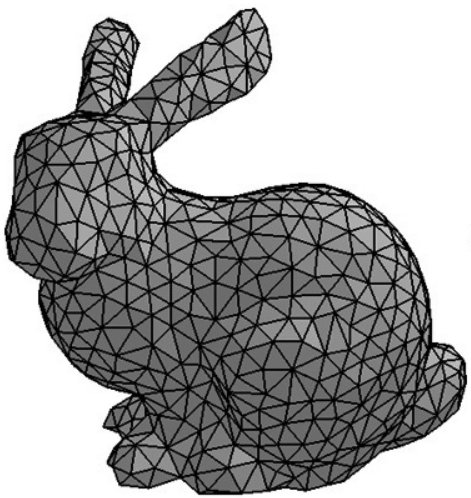
```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Example: Extracting 3D structure from images

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

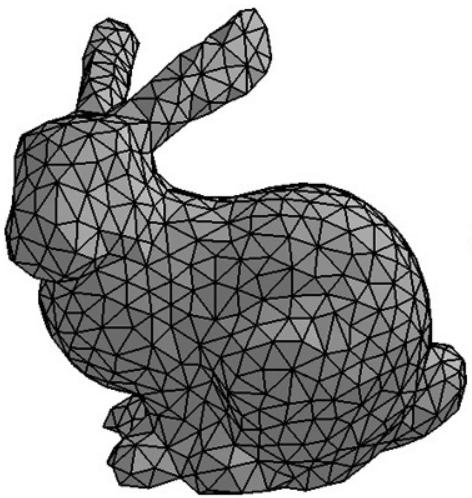
Example: Extracting 3D structure from images

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```



Example: Extracting 3D structure from images

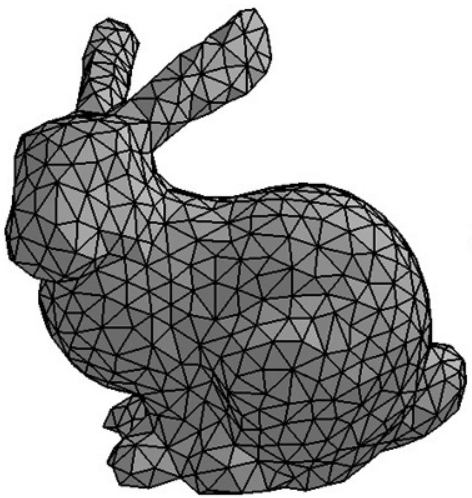
```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```



checker

Example: Extracting 3D structure from images

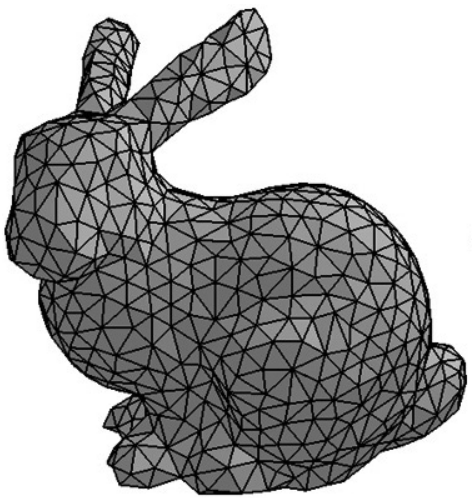
```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```



checker

Example: Extracting 3D structure from images

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```



checker

Example: Extracting 3D structure from images

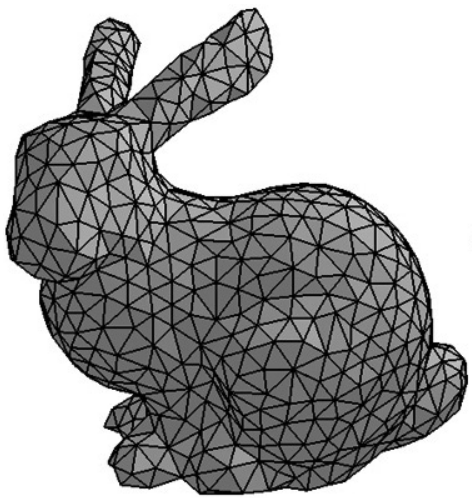
```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Example: Extracting 3D structure from images

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```

Example: Extracting 3D structure from images

```
accept = False
while (not accept):
    guess = guesser()
    accept = checker(guess)
return guess
```



inference

QUERY AS AN ALGORITHM

Key point: QUERY is not a serious proposal for an *algorithm*, but it denotes the operation we care about in Bayesian analysis.

QUERY AS AN ALGORITHM

Key point: `QUERY` is not a serious proposal for an *algorithm*, but it denotes the operation we care about in Bayesian analysis.

How efficient is `QUERY`? Let `model()` represent a distribution P and `pred` represent an indicator function $\mathbf{1}_A$.

Proposition

In expectation, `QUERY(model,pred)` *takes* $\frac{1}{P(A)}$ *times as long to run as* `pred(model())`.

Corollary

If `pred(model())` *is efficient and* $P(A)$ *not too small, then* `QUERY(model,pred)` *is efficient.*

MIT-Church aka Trace-MCMC

```
1 def geometric(p):
2     if bernoulli(p) == 1: return 1
3     else: return 1 + geometric(p)
```

MIT-Church aka Trace-MCMC

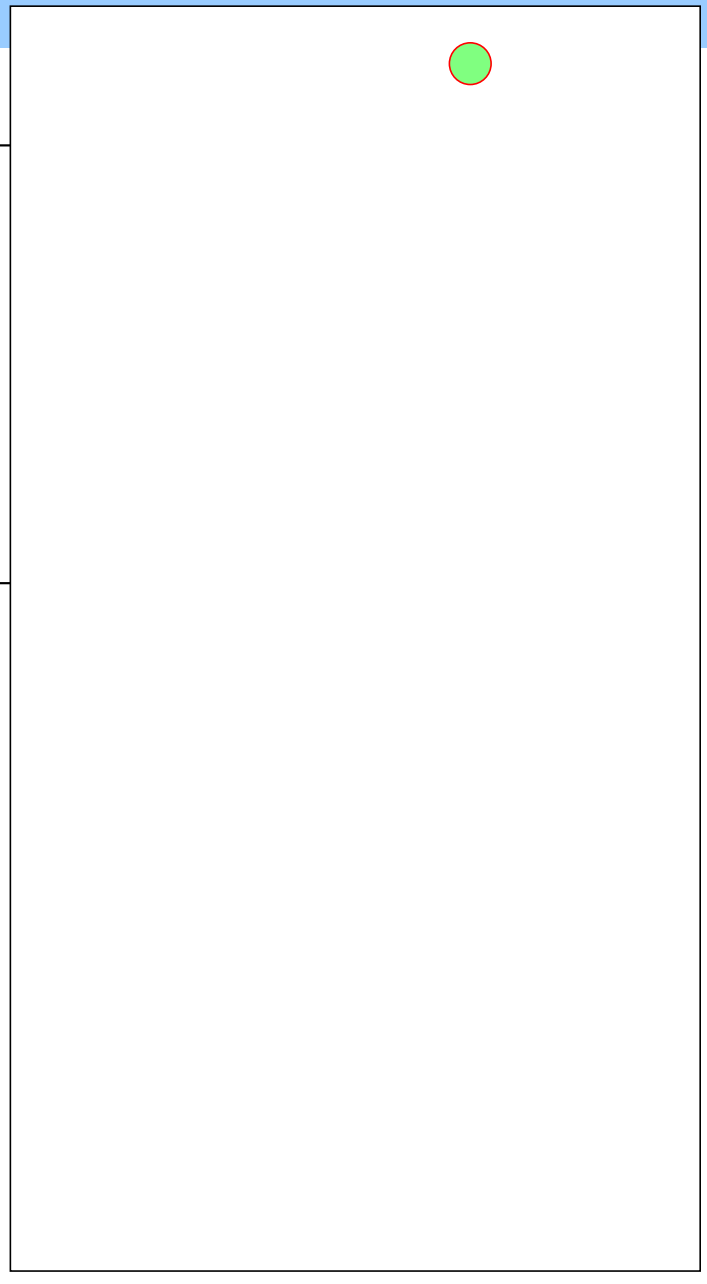
```
def geometric(p):
    if bernoulli(p) == 1: return 1
    else: return 1 + geometric(p)
```

```
def aliased_geometric(p):
    g = geometric(p)
    return 1 if g < 3 else 0
```

MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

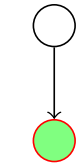
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

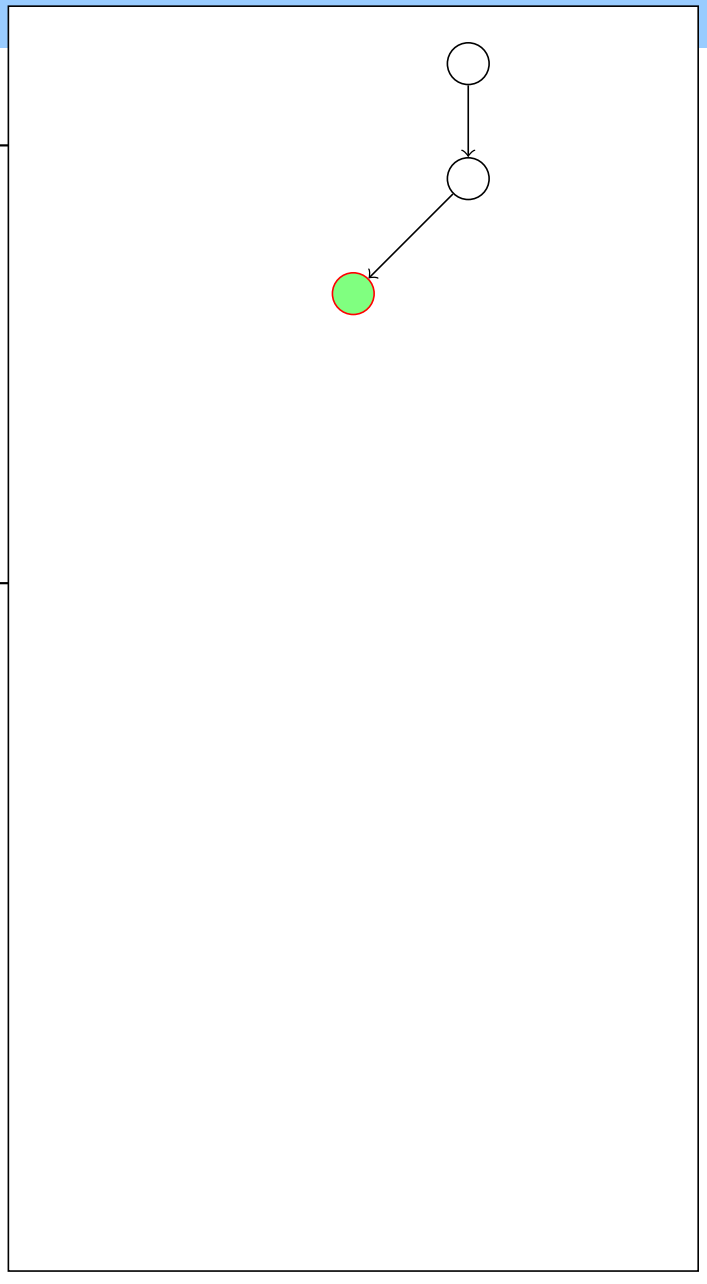
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

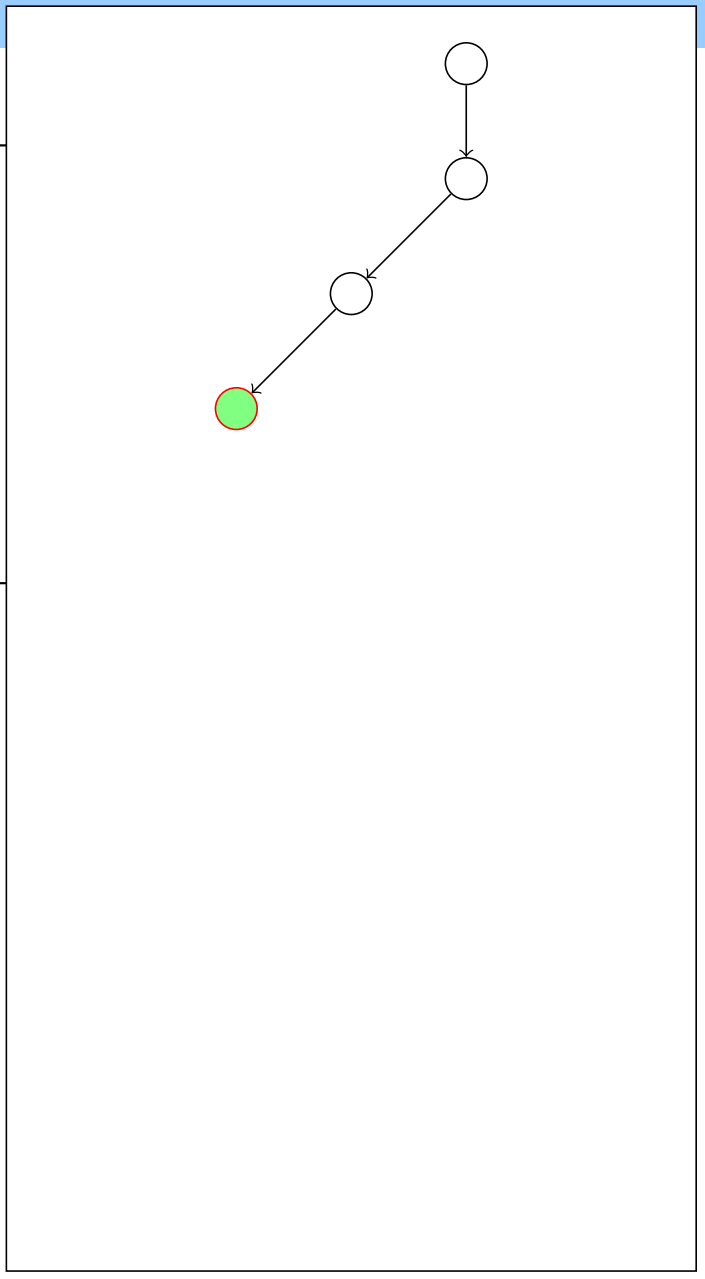
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

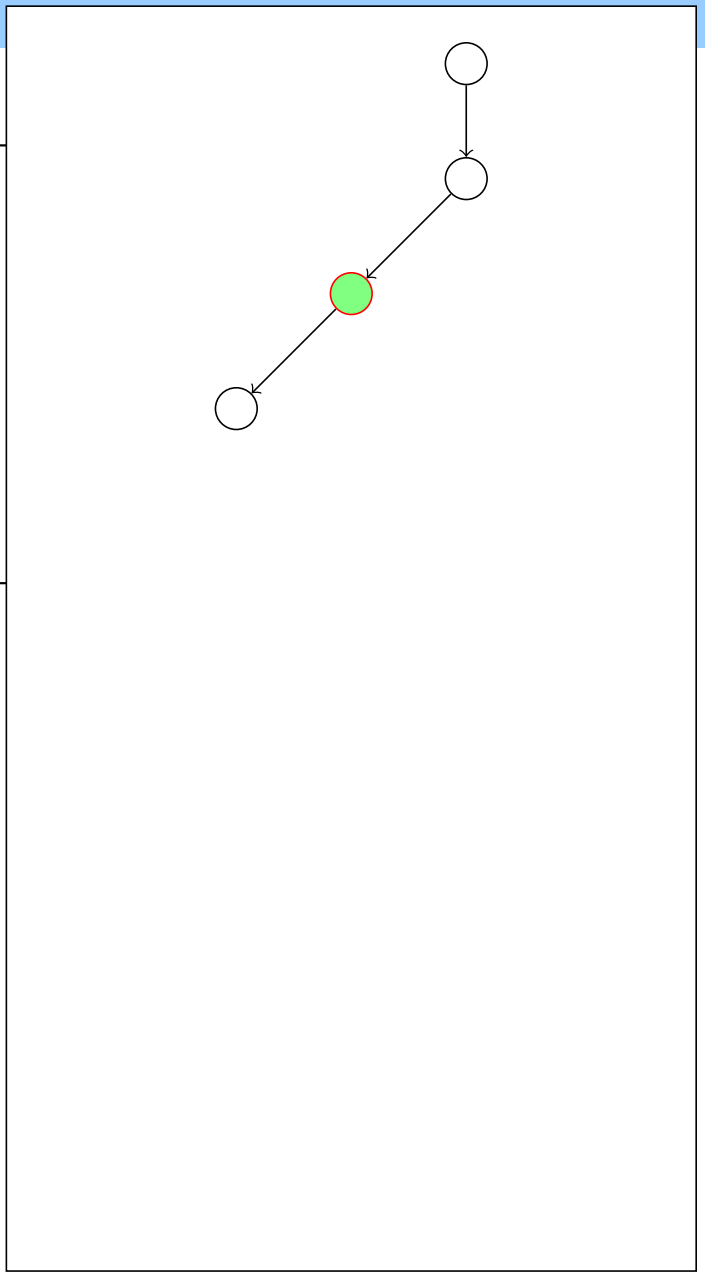
```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

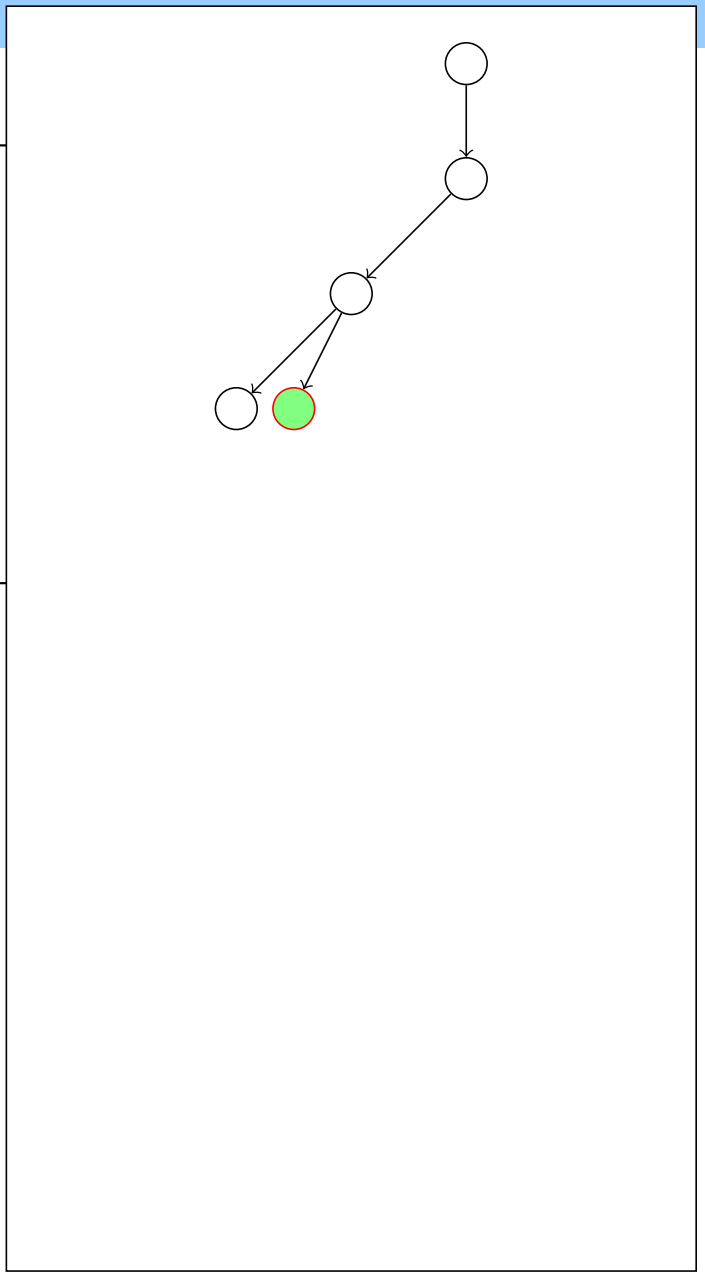
```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)

1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

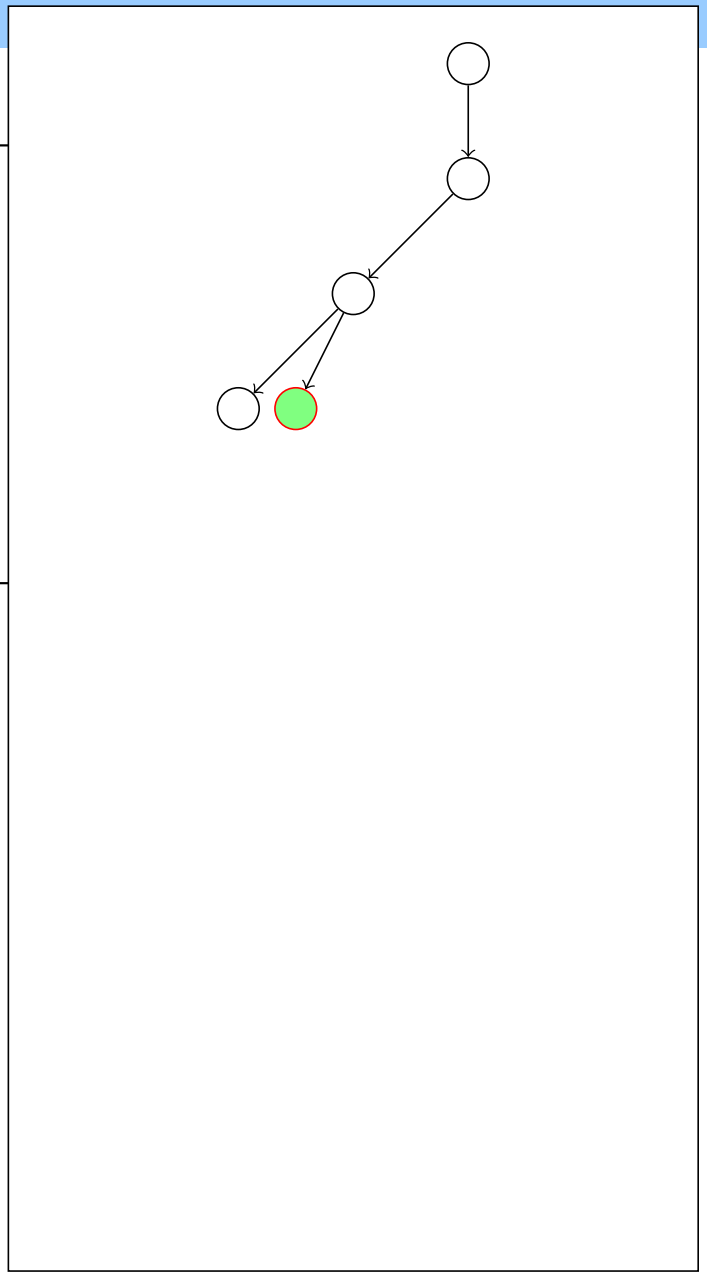
```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



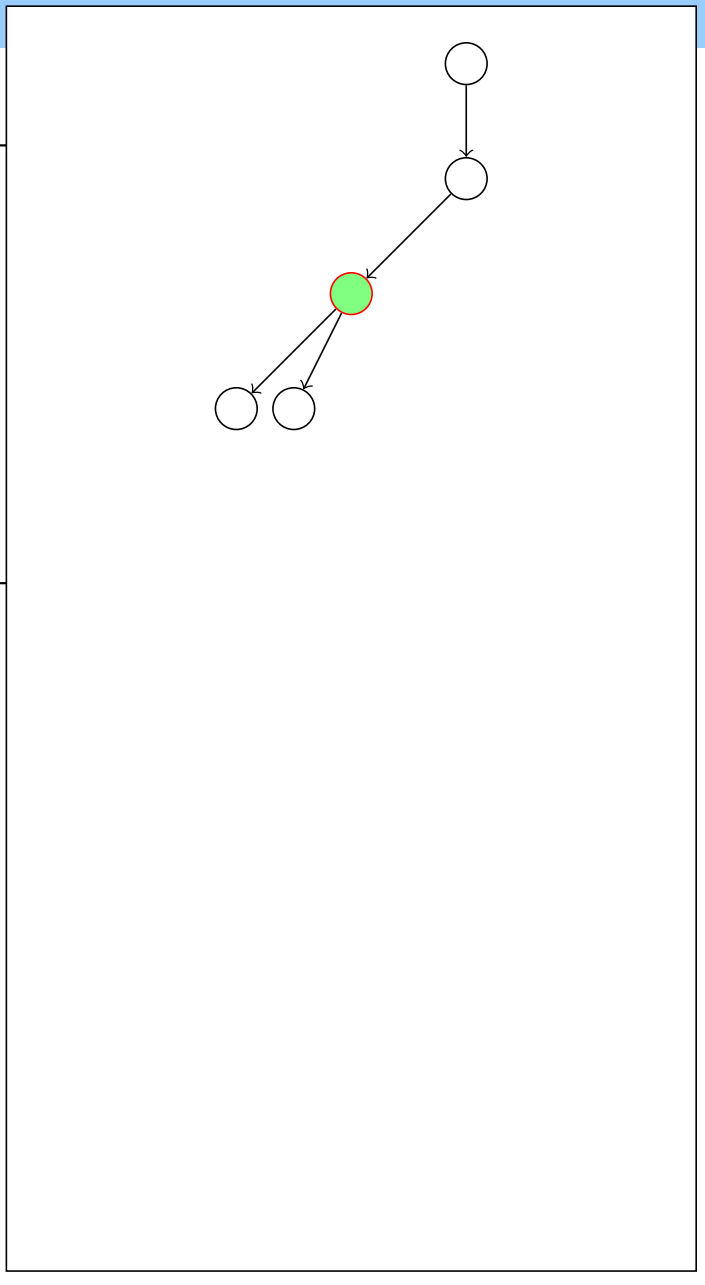
MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)

1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

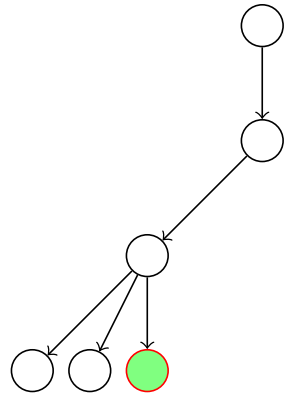
```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)

1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

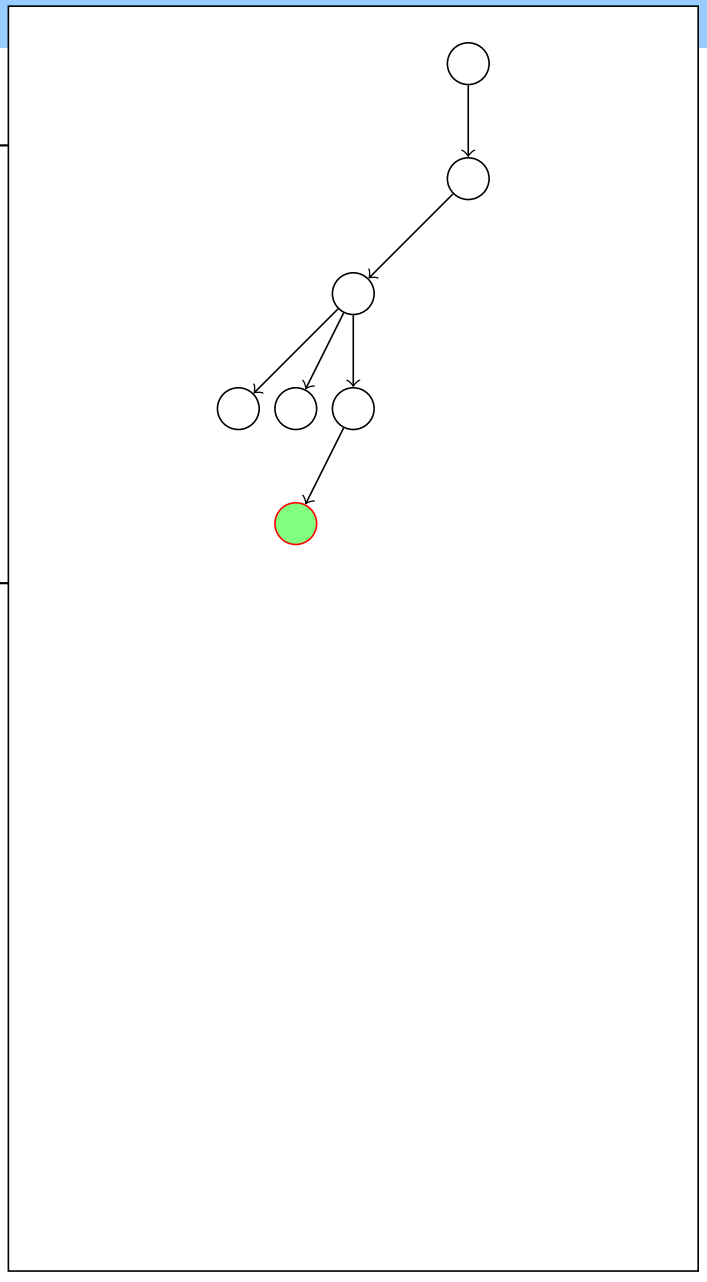
```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

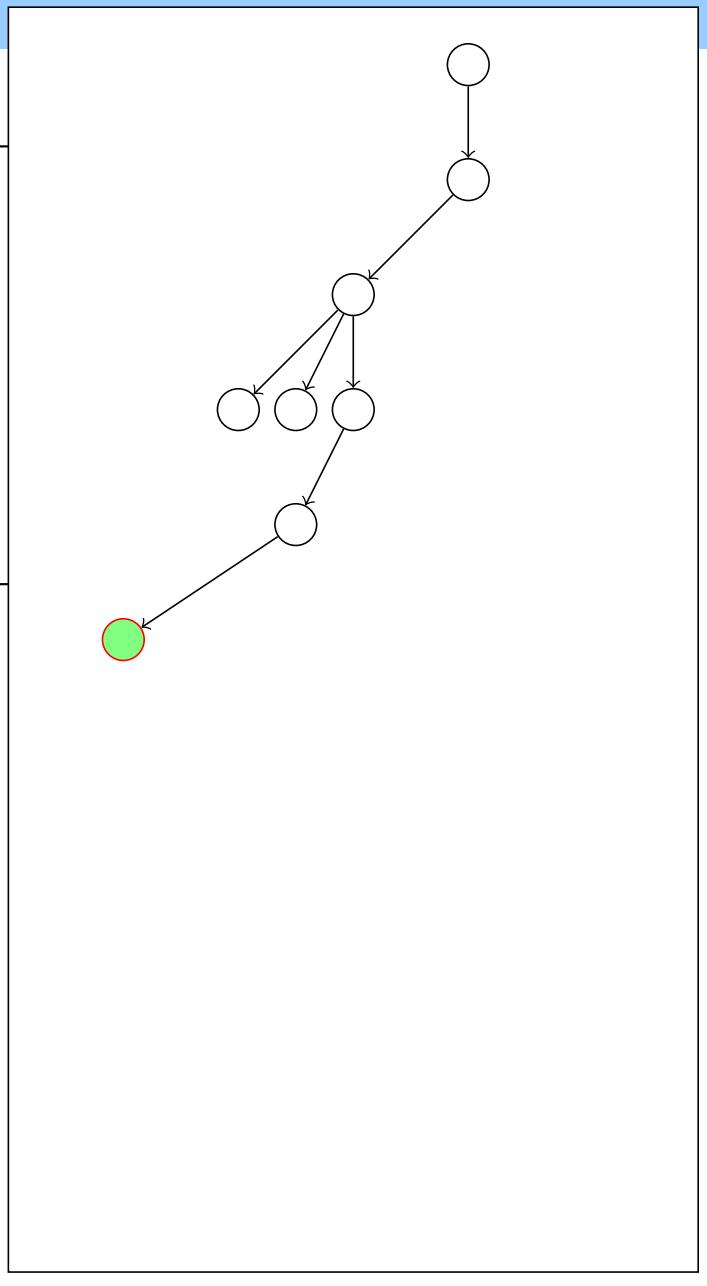
```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)

1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

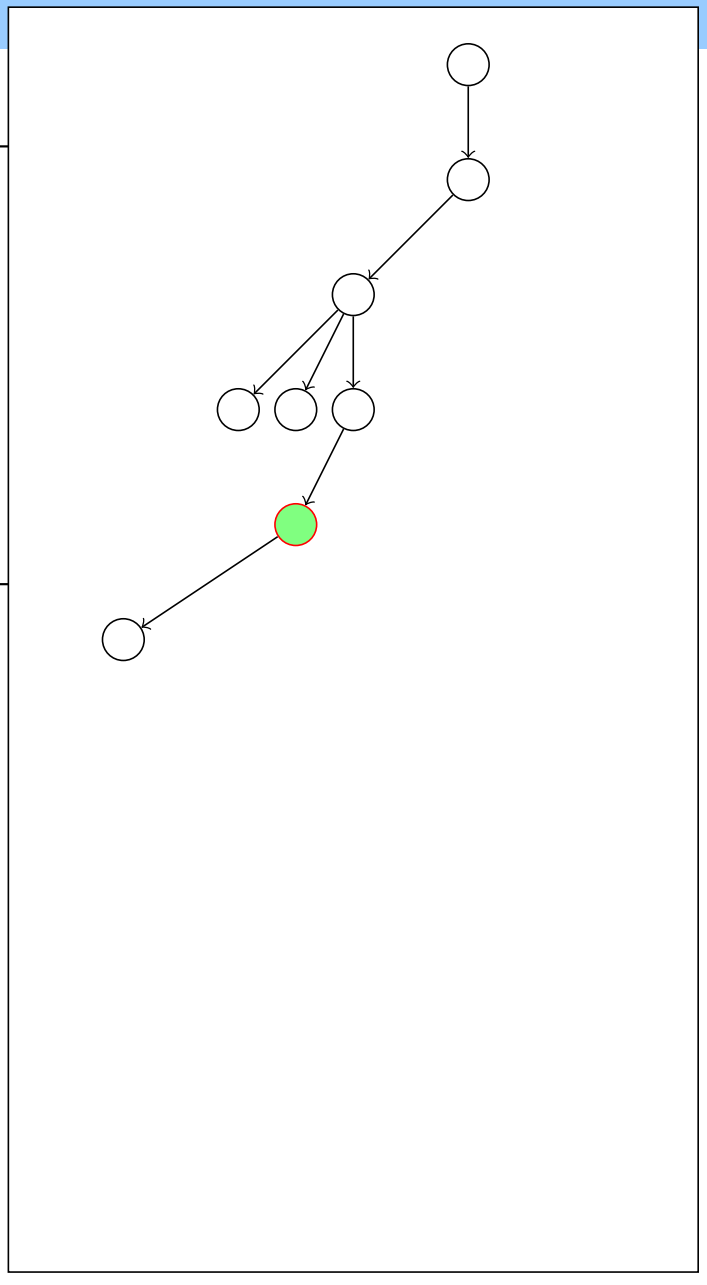
```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

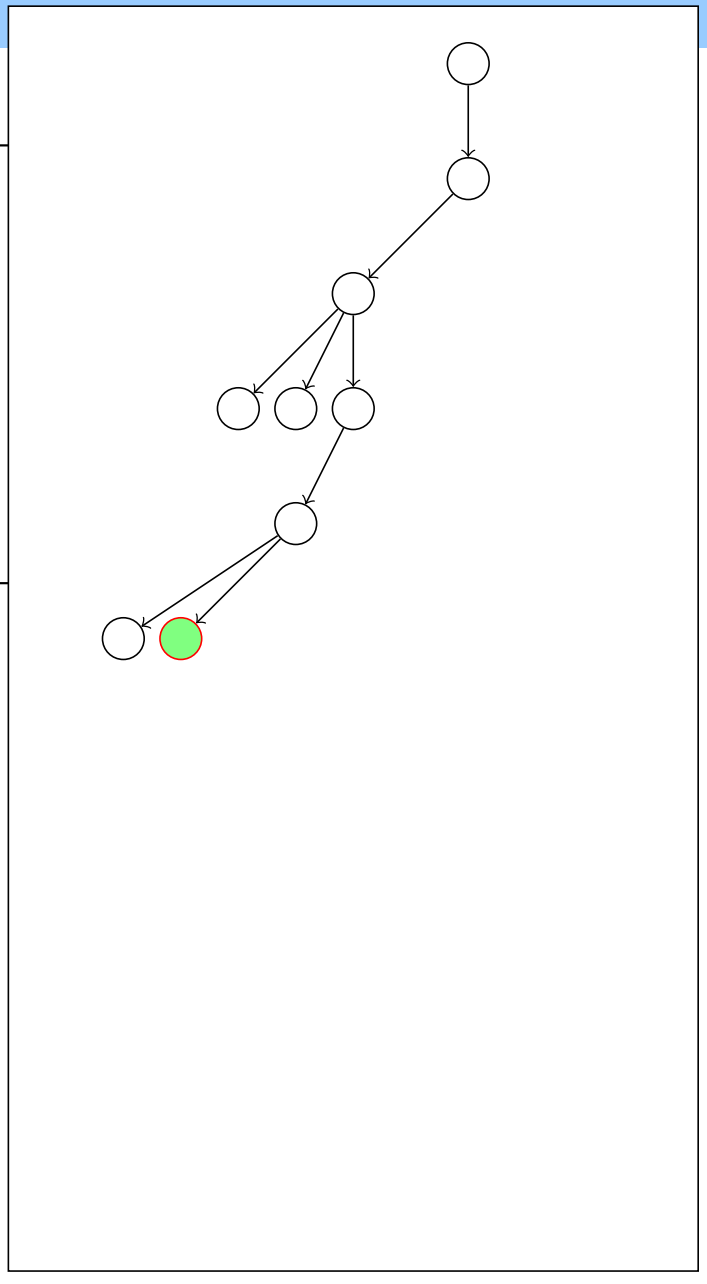
```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)

1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

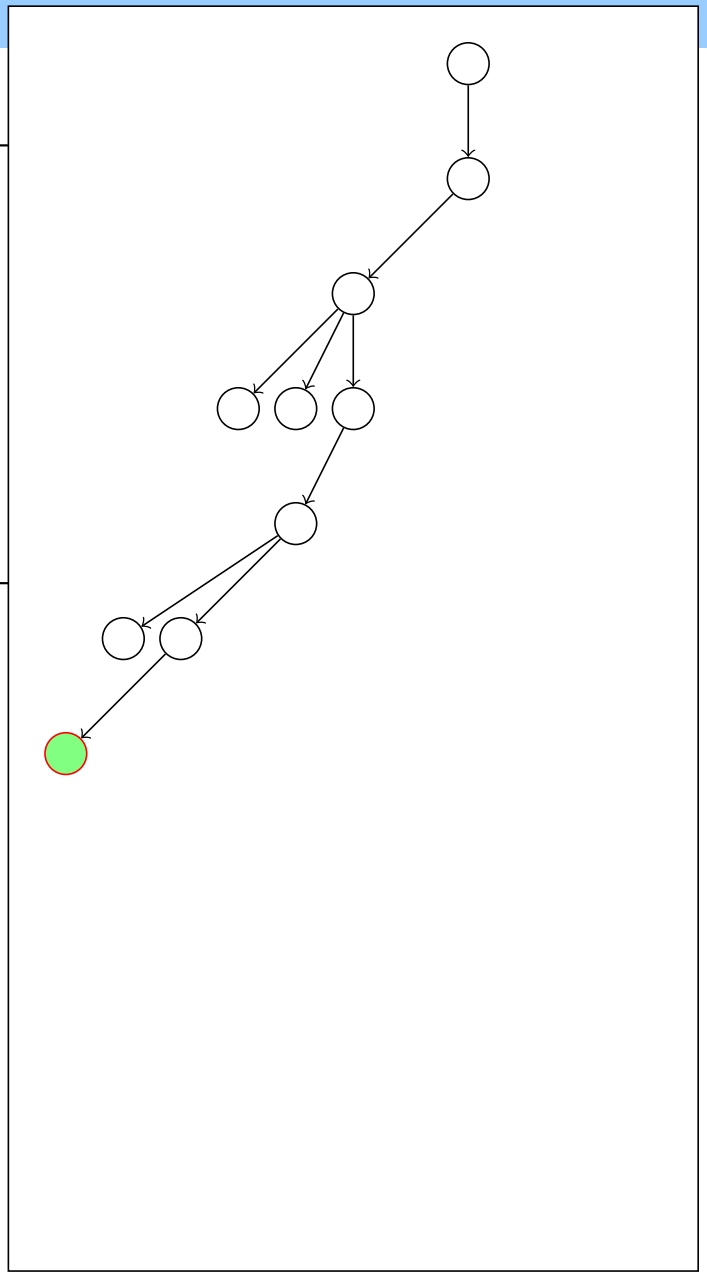
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

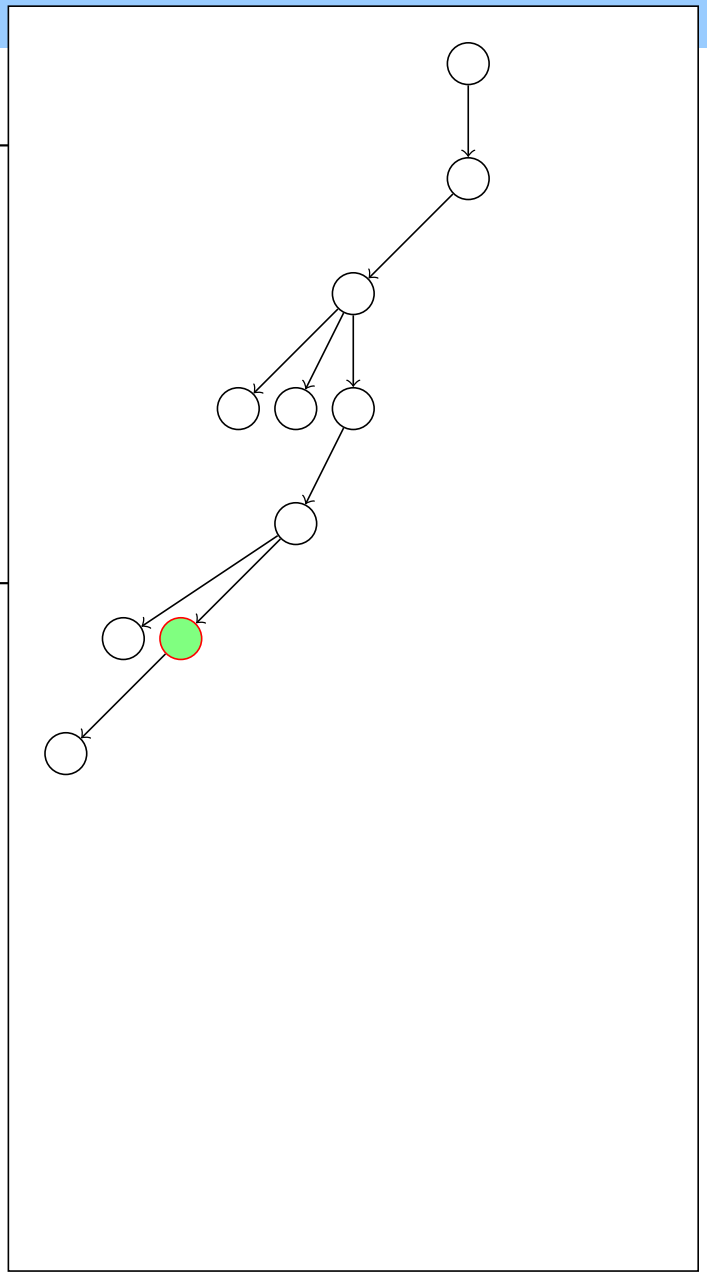
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

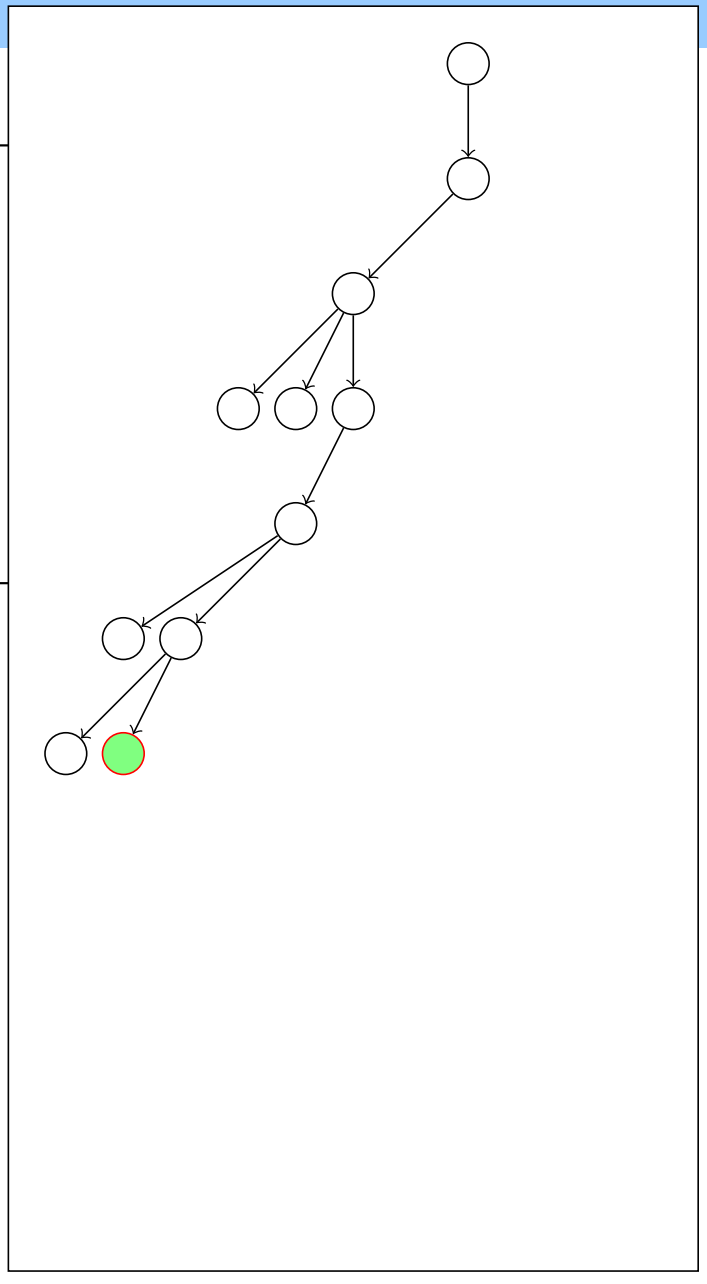
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

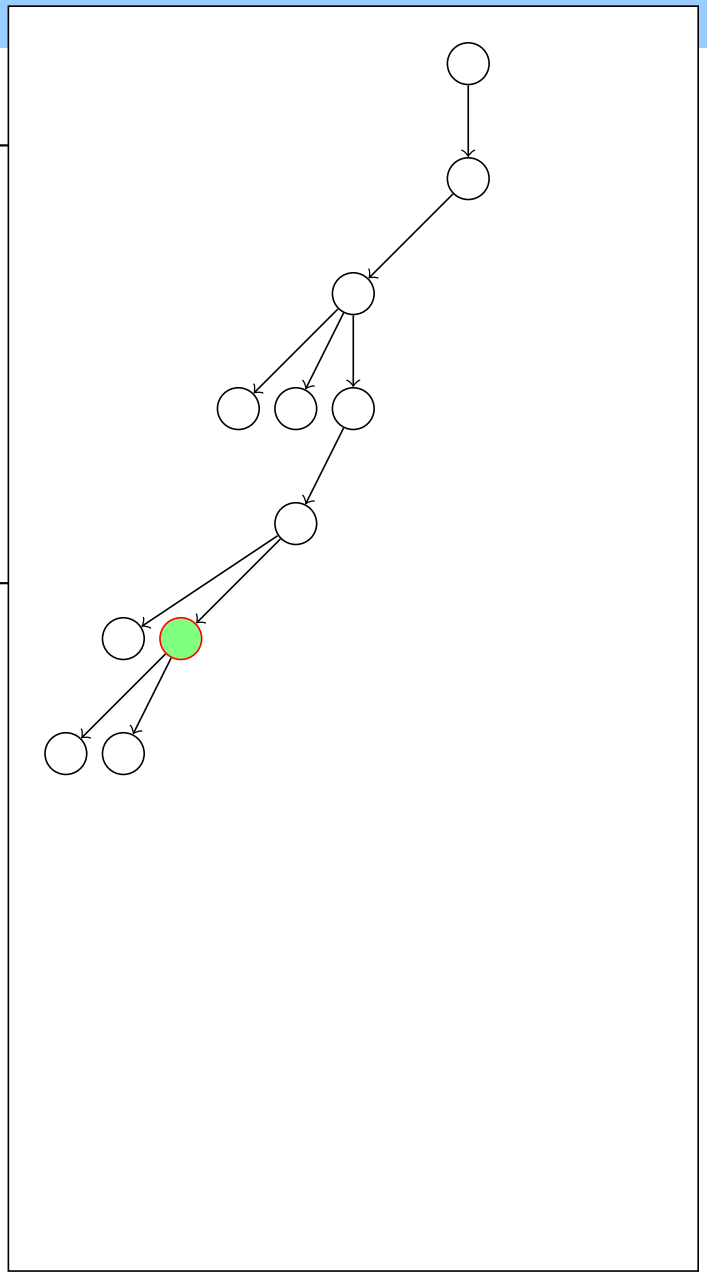
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

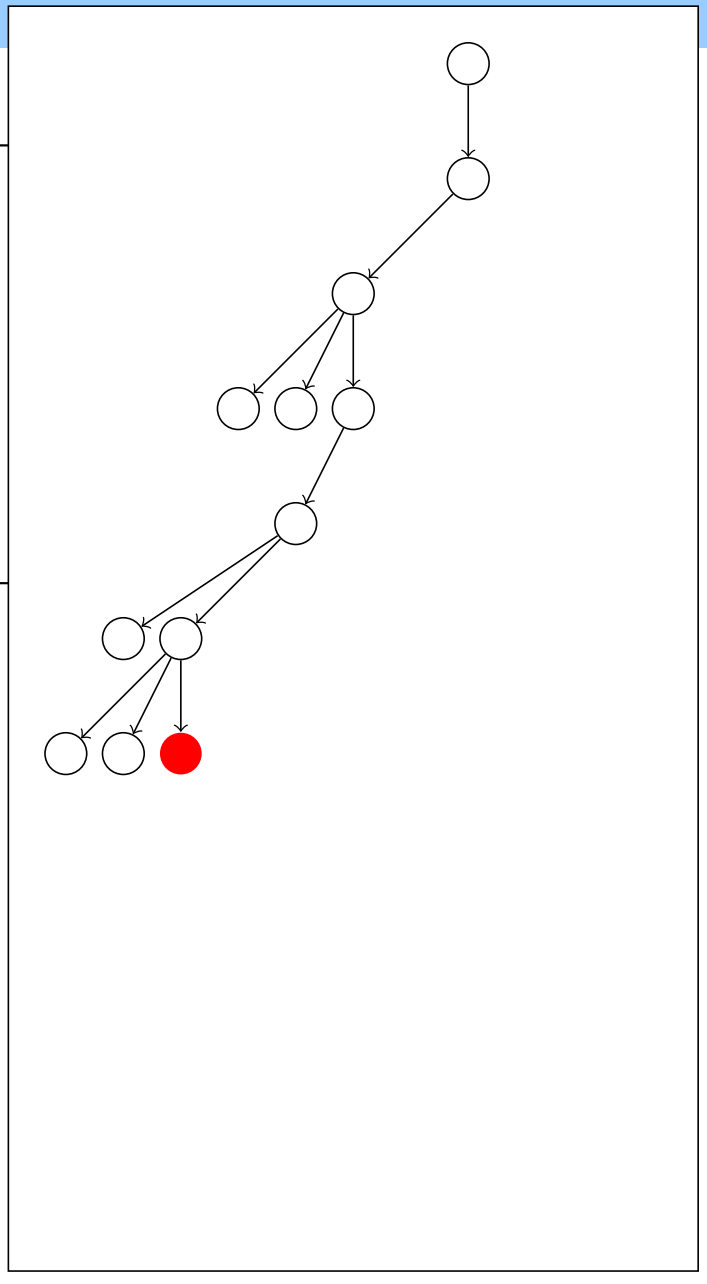
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

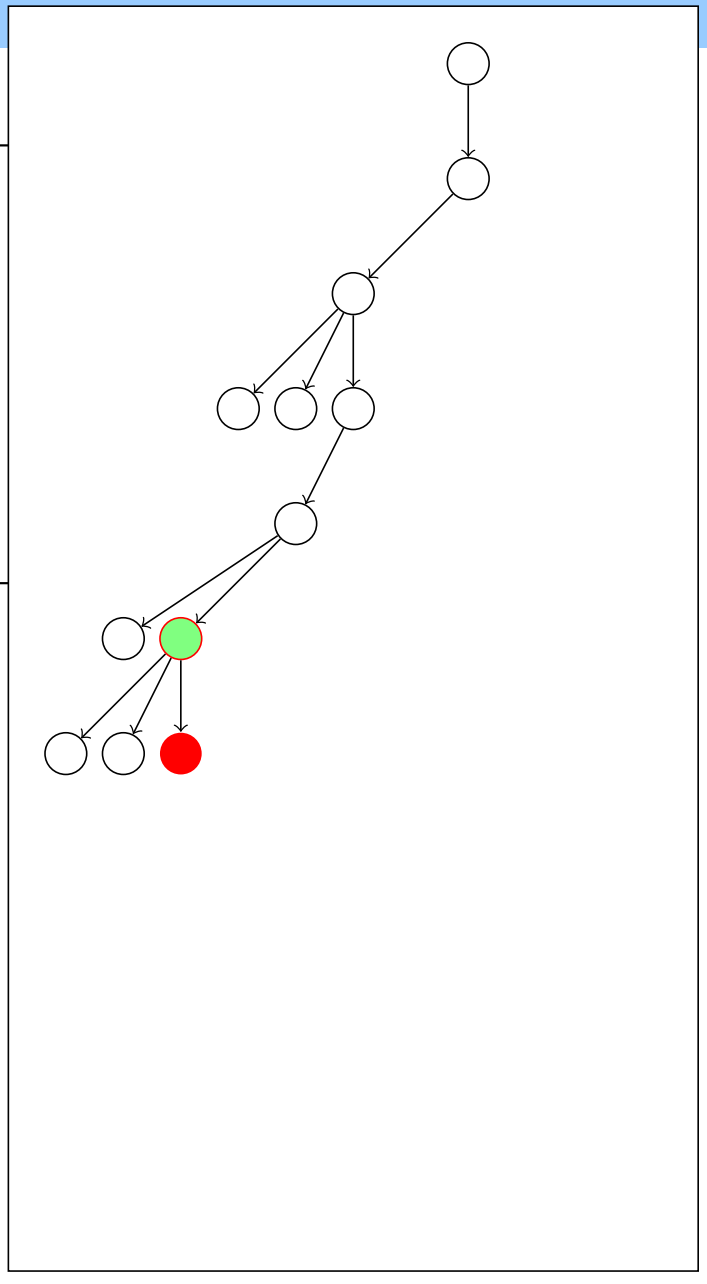
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

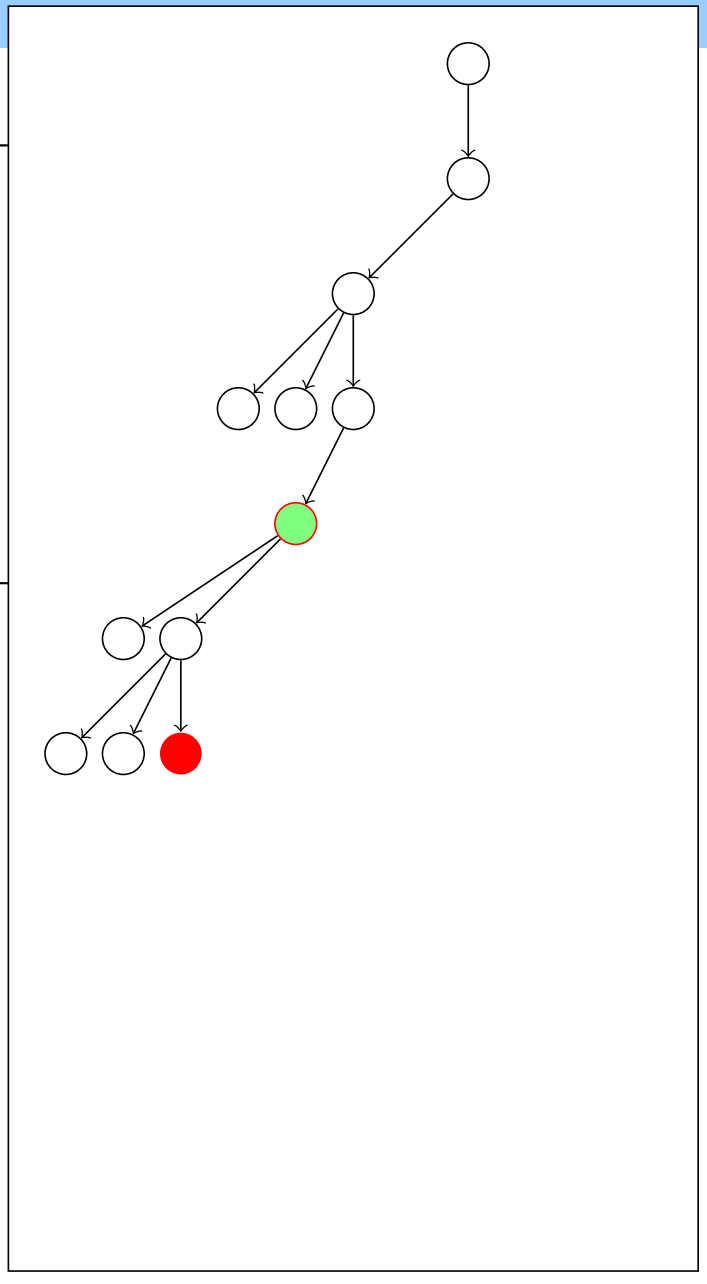
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

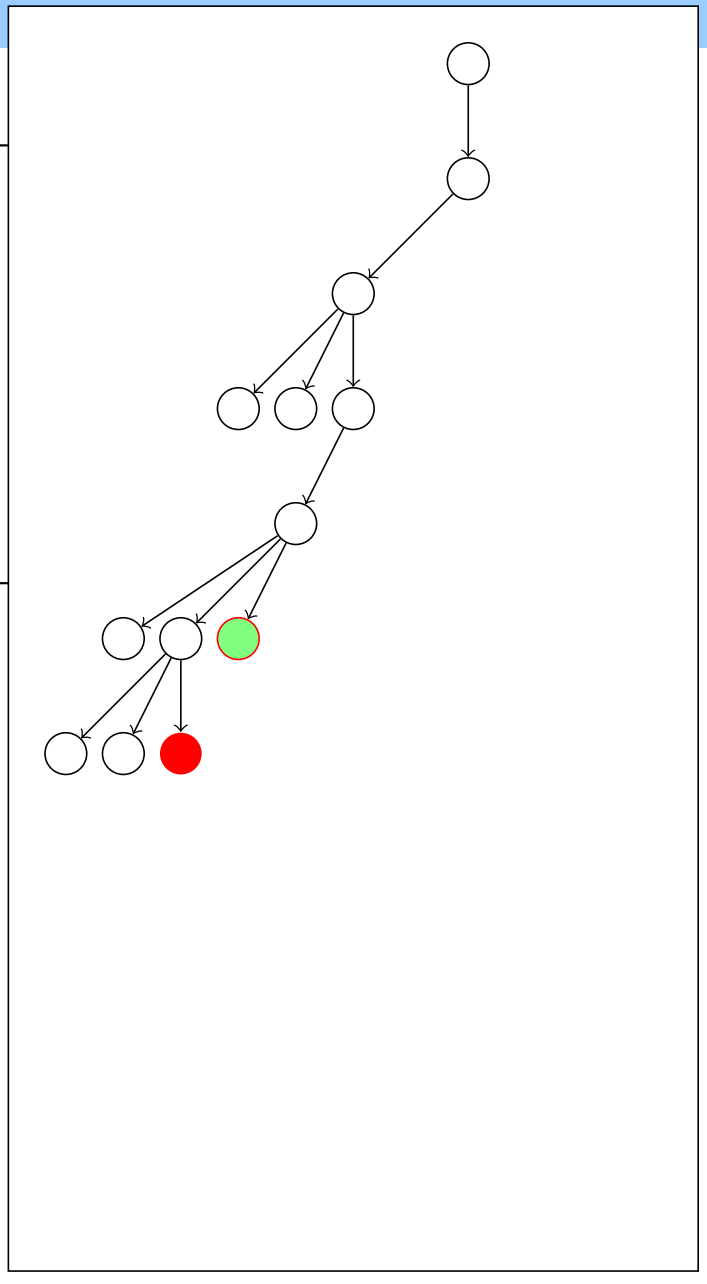
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

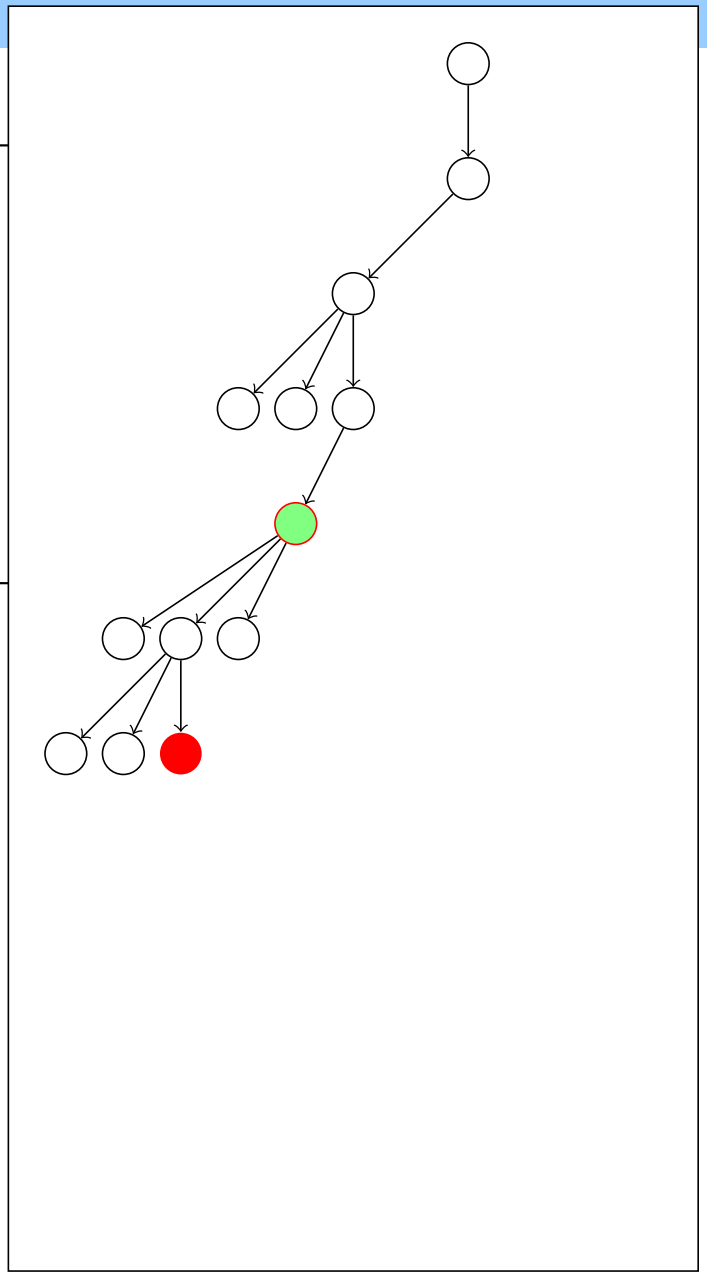
```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

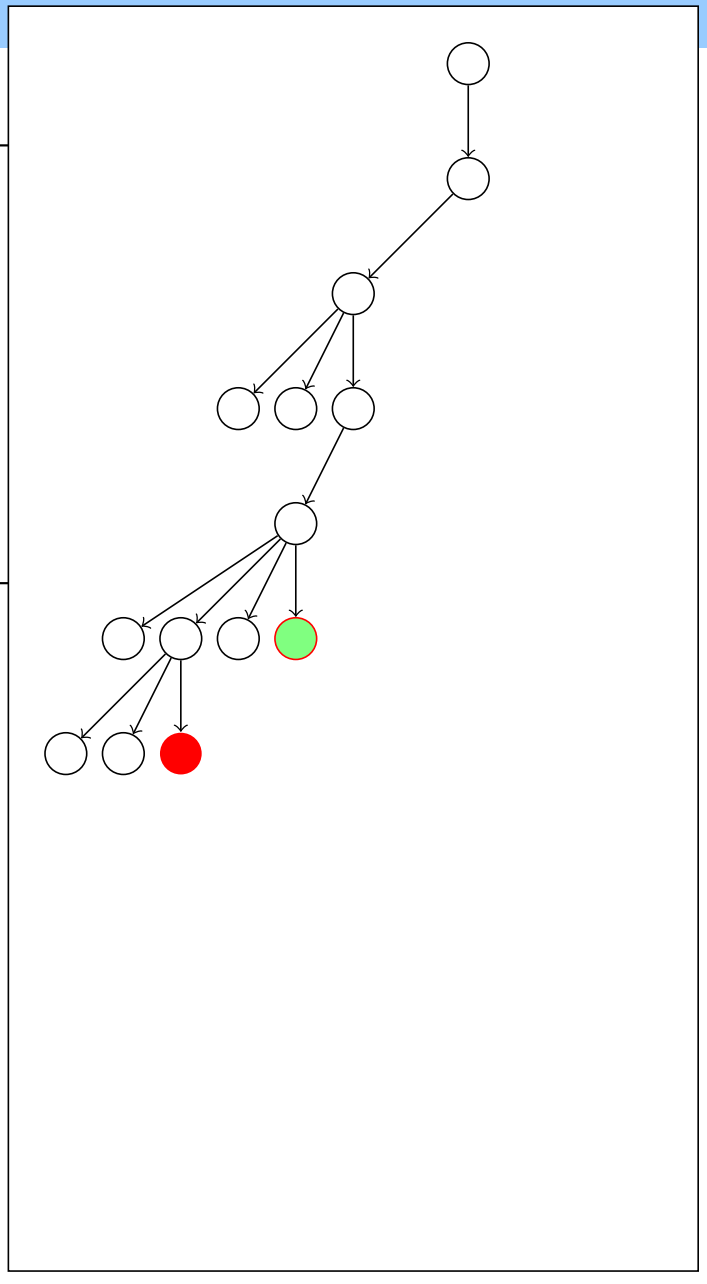
```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)

1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

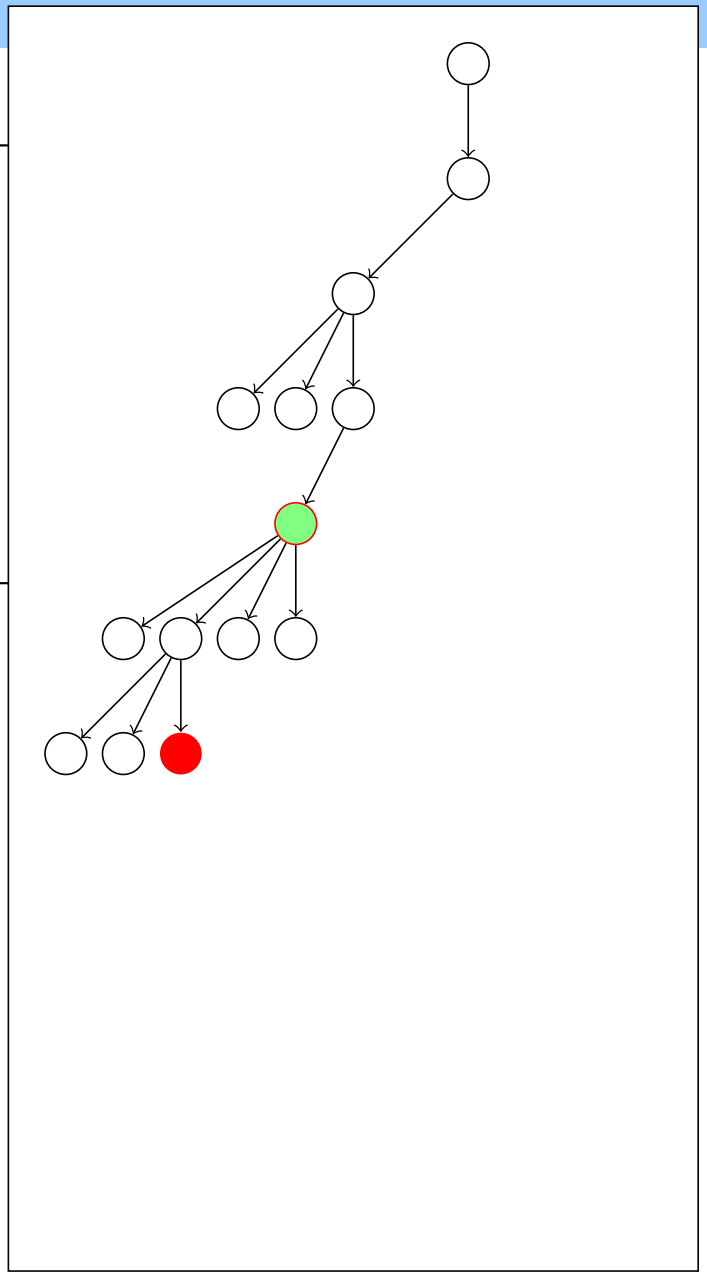
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

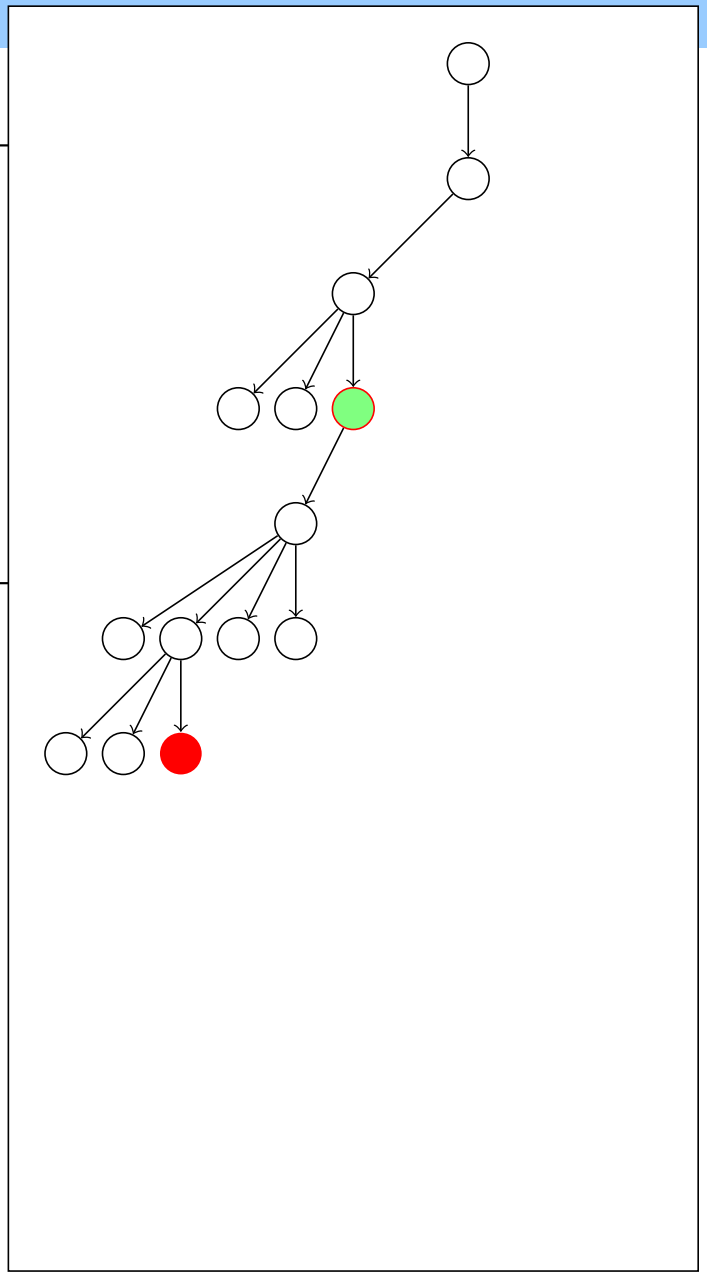
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

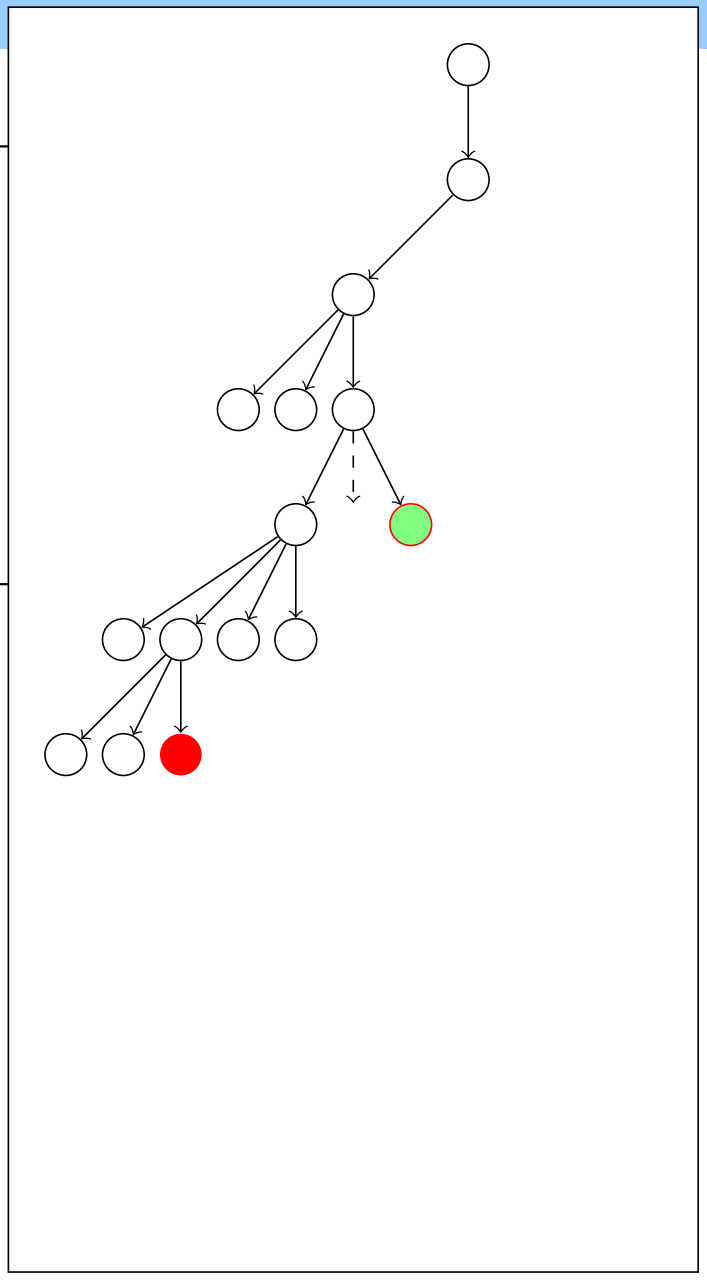
```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

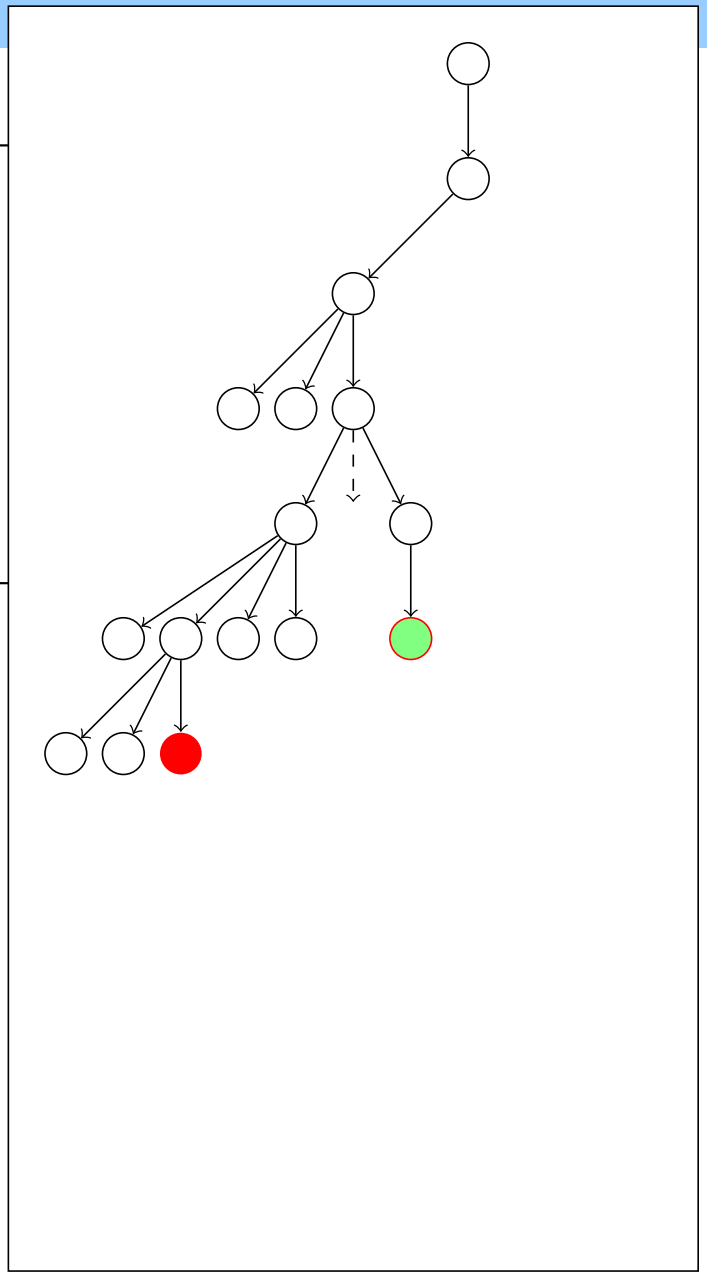
```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)

1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

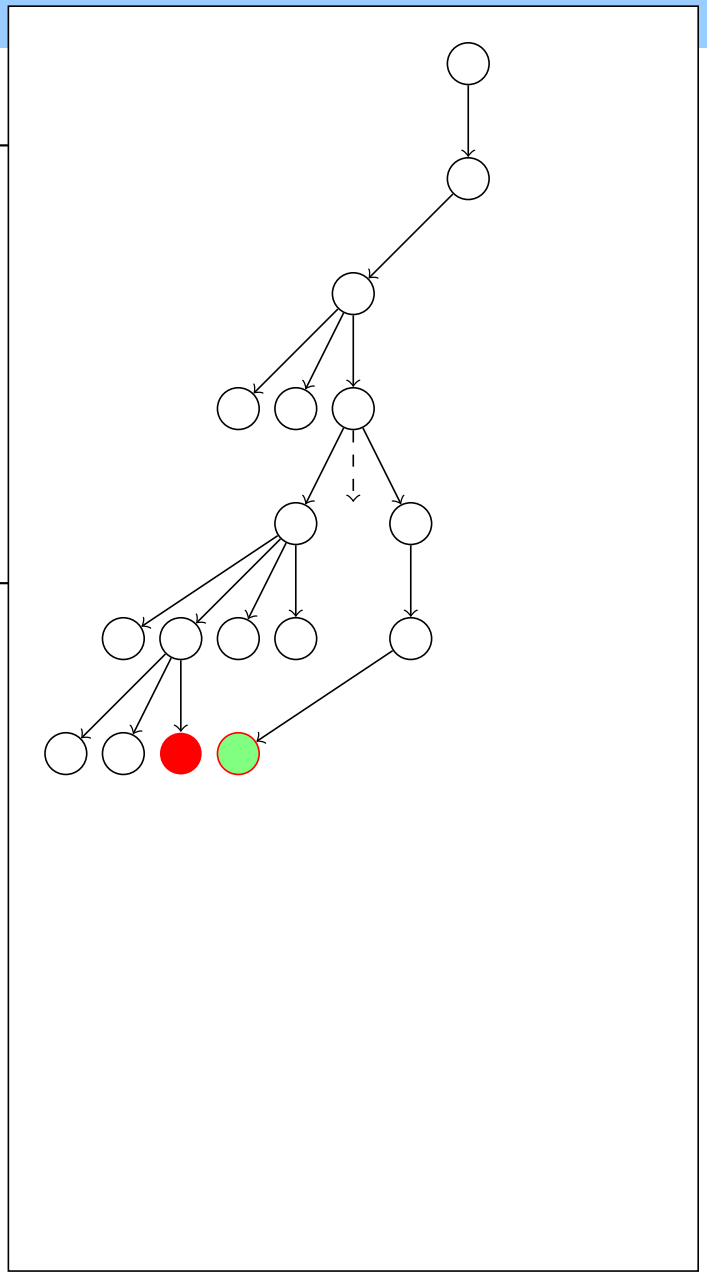
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

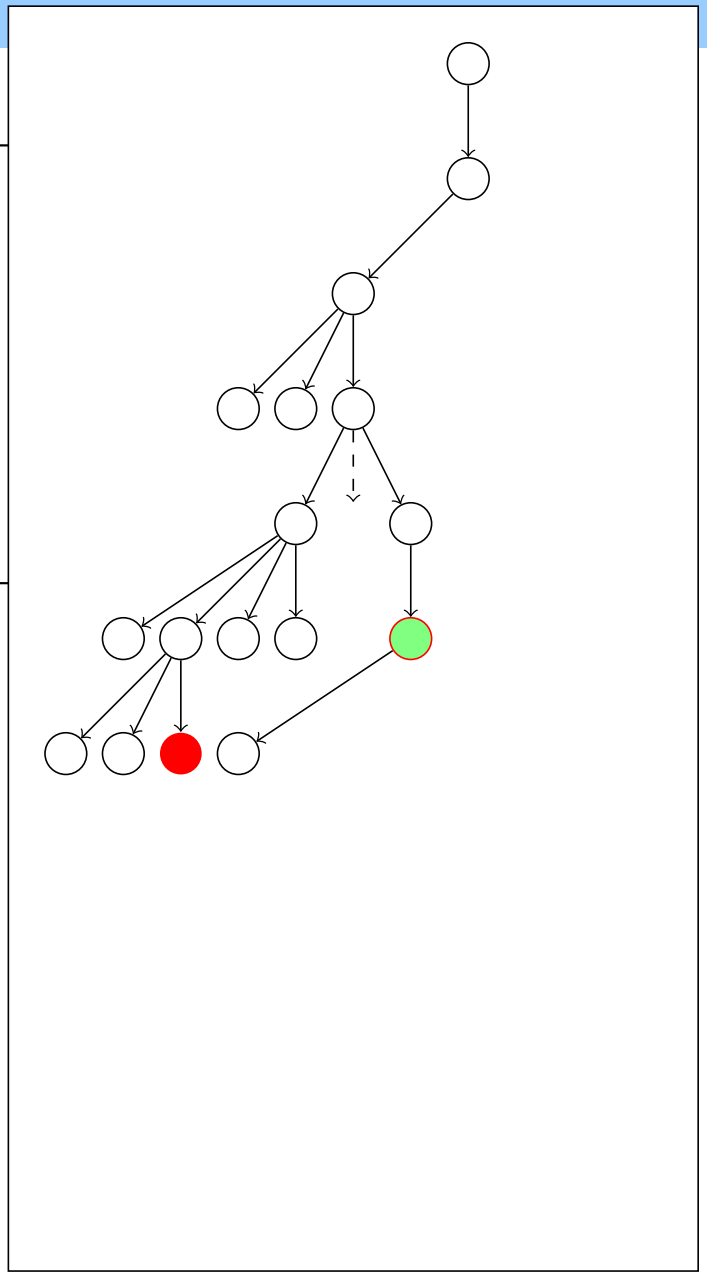
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

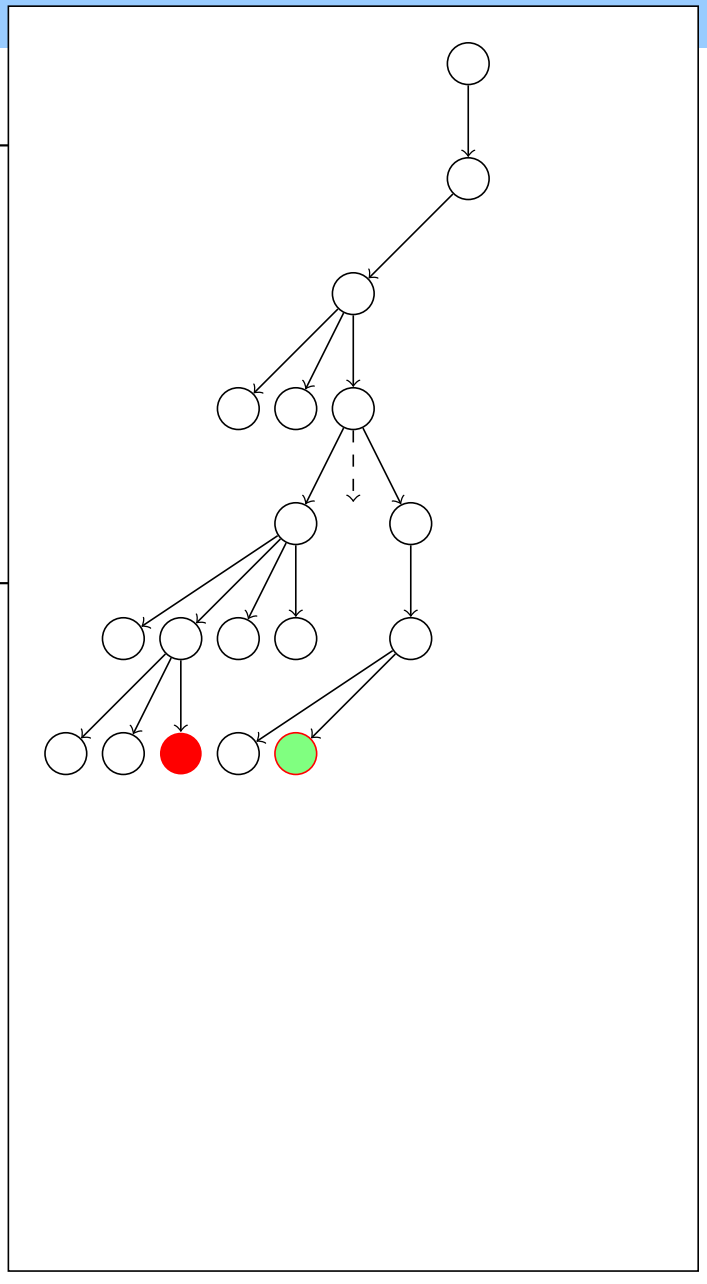
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

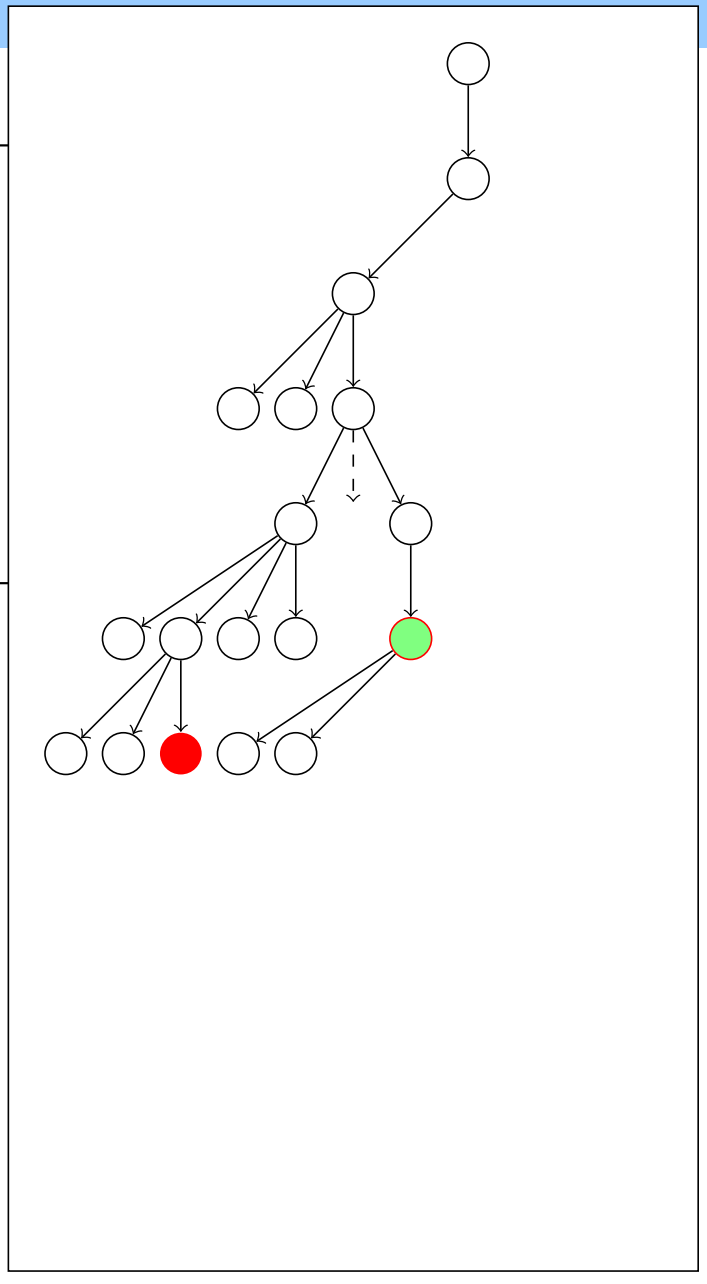
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

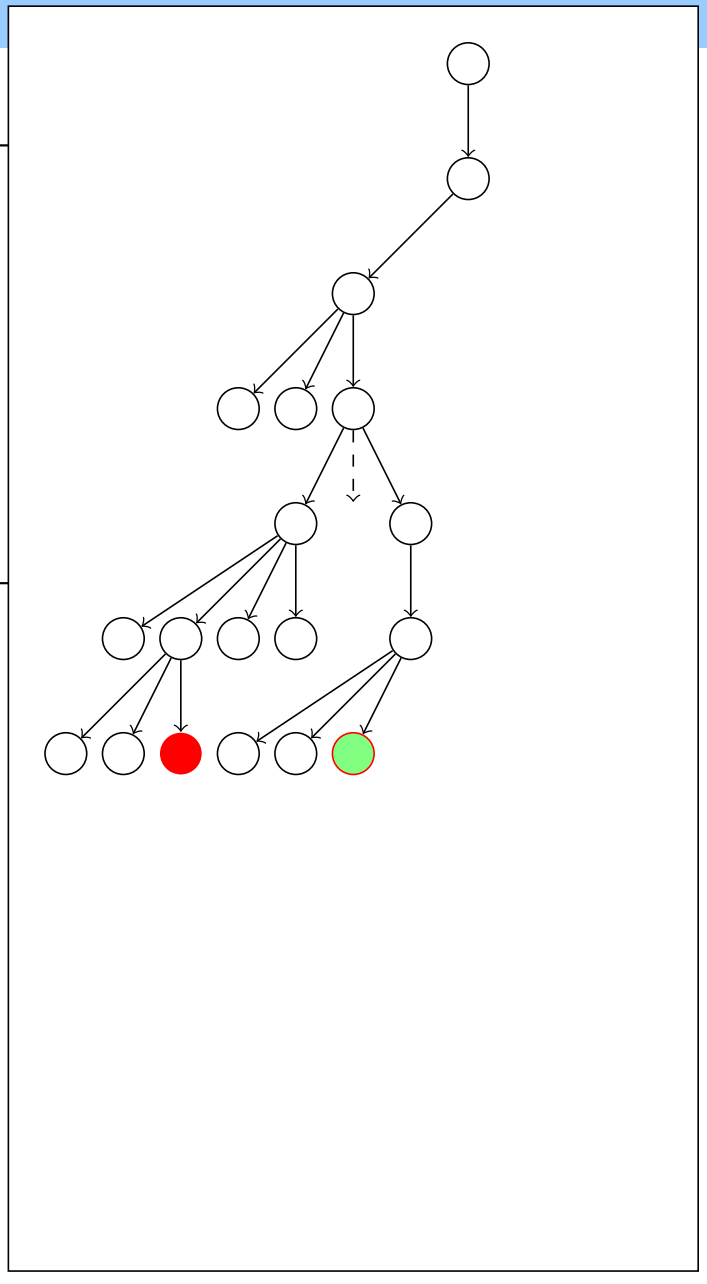
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

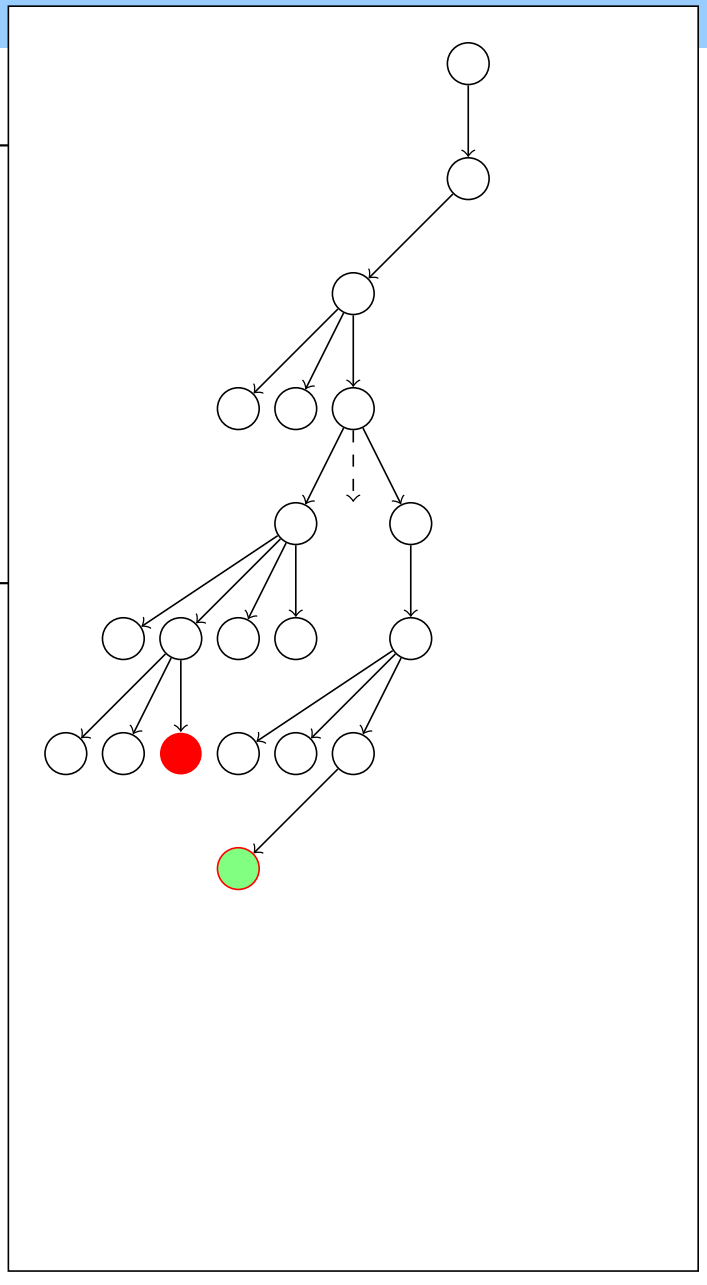
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

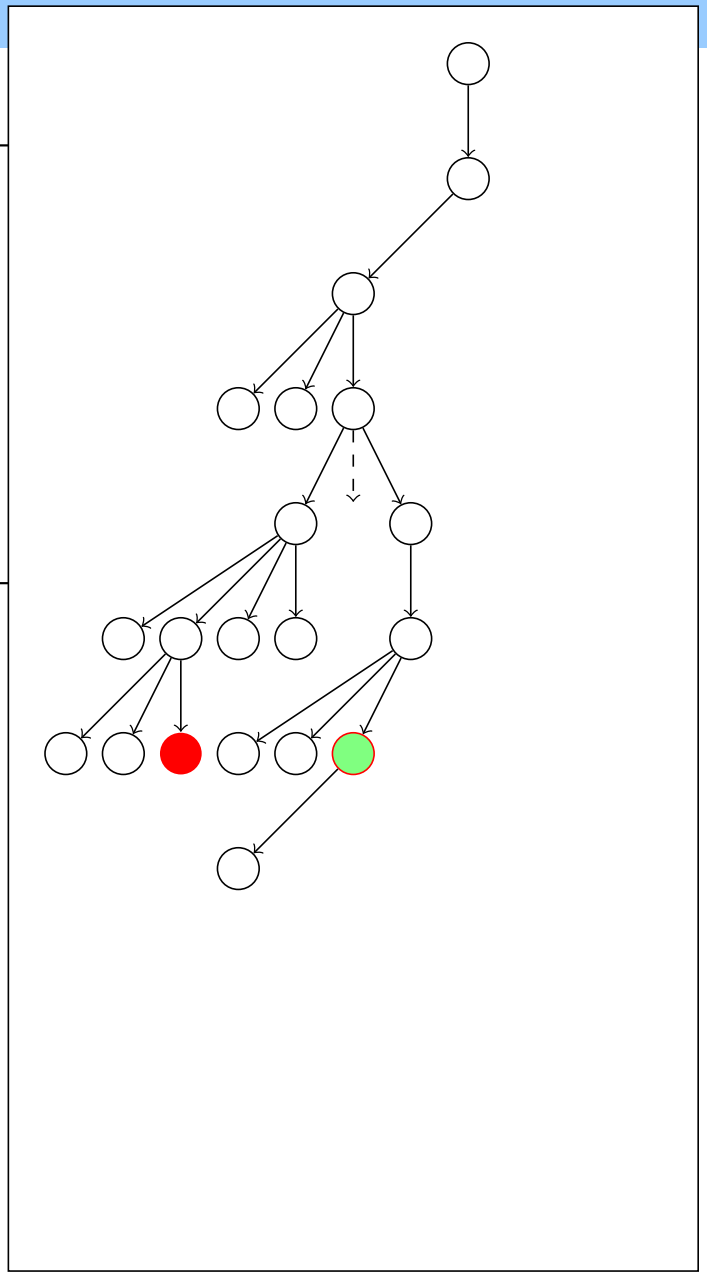
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

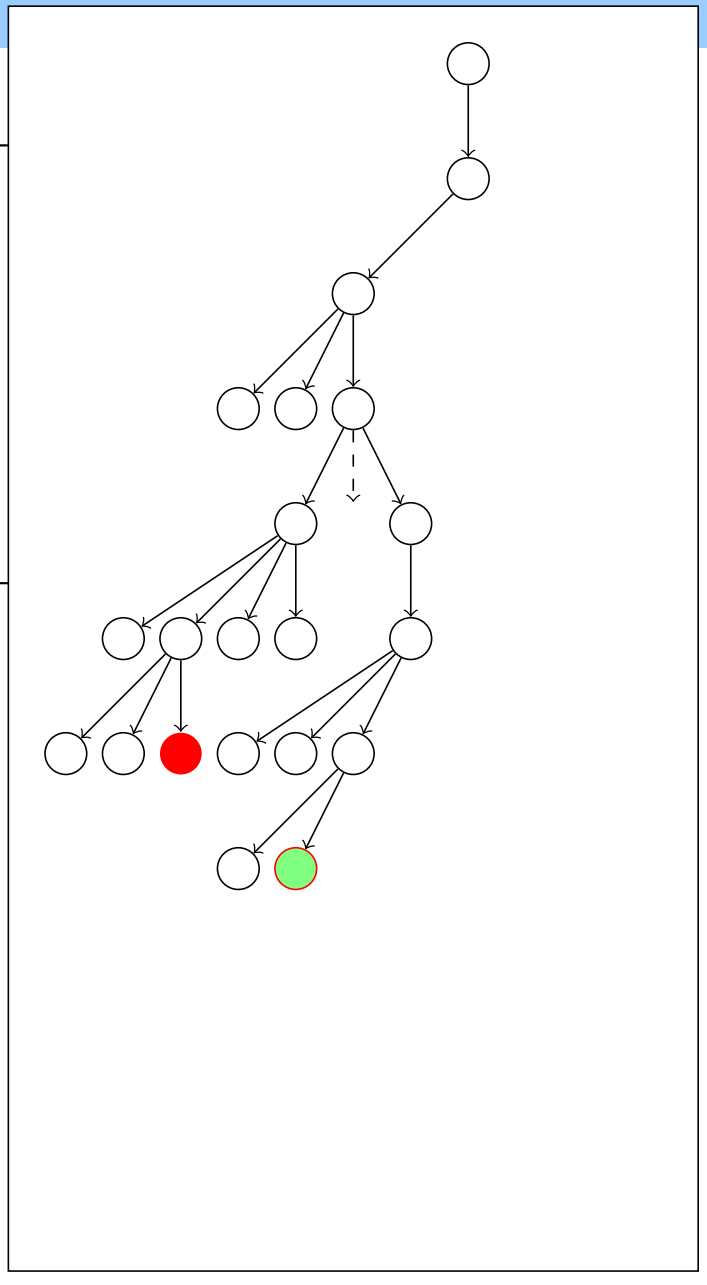
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

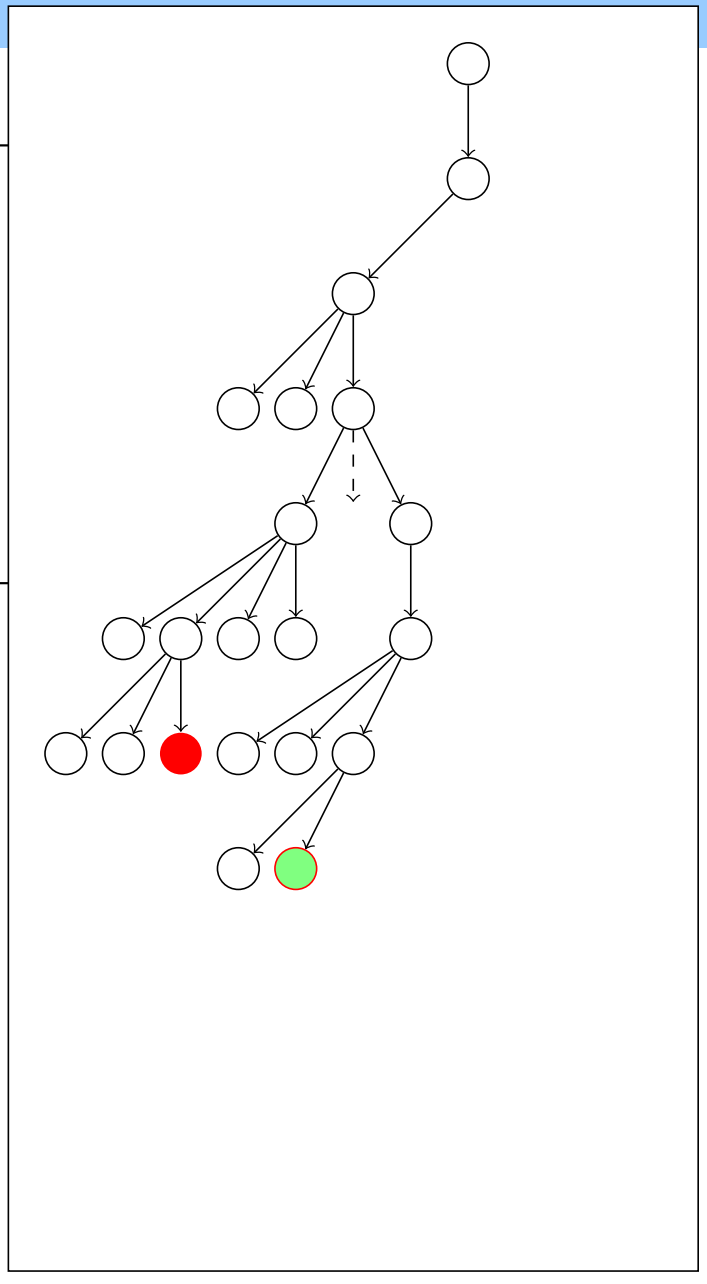
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

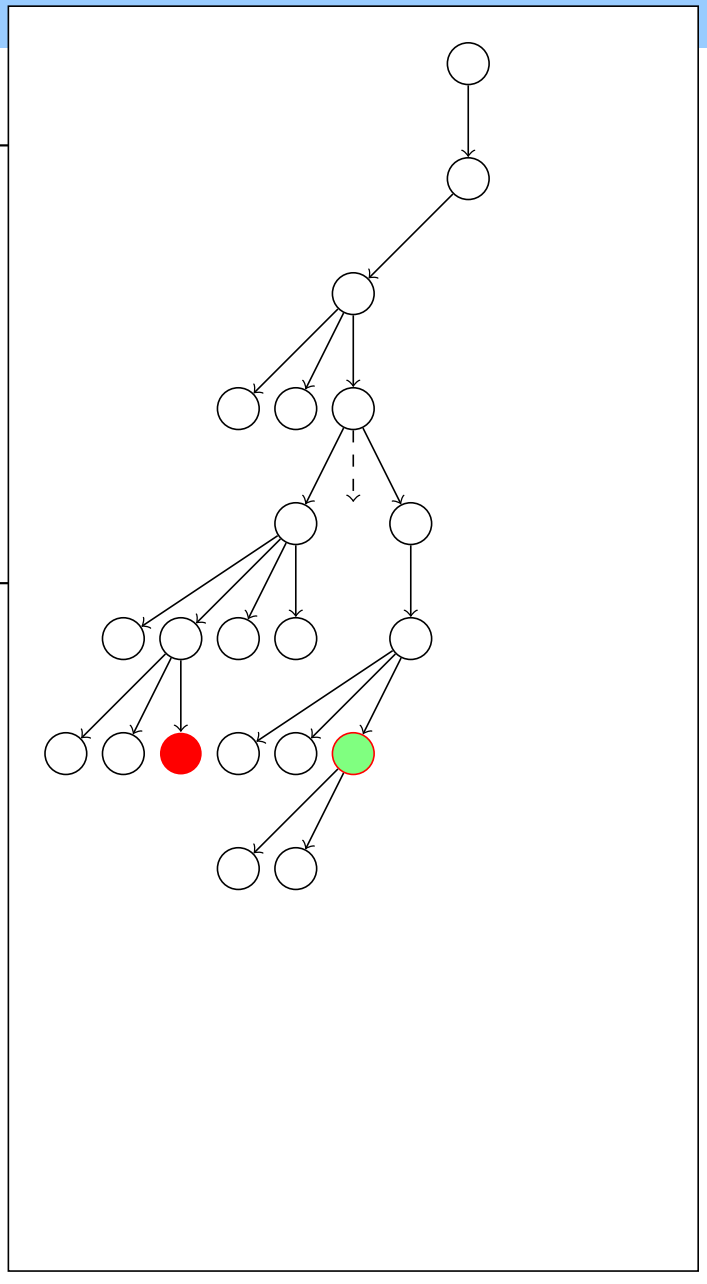
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

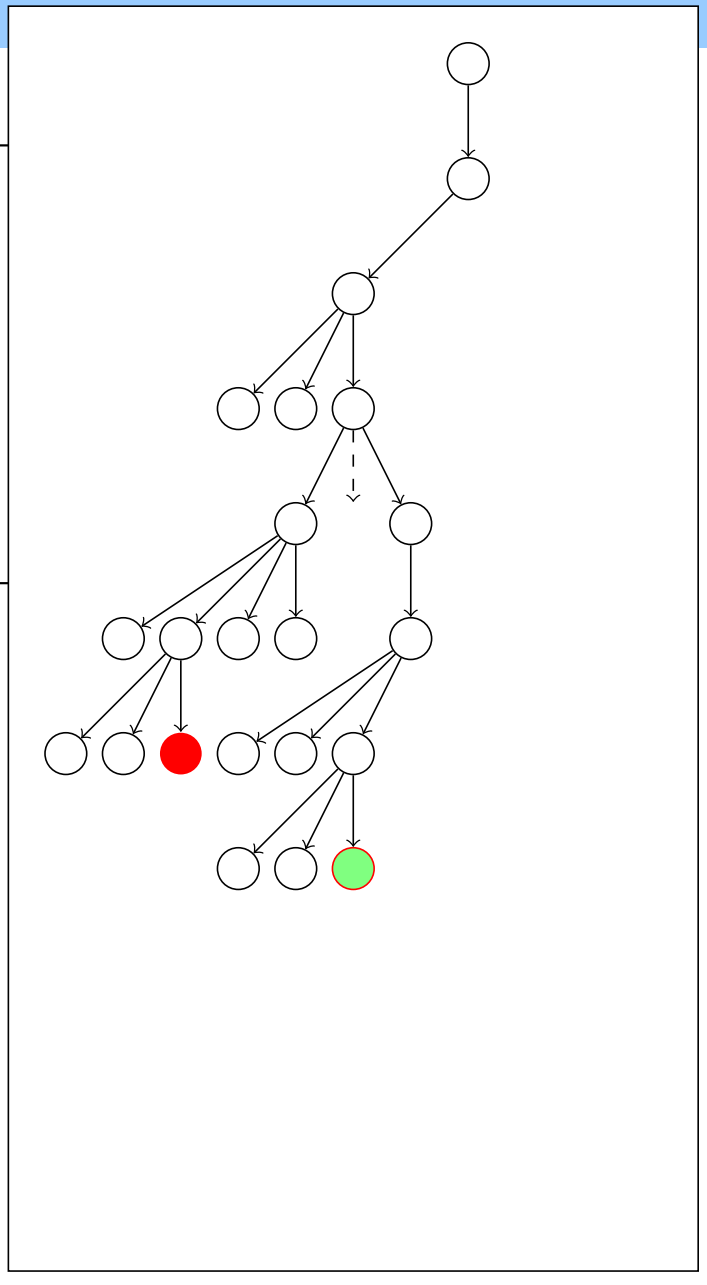
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

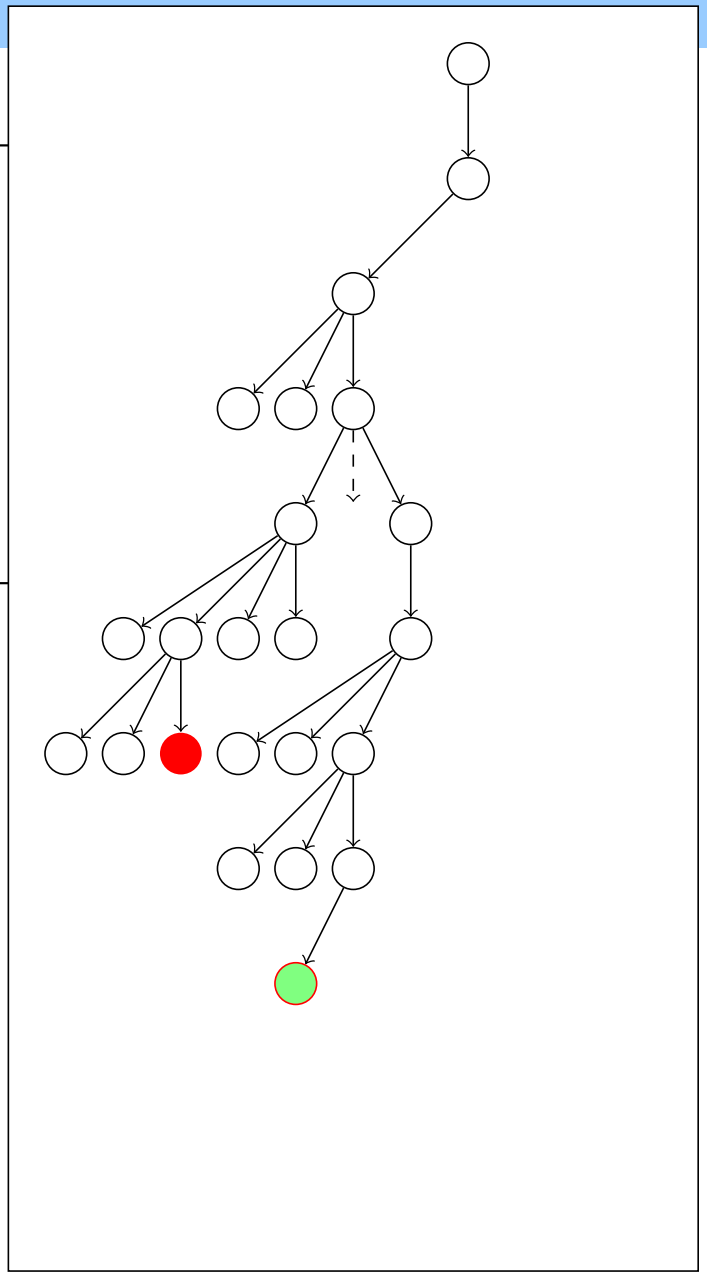
```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



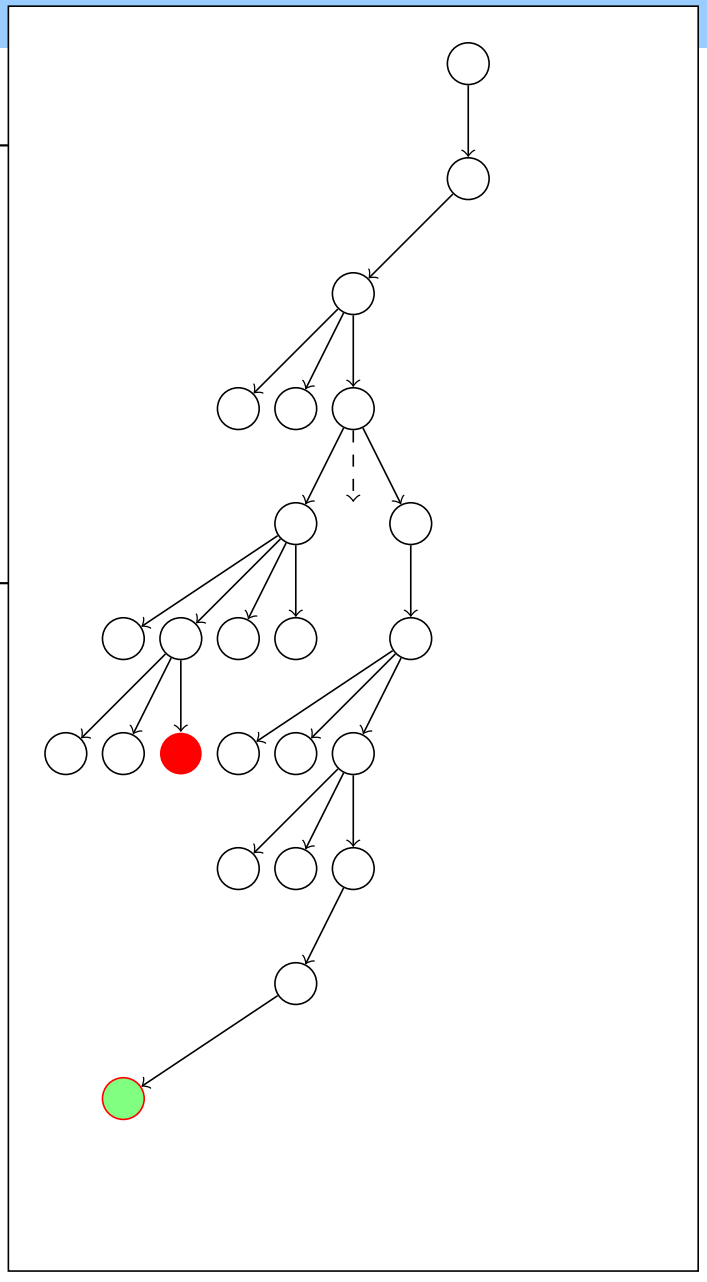
MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)

1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



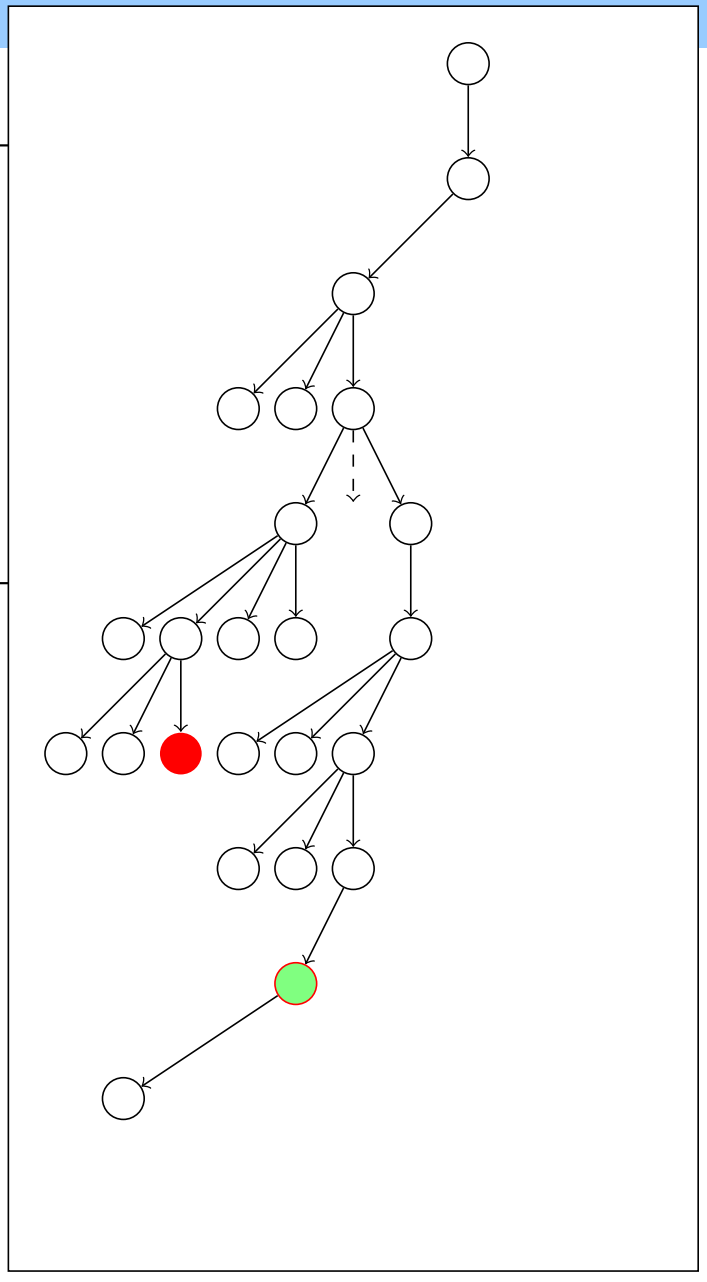
MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)

1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

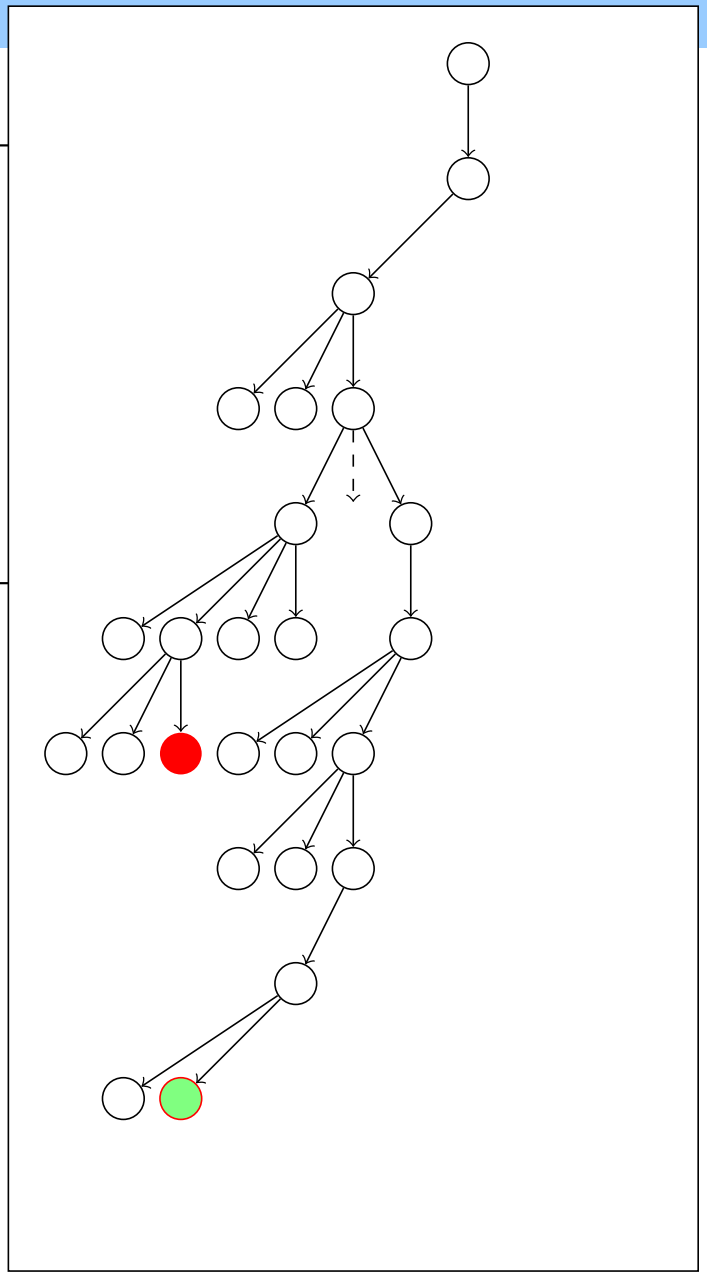
```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)

1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

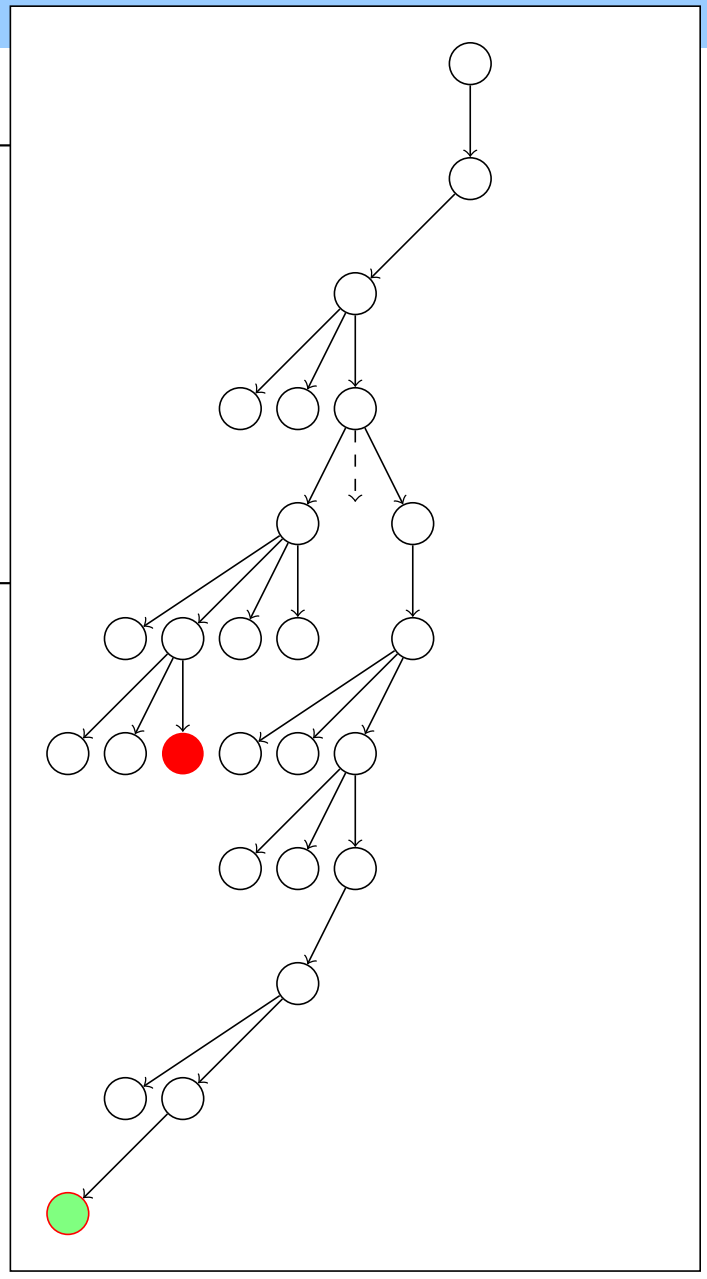
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

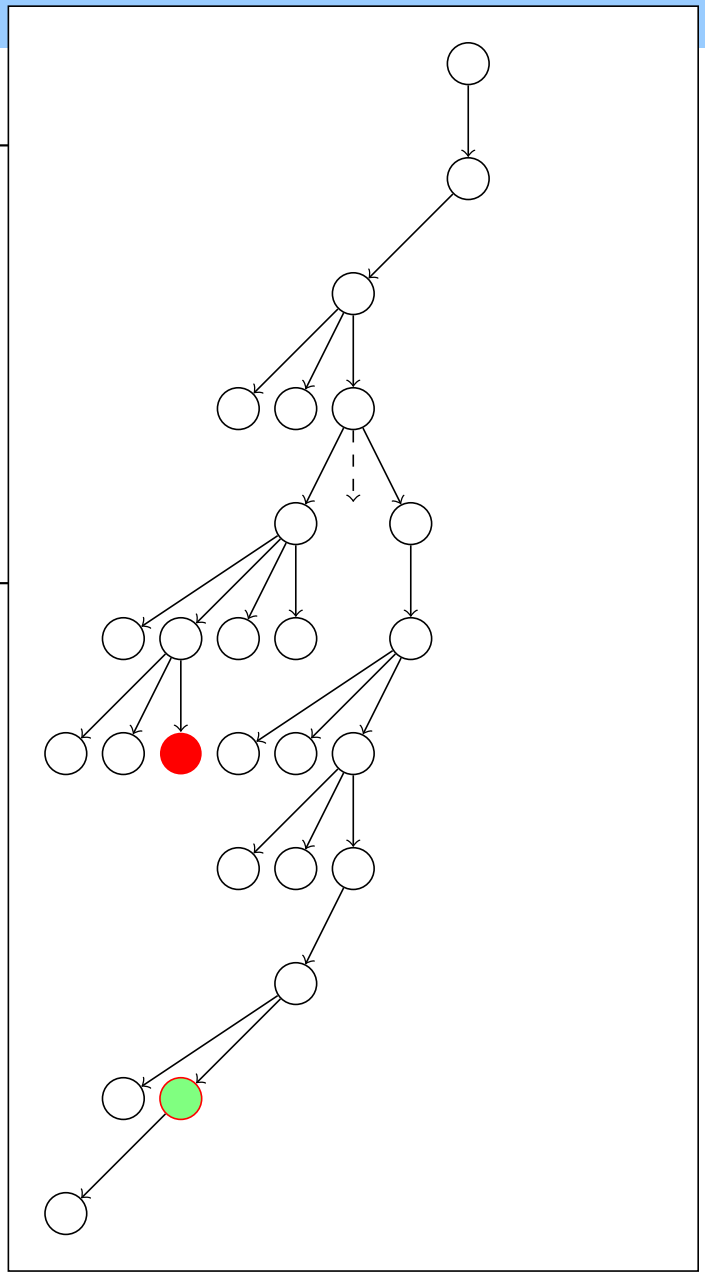
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

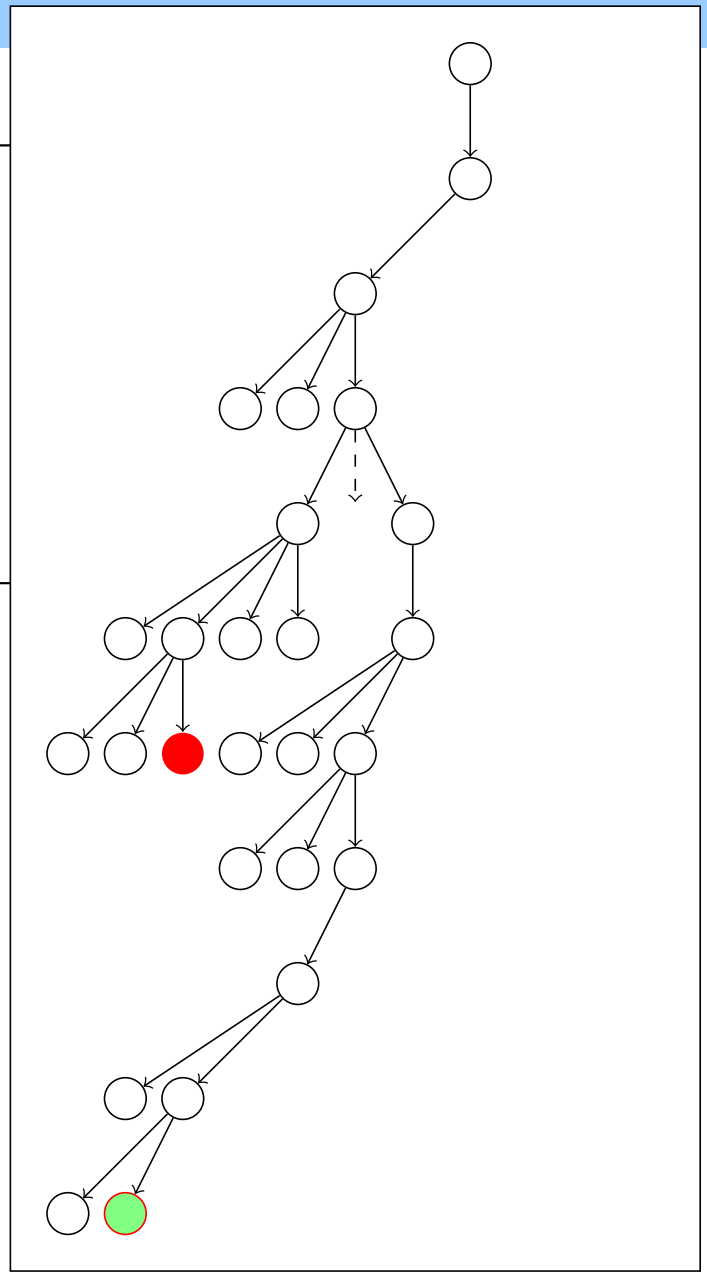
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

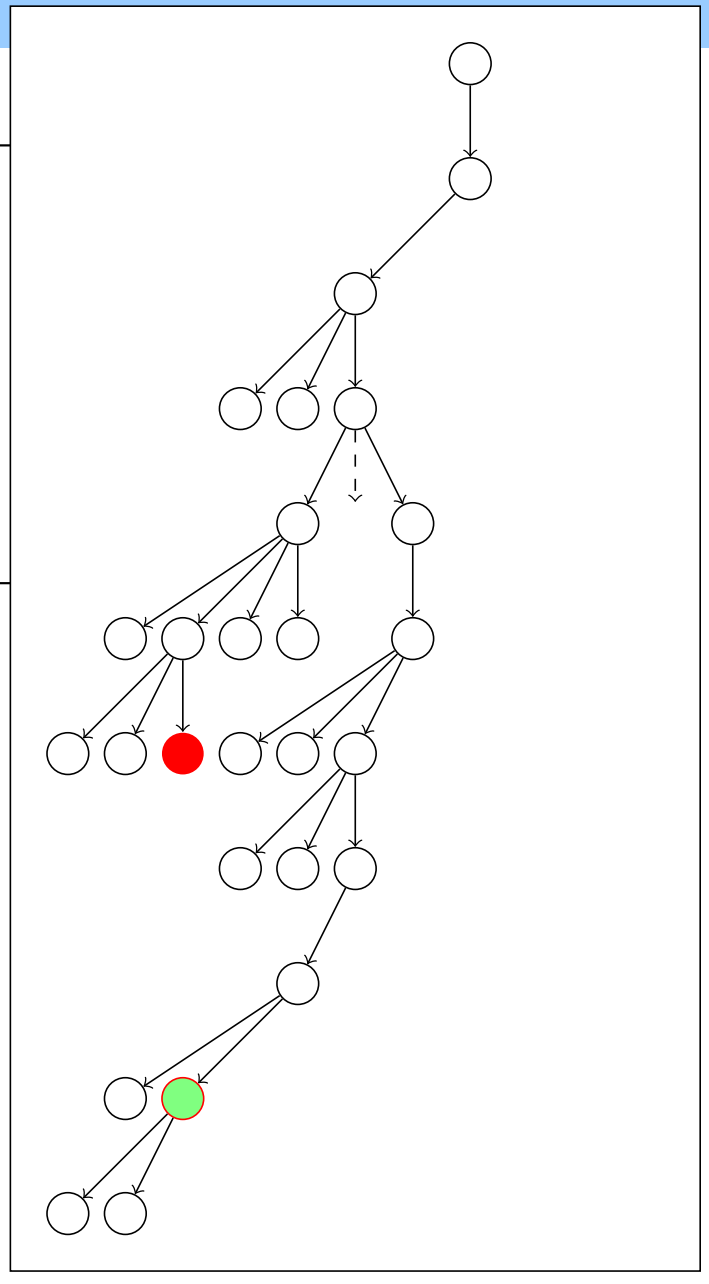
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

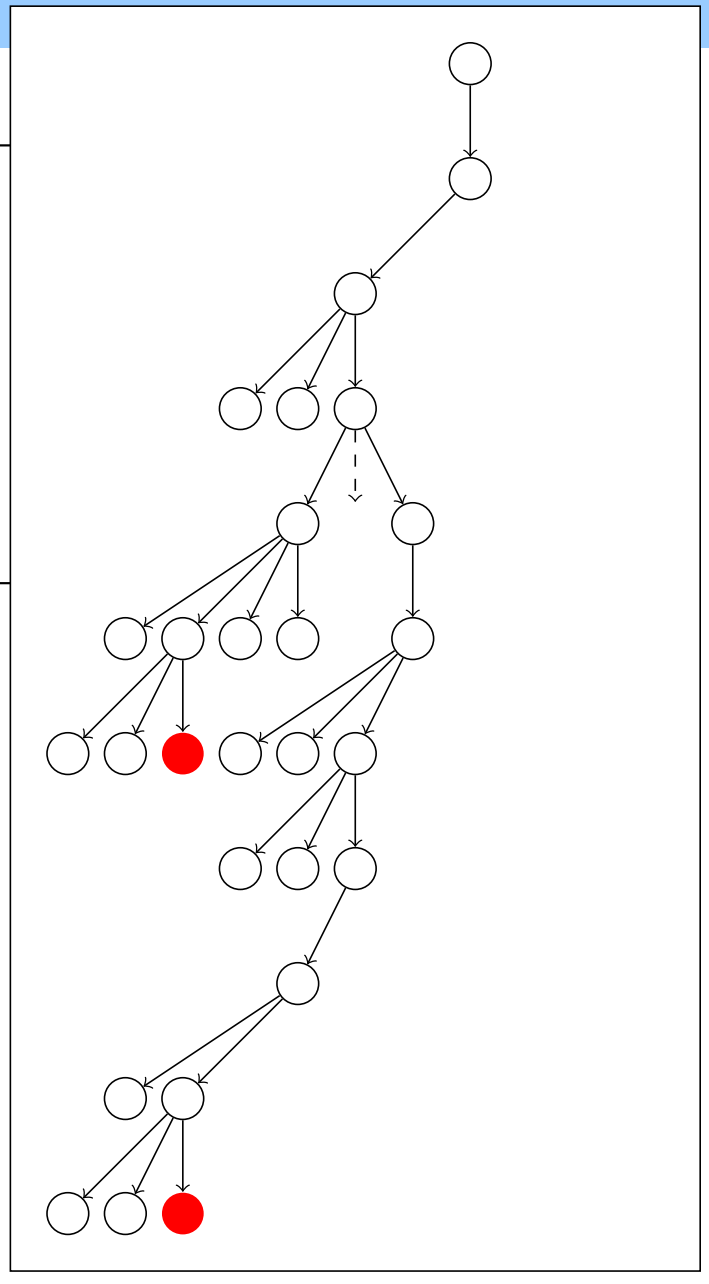
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

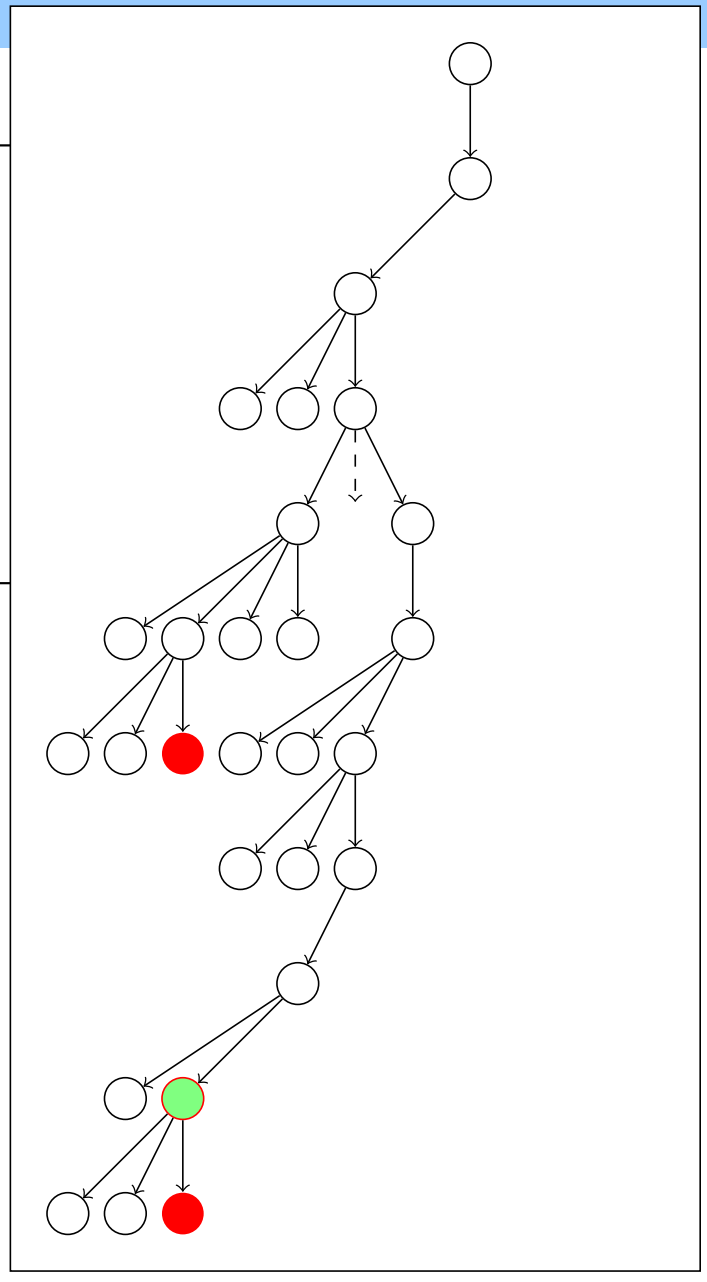
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

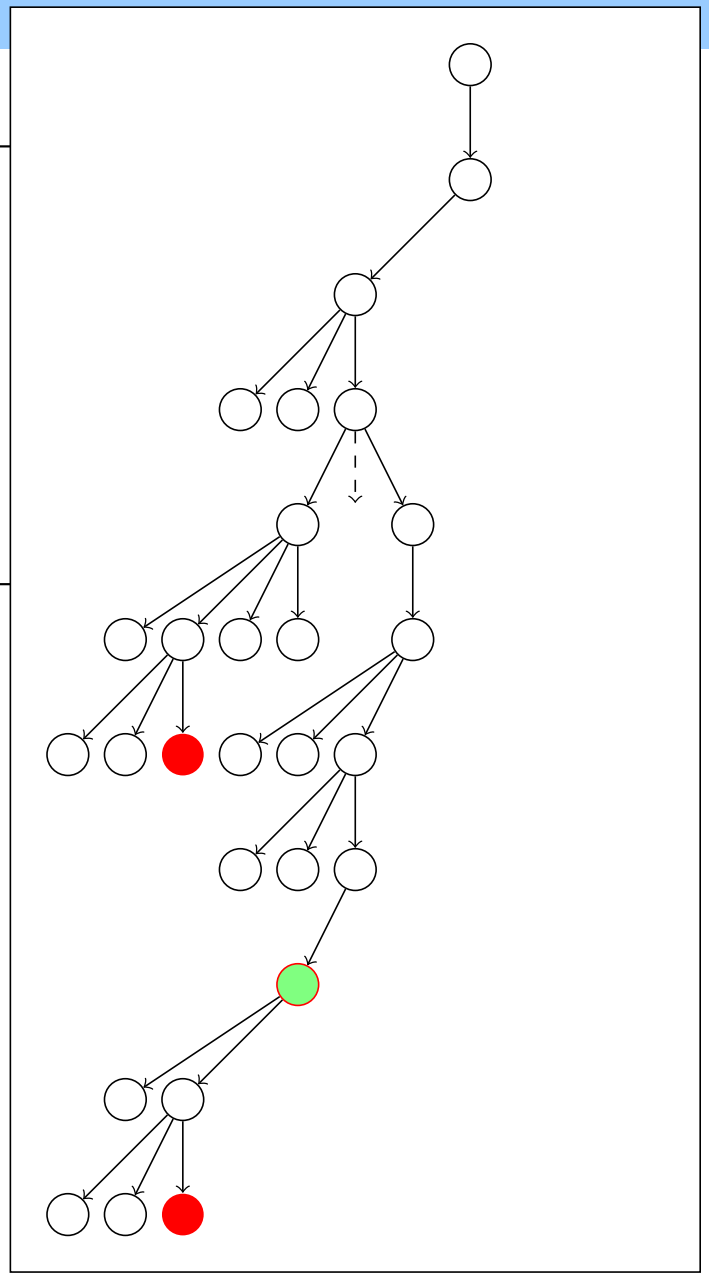
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
def geometric(p):
    if bernoulli(p) == 1: return 1
    else: return 1 + geometric(p)
```

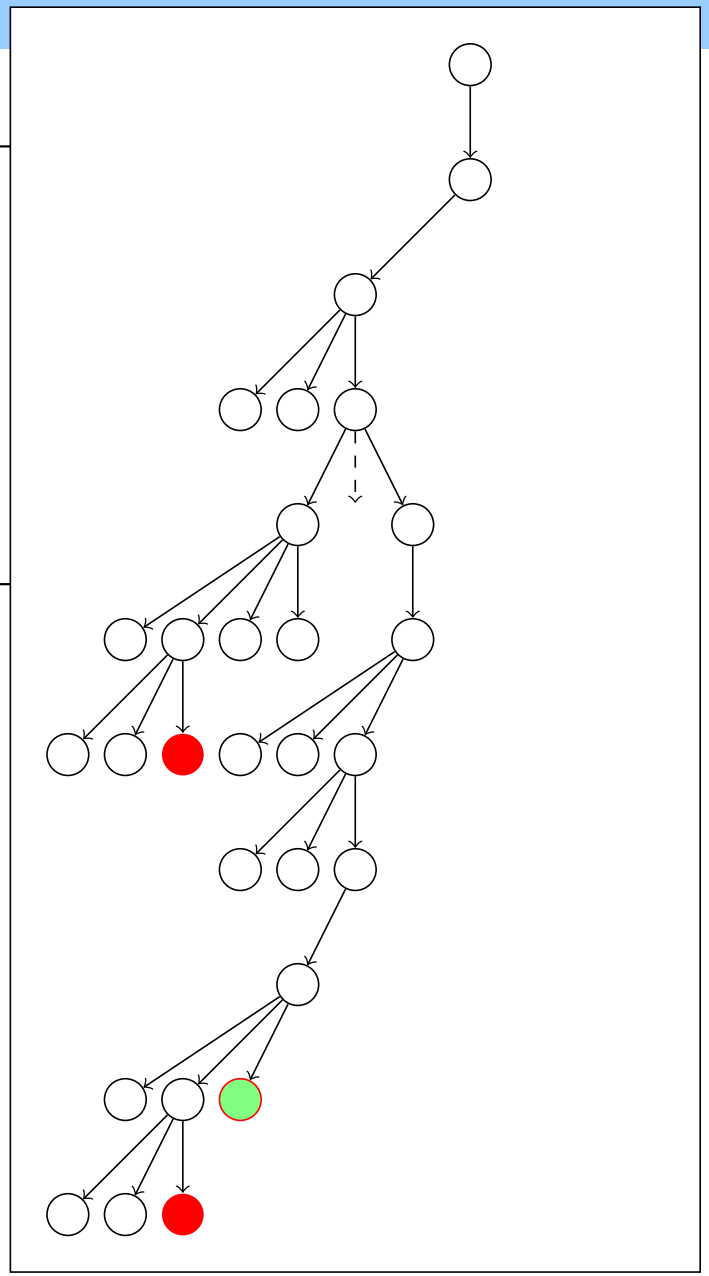
```
def aliased_geometric(p):
    g = geometric(p)
    return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

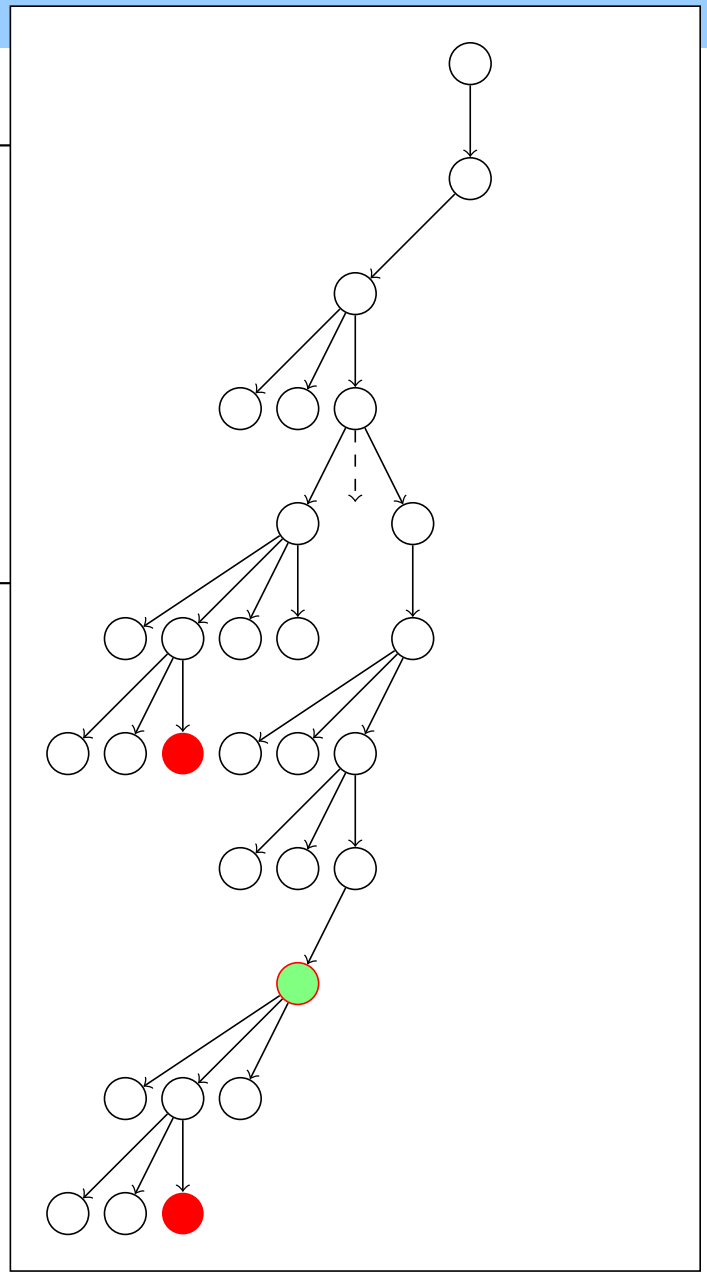
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

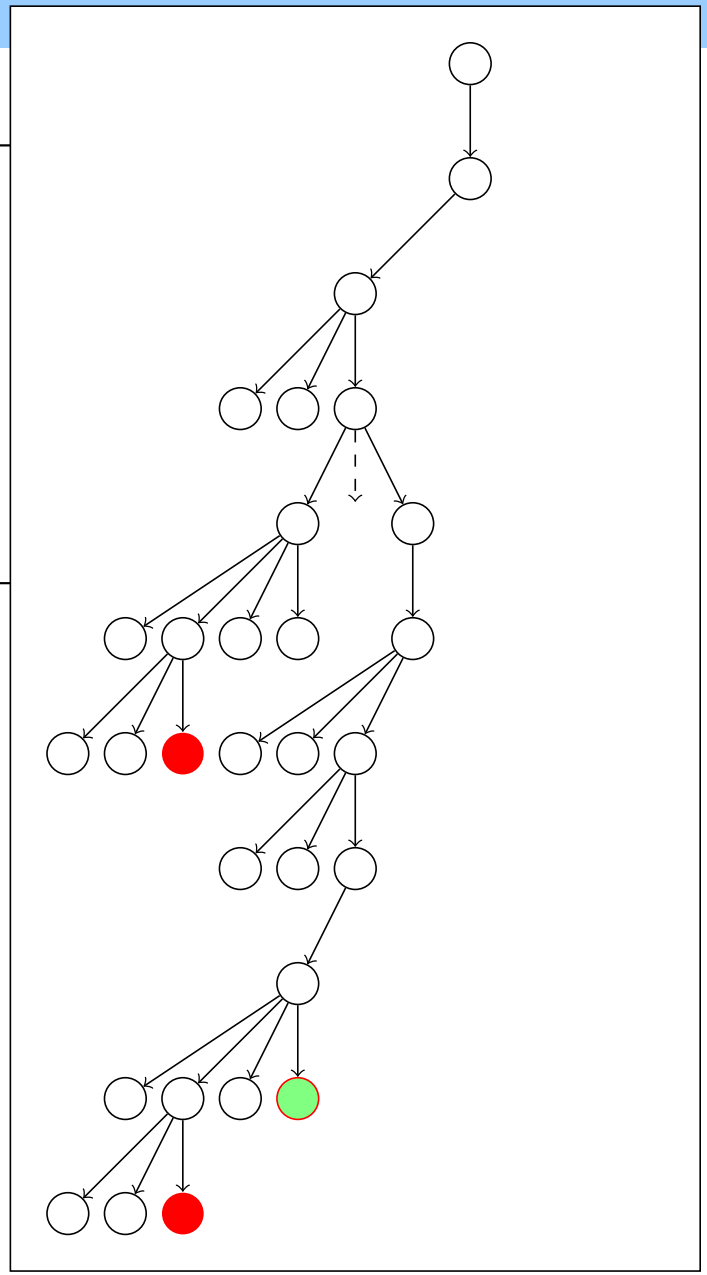
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
def geometric(p):
    if bernoulli(p) == 1: return 1
    else: return 1 + geometric(p)
```

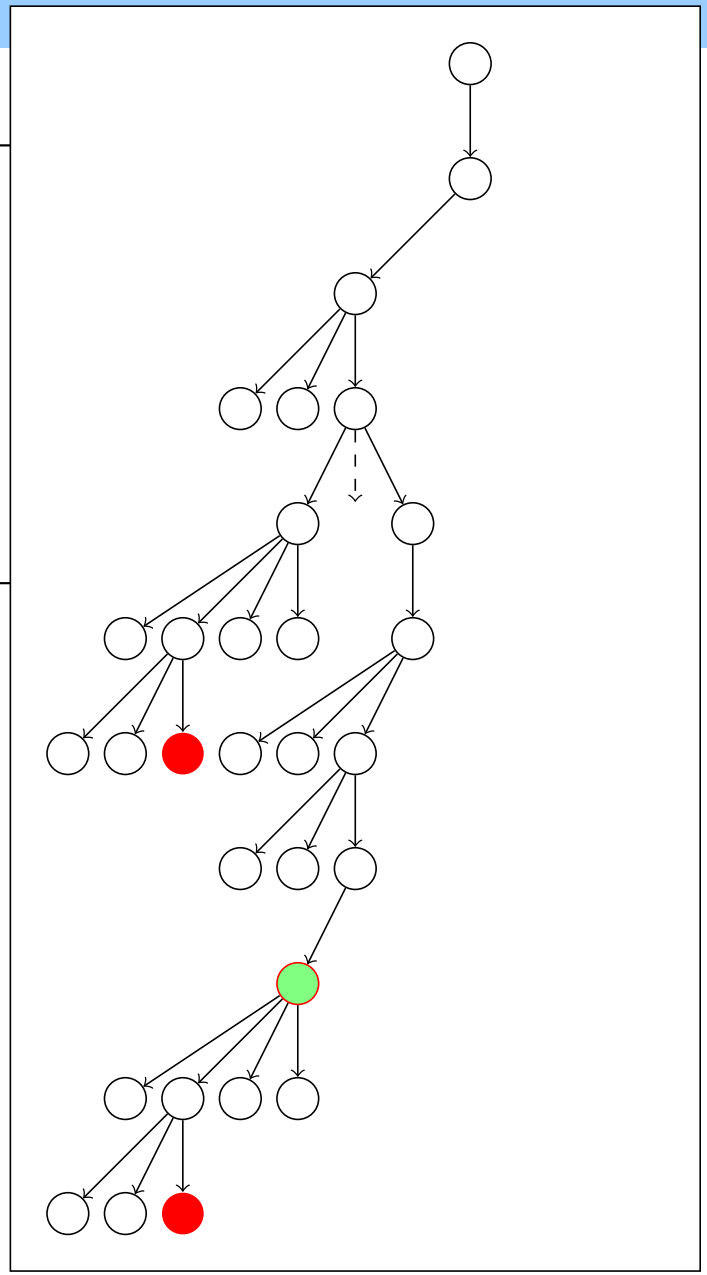
```
def aliased_geometric(p):
    g = geometric(p)
    return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
def geometric(p):
    if bernoulli(p) == 1: return 1
    else: return 1 + geometric(p)
```

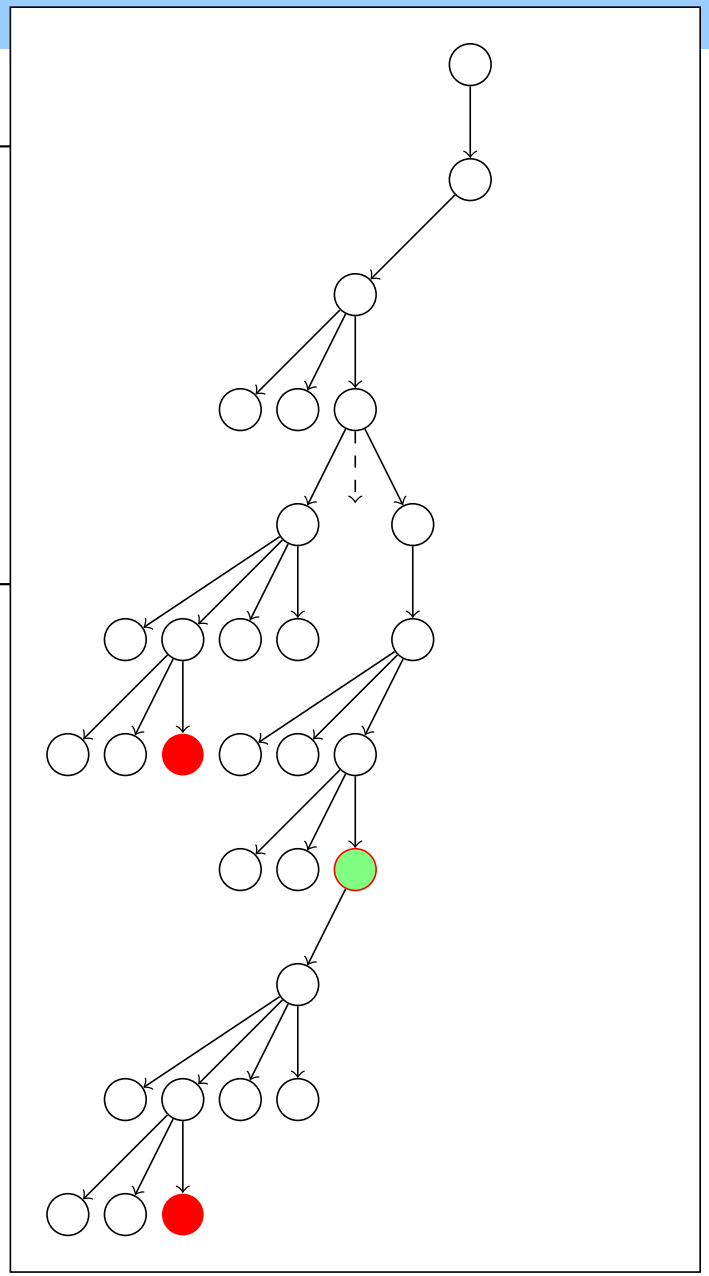
```
def aliased_geometric(p):
    g = geometric(p)
    return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

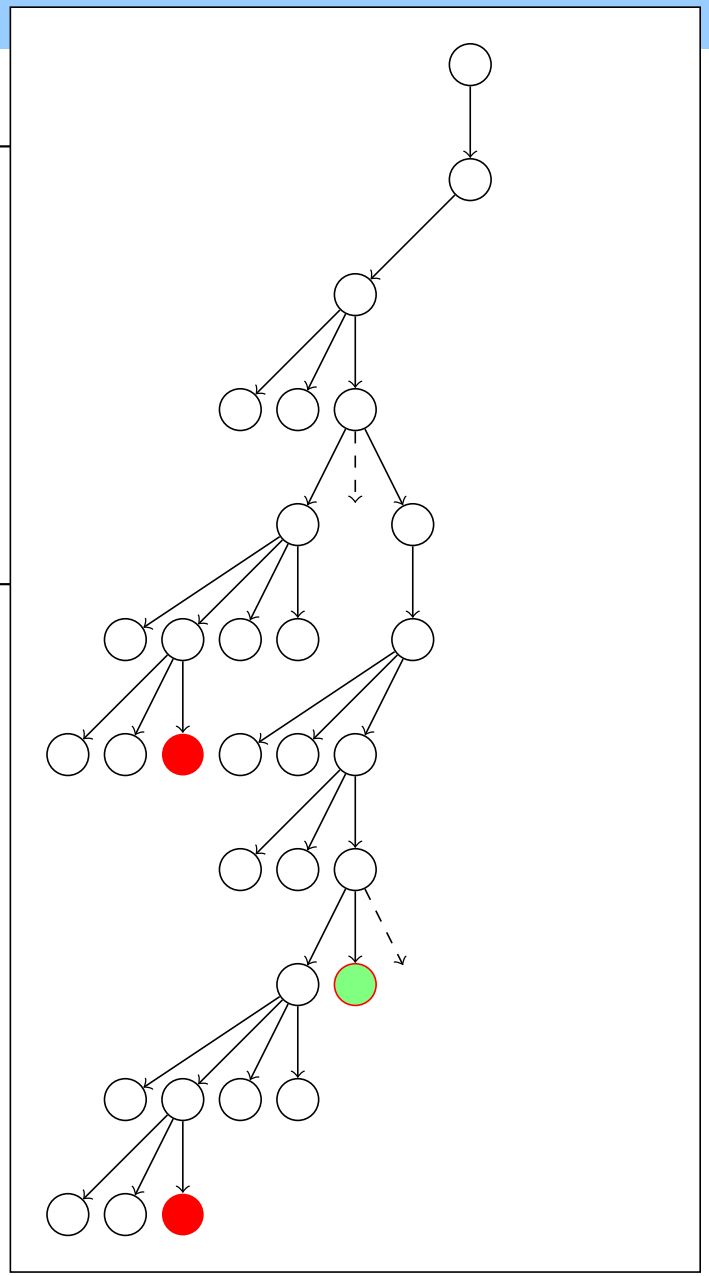
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

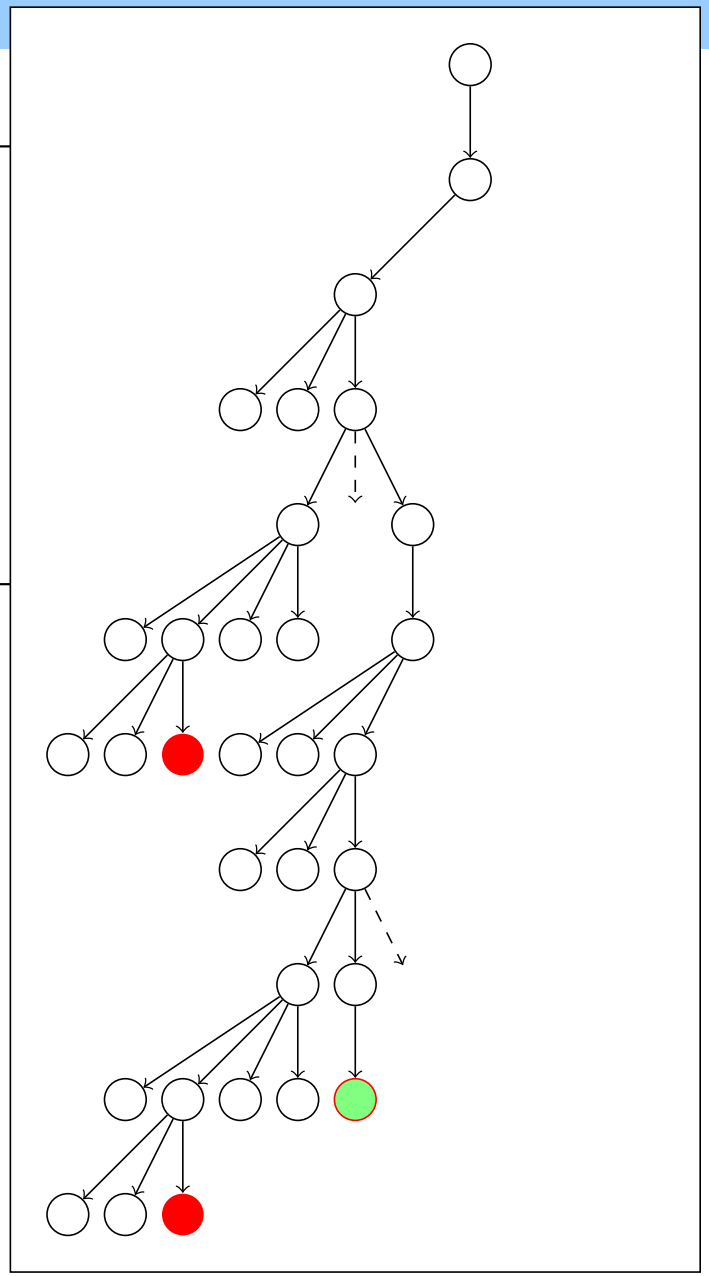
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

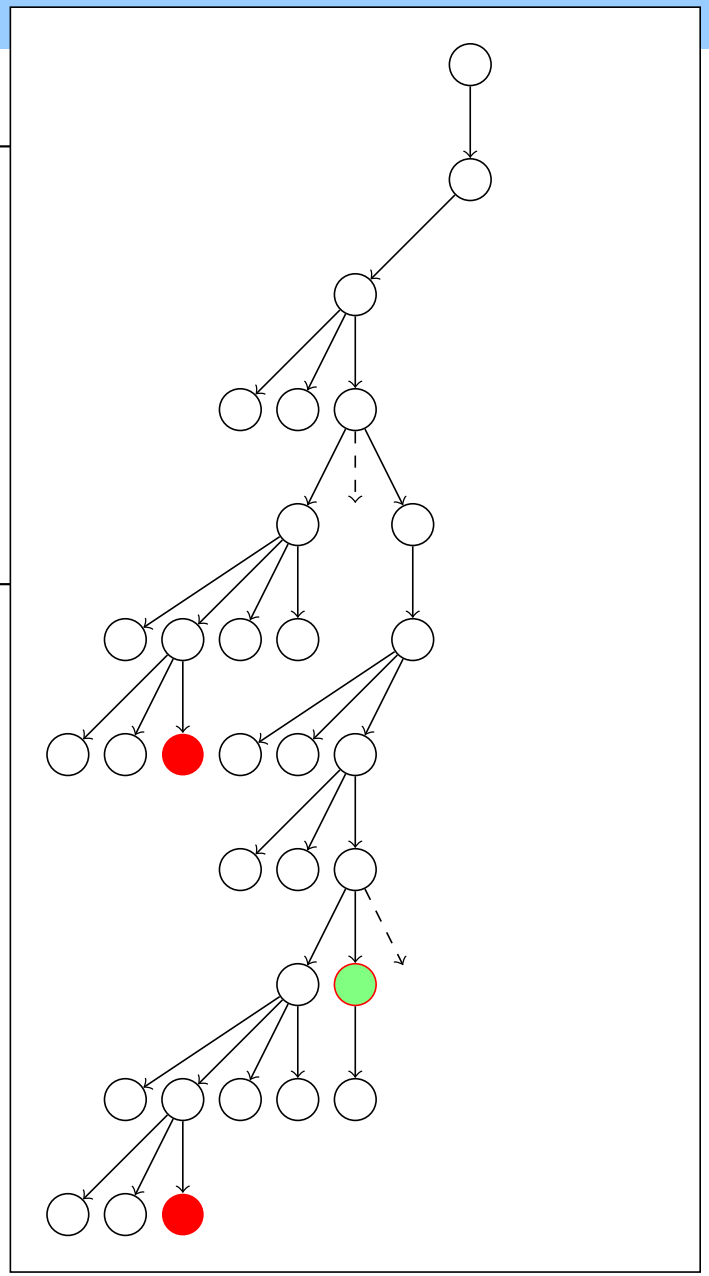
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
def geometric(p):
    if bernoulli(p) == 1: return 1
    else: return 1 + geometric(p)
```

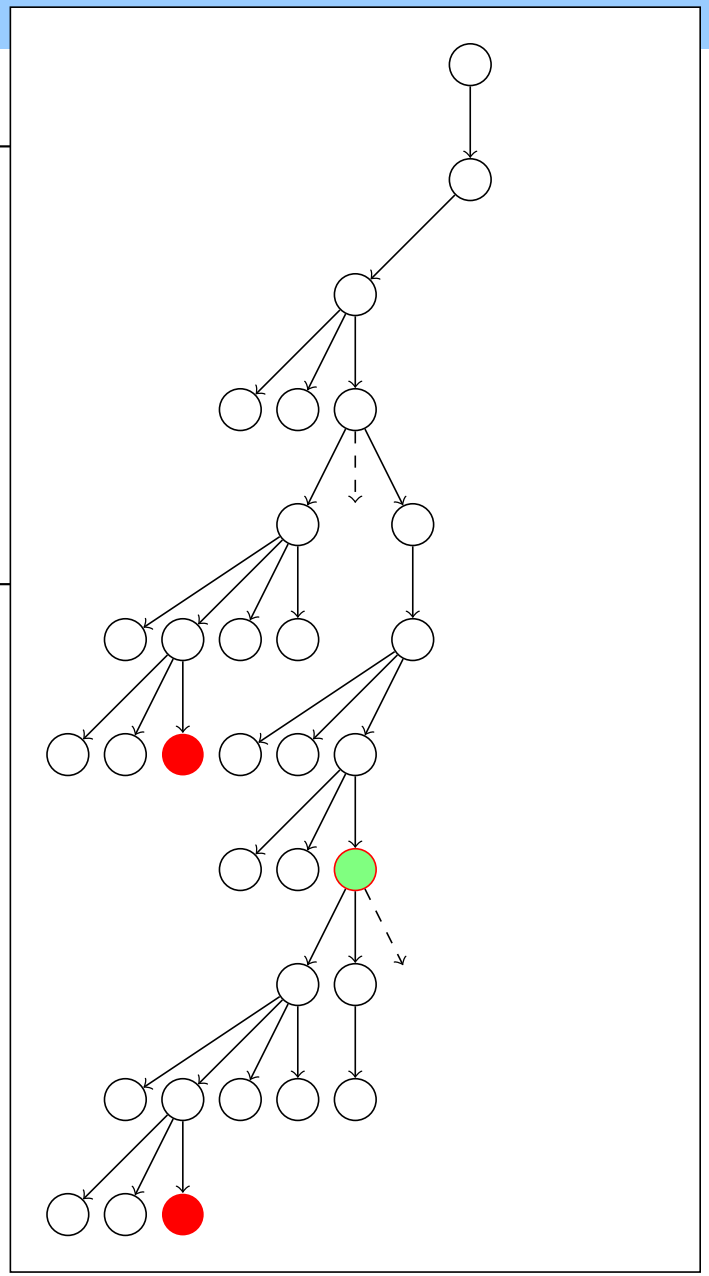
```
def aliased_geometric(p):
    g = geometric(p)
    return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

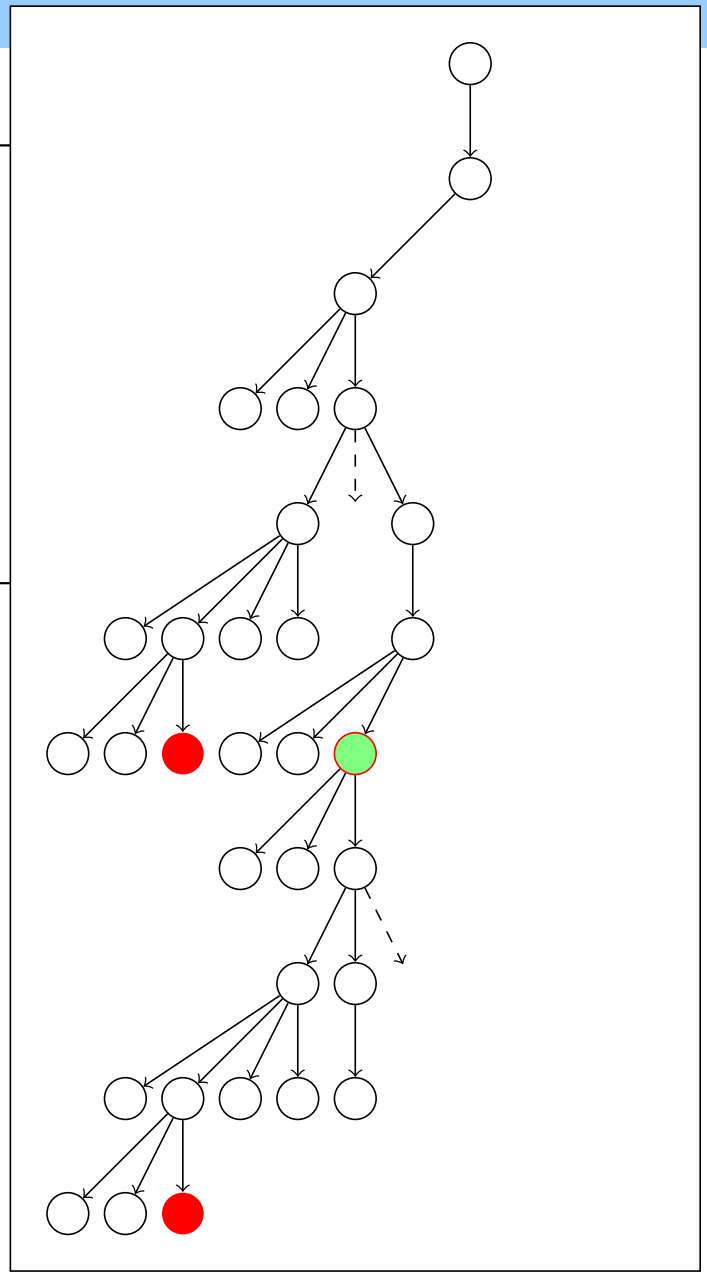
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

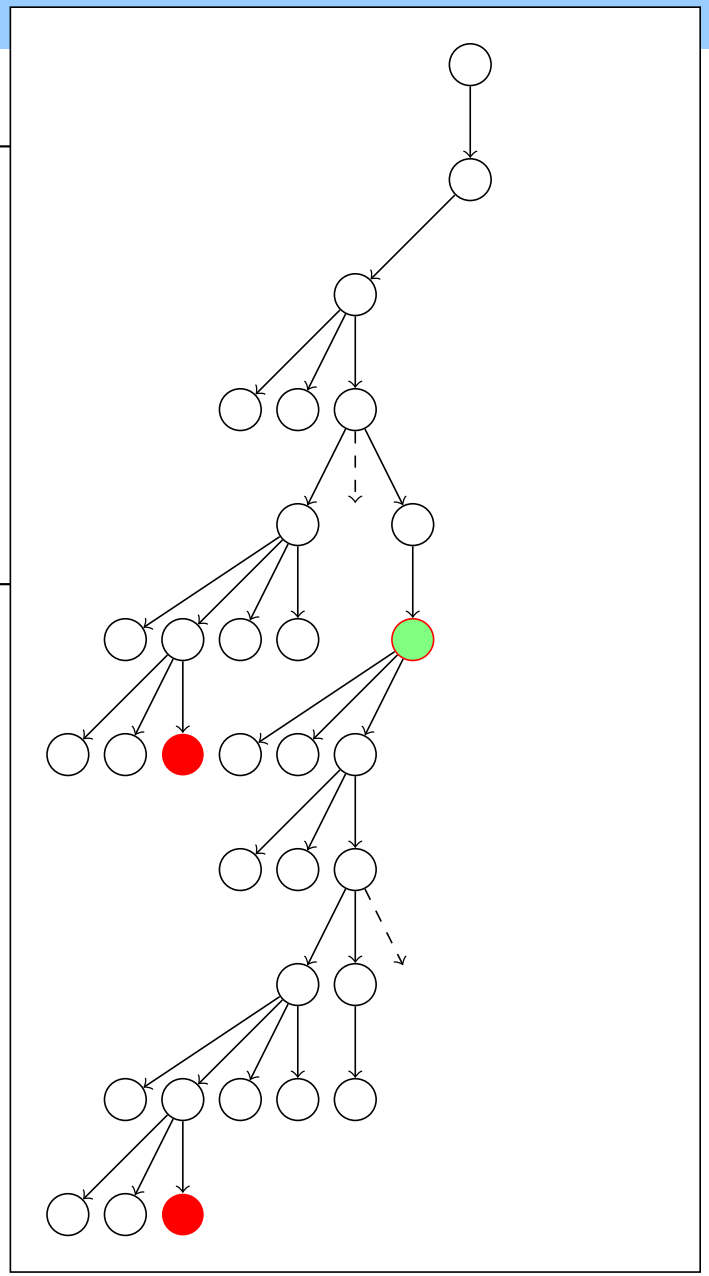
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
def geometric(p):
    if bernoulli(p) == 1: return 1
    else: return 1 + geometric(p)
```

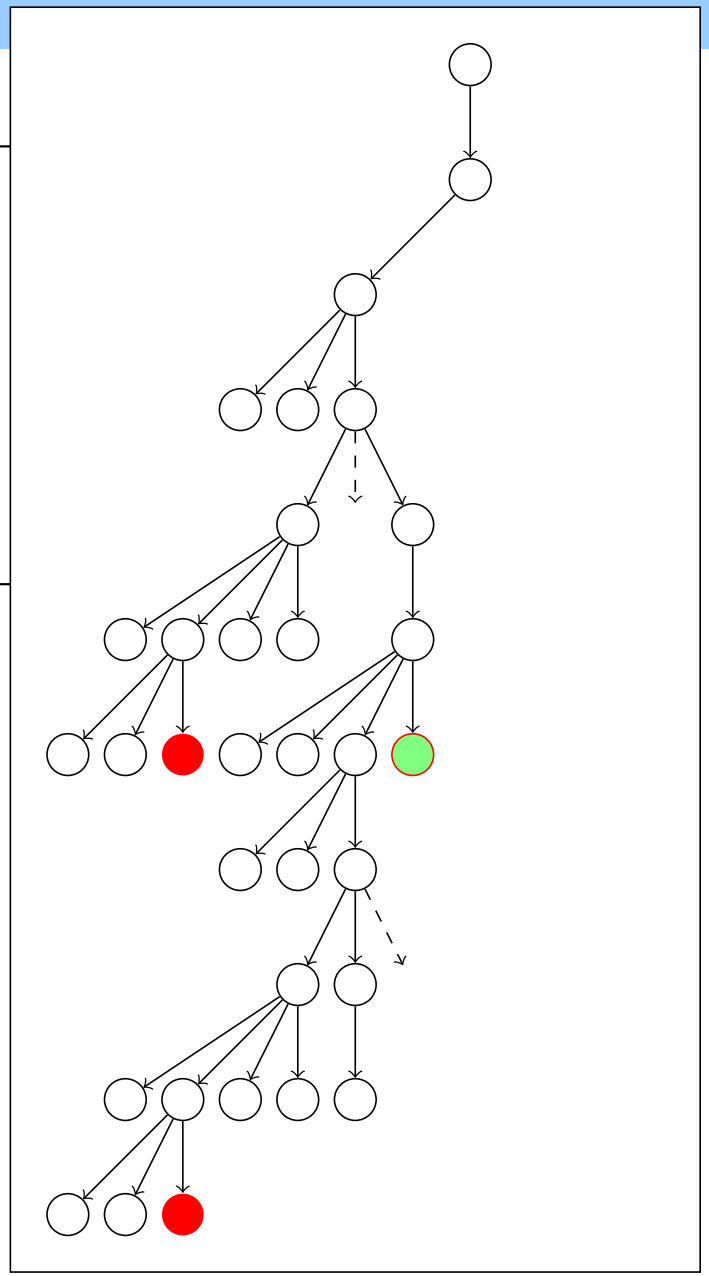
```
def aliased_geometric(p):
    g = geometric(p)
    return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
def geometric(p):
    if bernoulli(p) == 1: return 1
    else: return 1 + geometric(p)
```

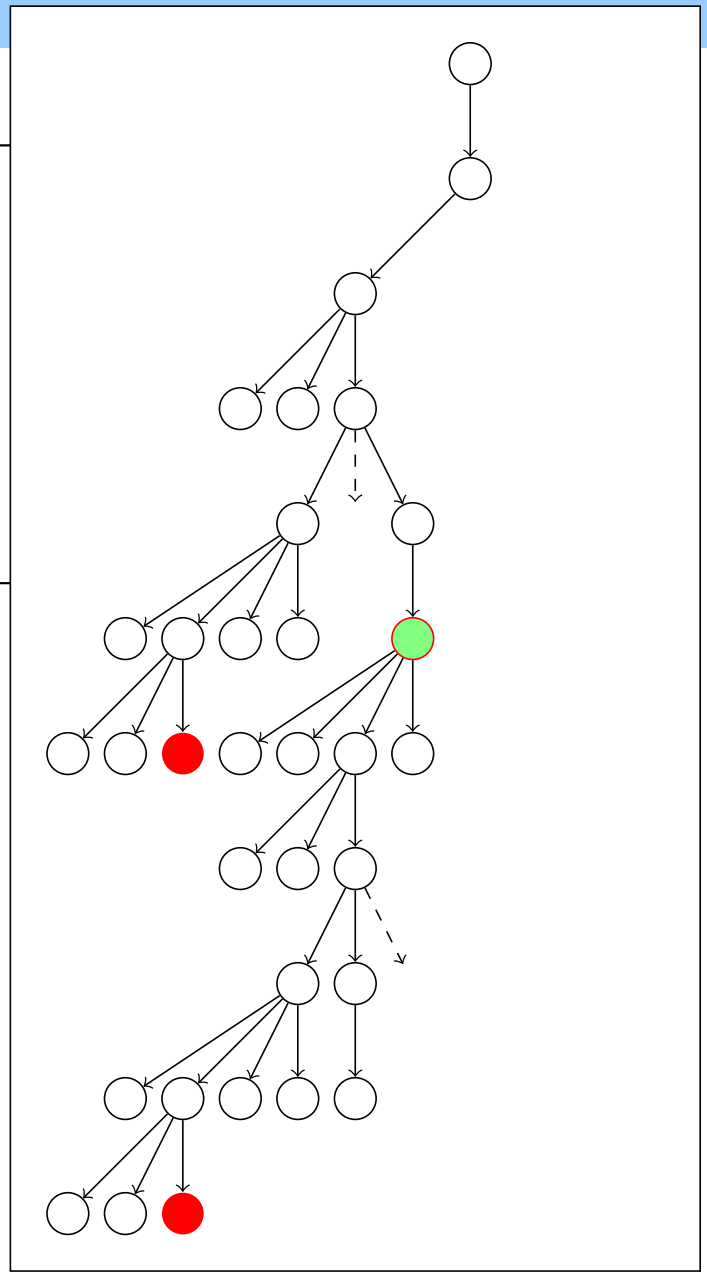
```
def aliased_geometric(p):
    g = geometric(p)
    return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

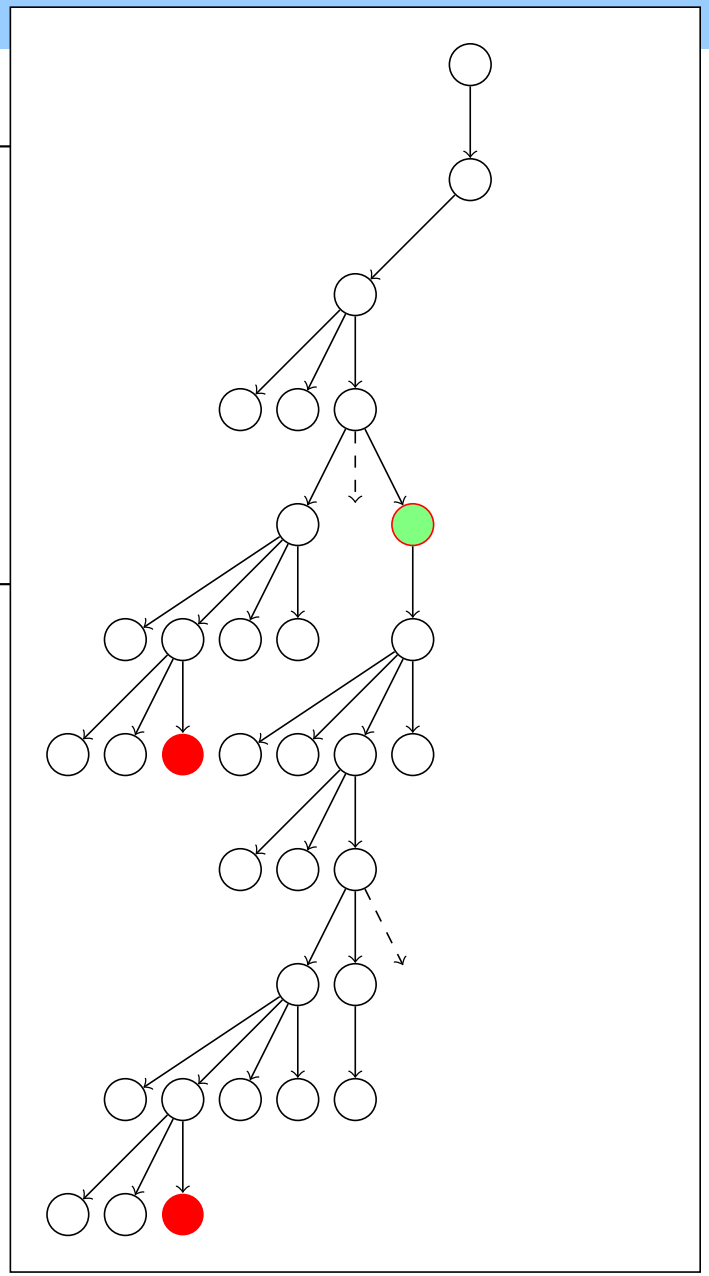
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
def geometric(p):
    if bernoulli(p) == 1: return 1
    else: return 1 + geometric(p)
```

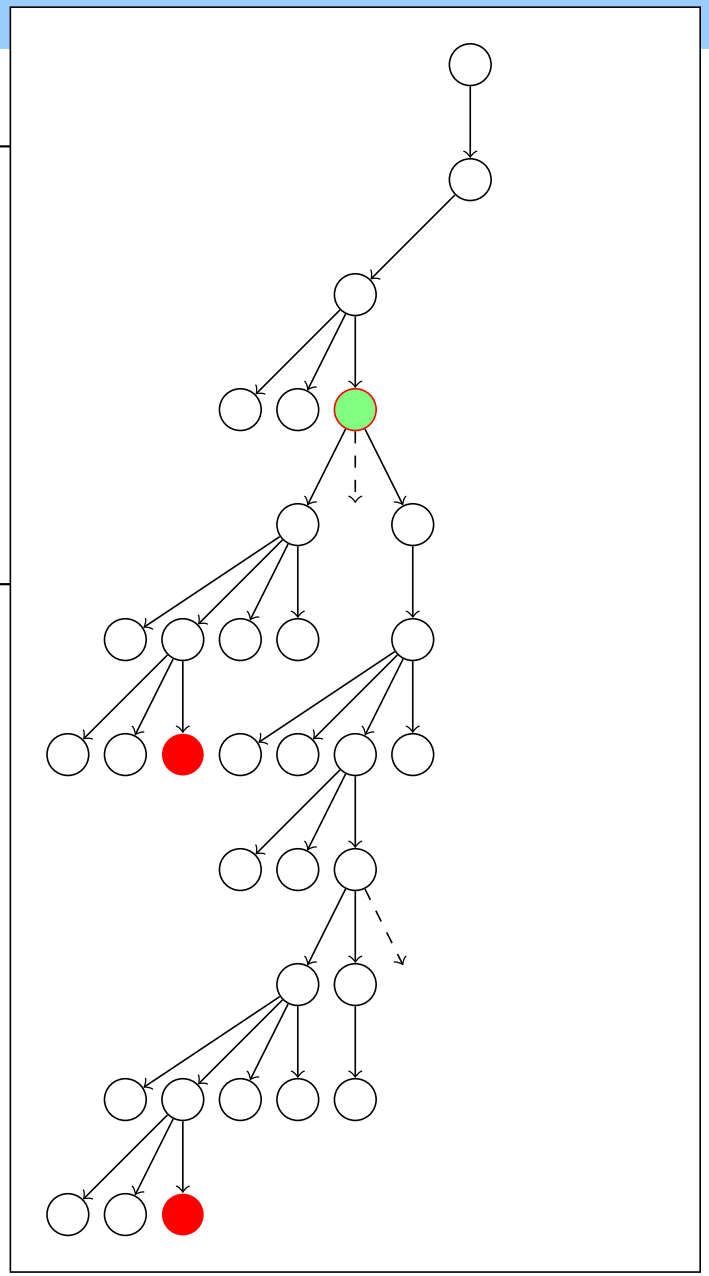
```
def aliased_geometric(p):
    g = geometric(p)
    return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

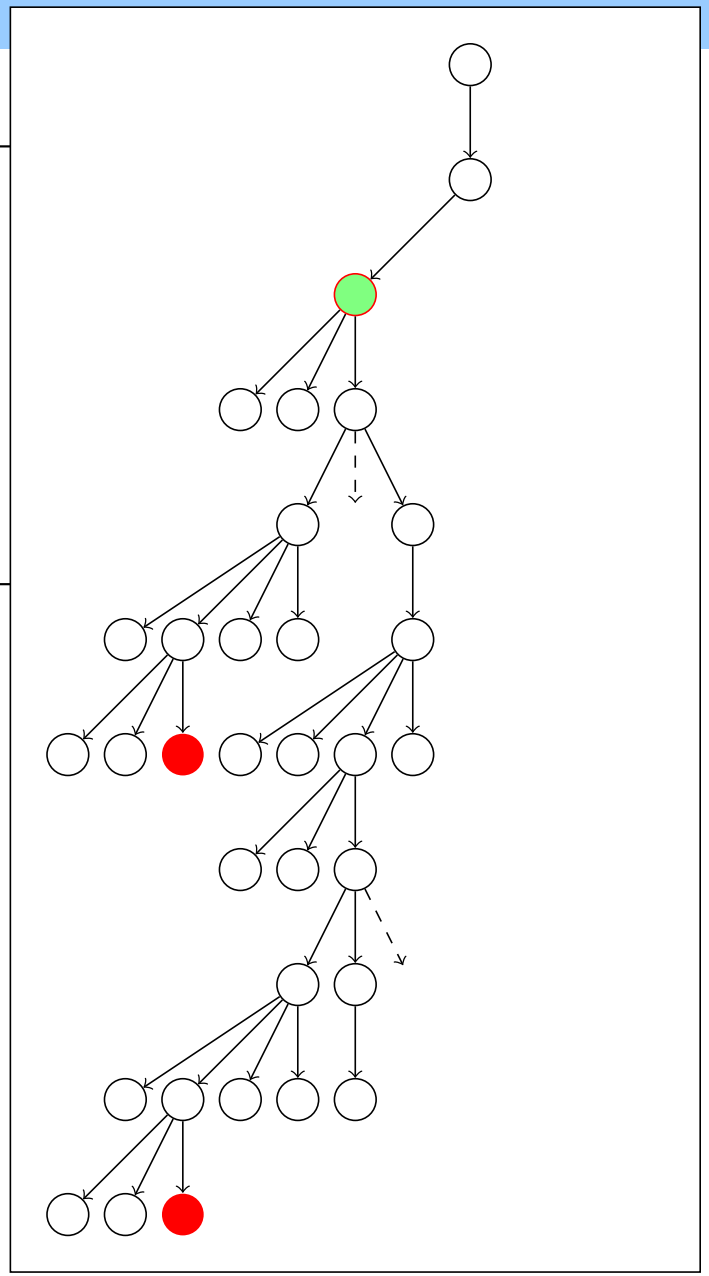
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

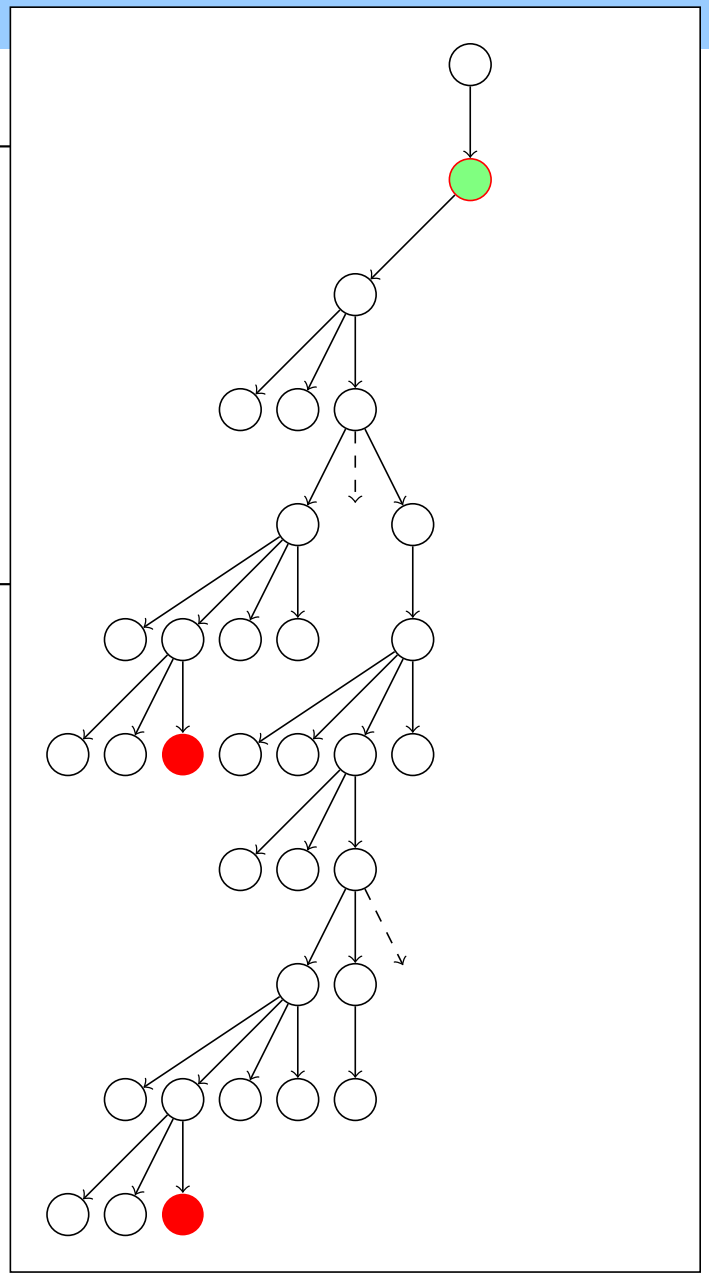
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

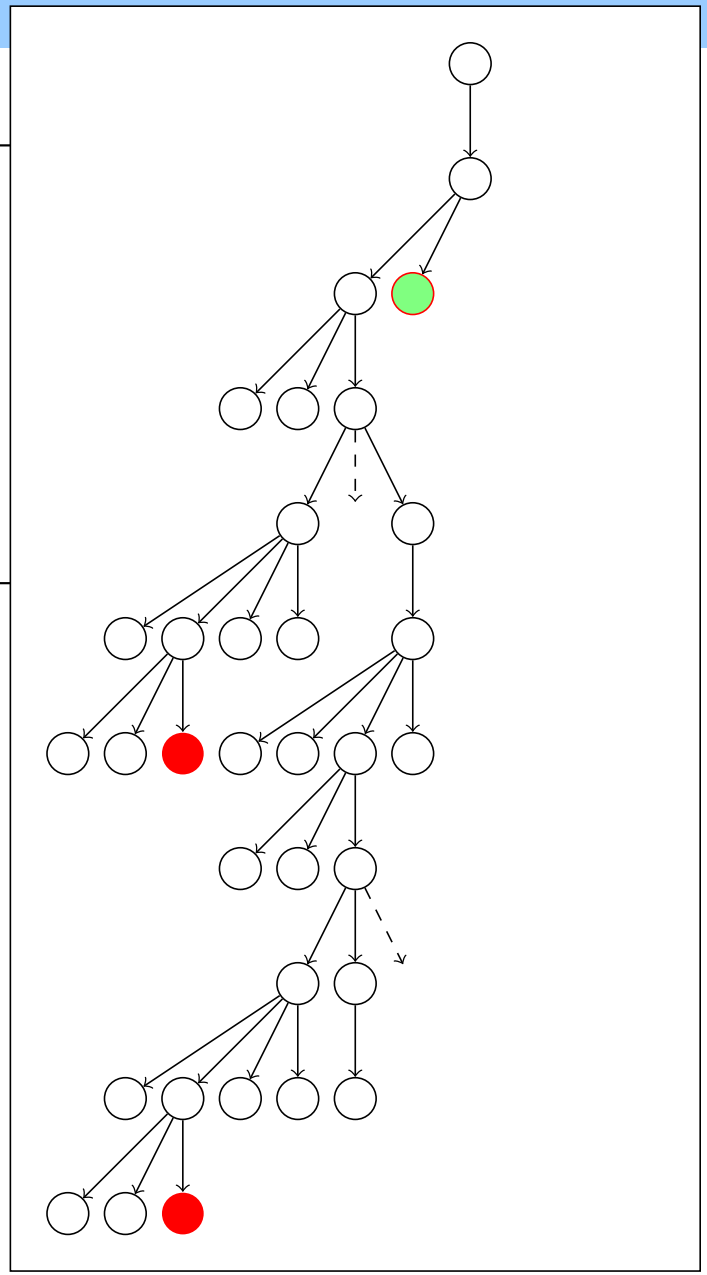
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

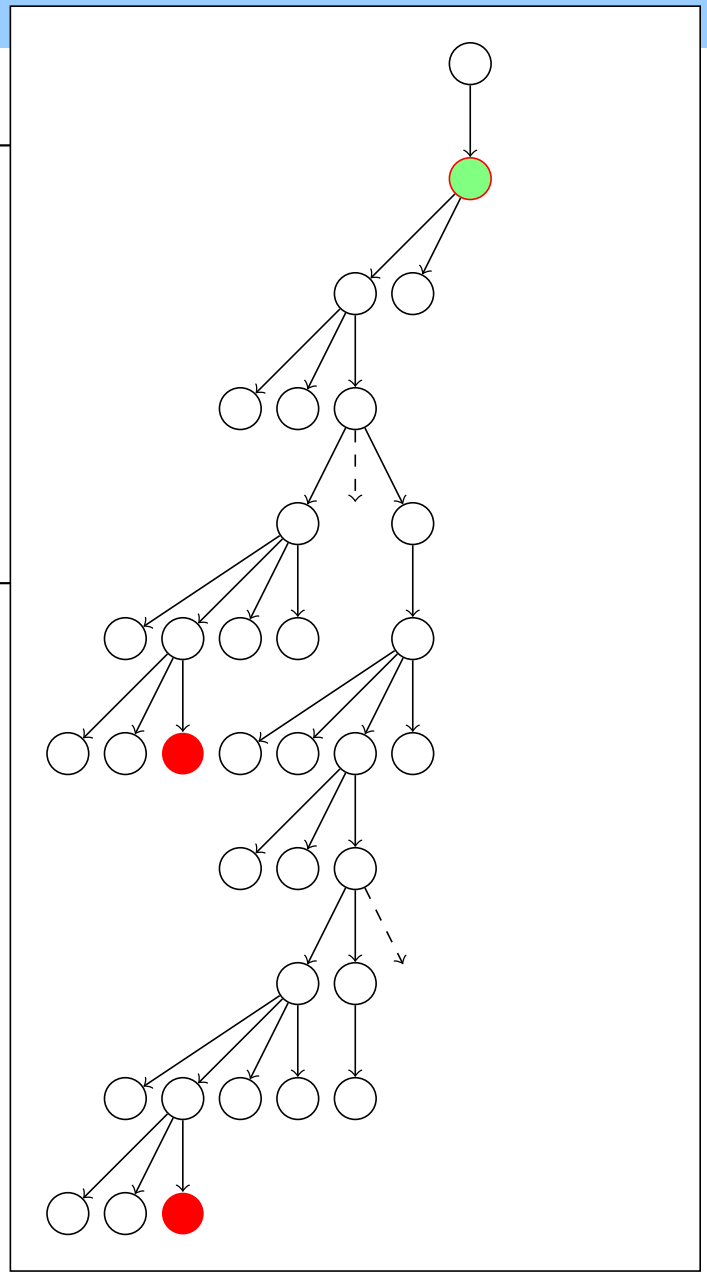
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

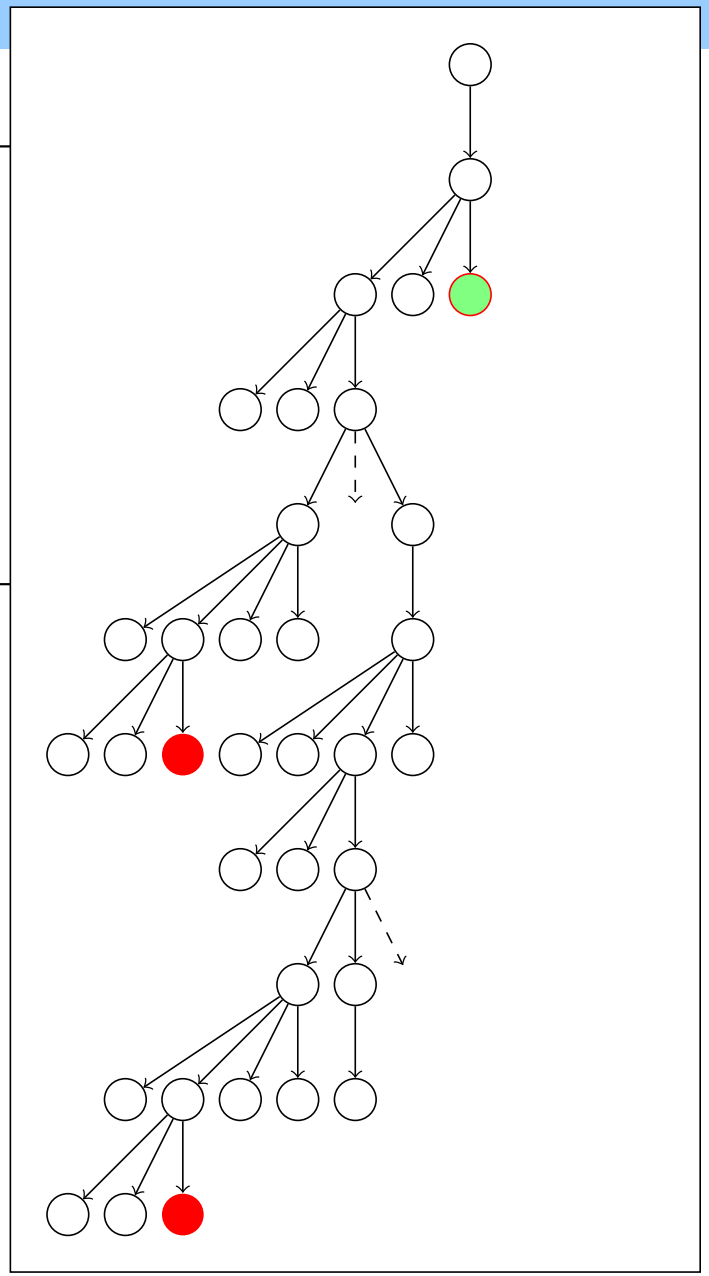
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

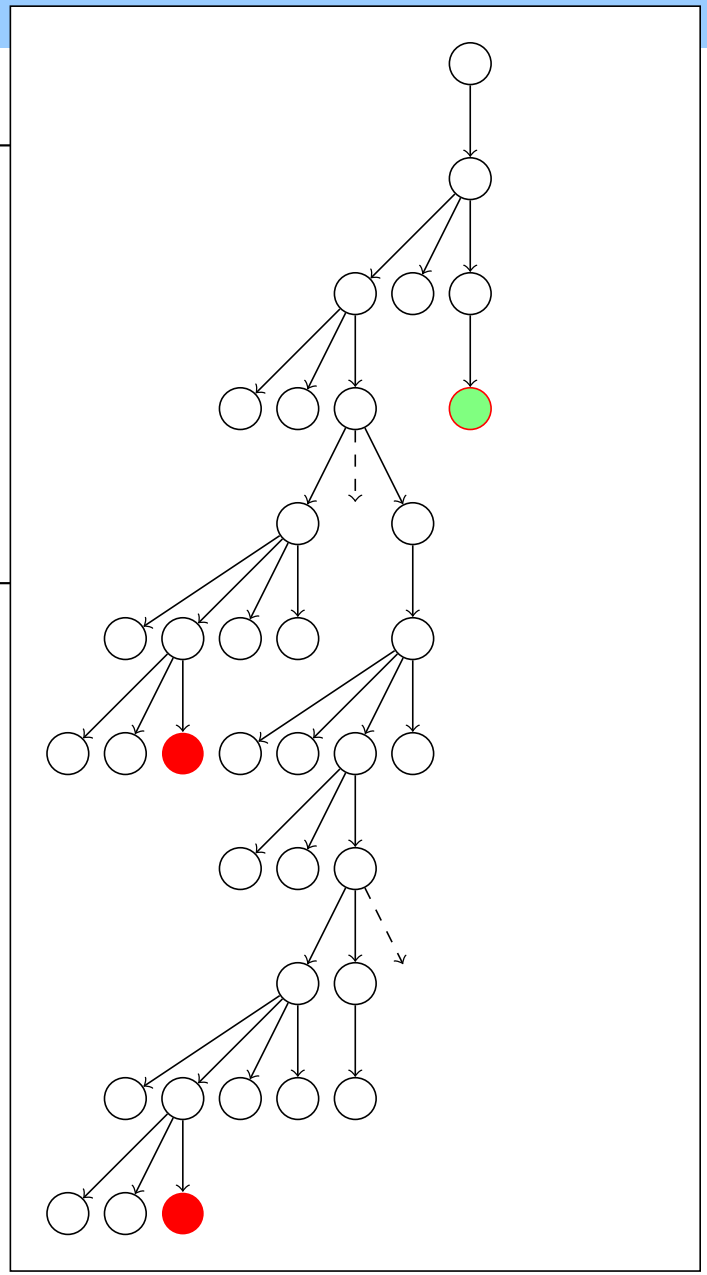
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

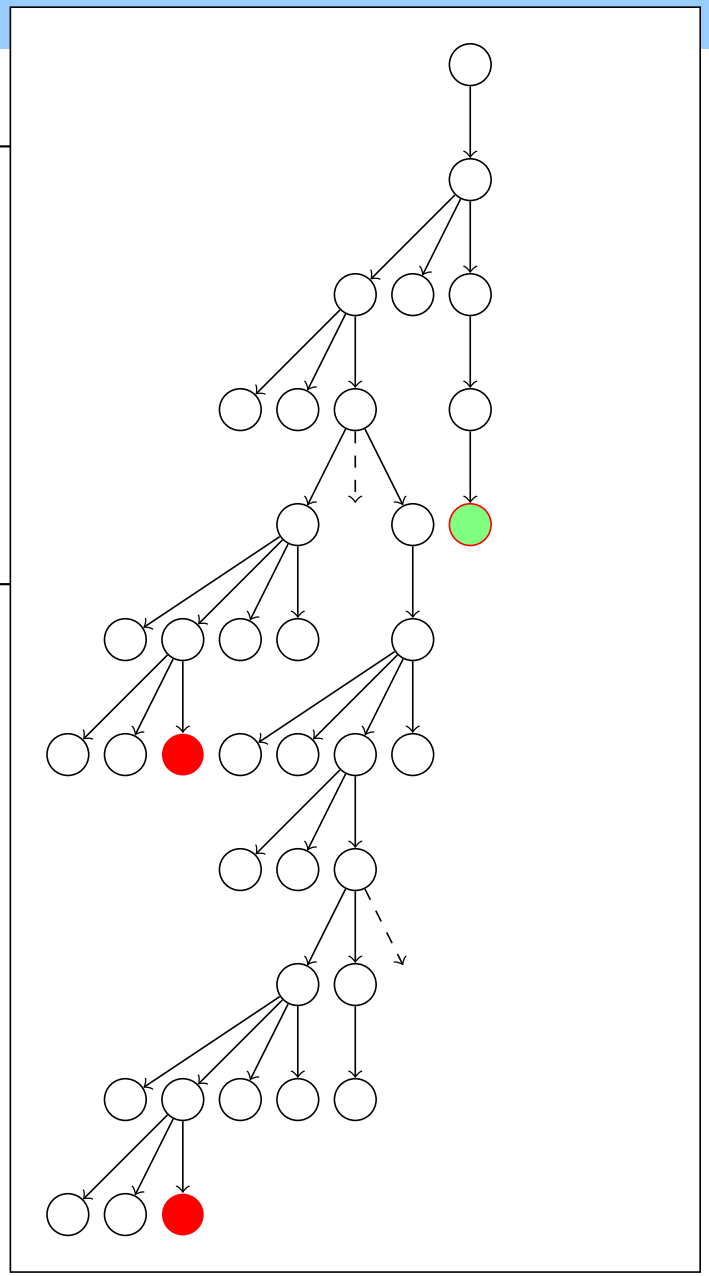
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
def geometric(p):
    if bernoulli(p) == 1: return 1
    else: return 1 + geometric(p)
```

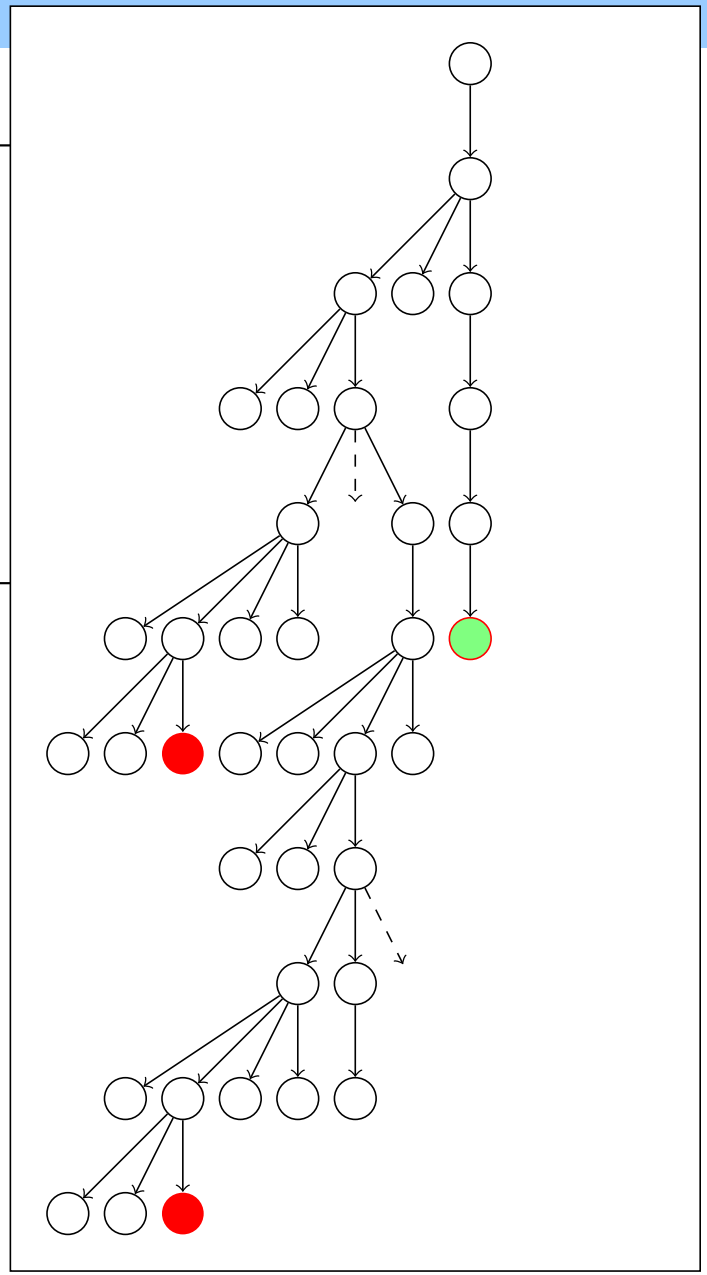
```
def aliased_geometric(p):
    g = geometric(p)
    return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

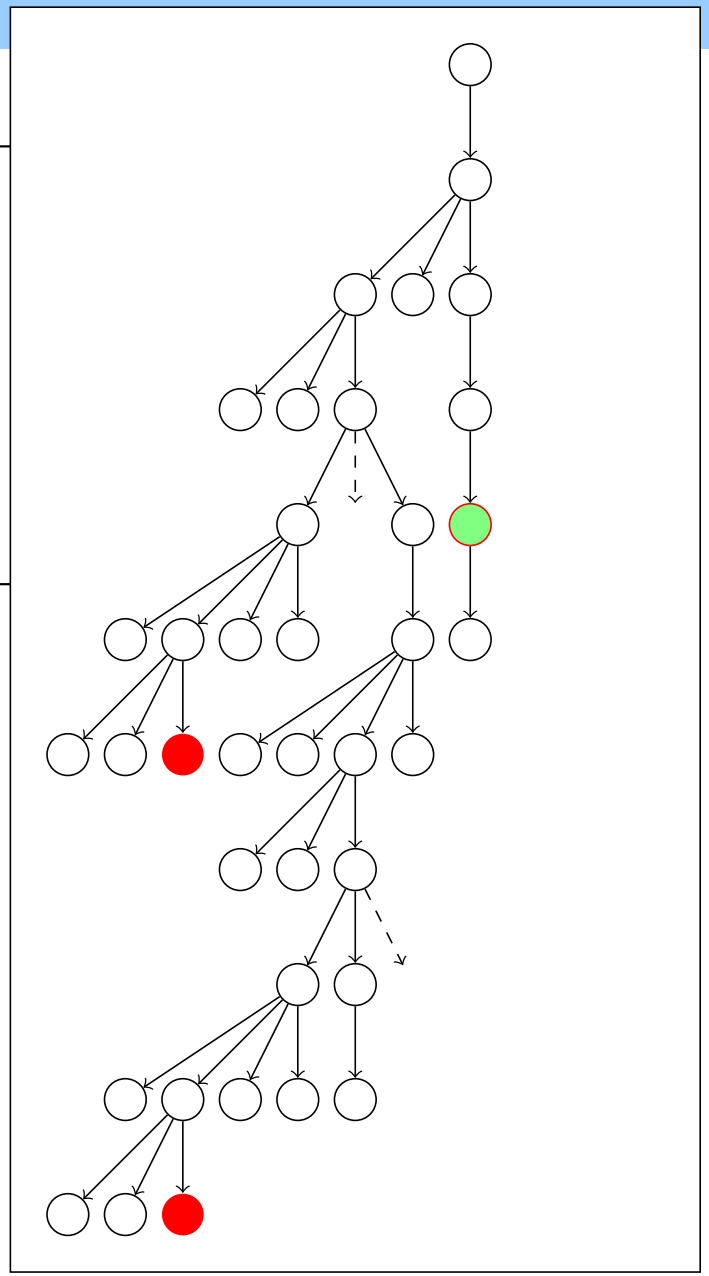
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
def geometric(p):
    if bernoulli(p) == 1: return 1
    else: return 1 + geometric(p)
```

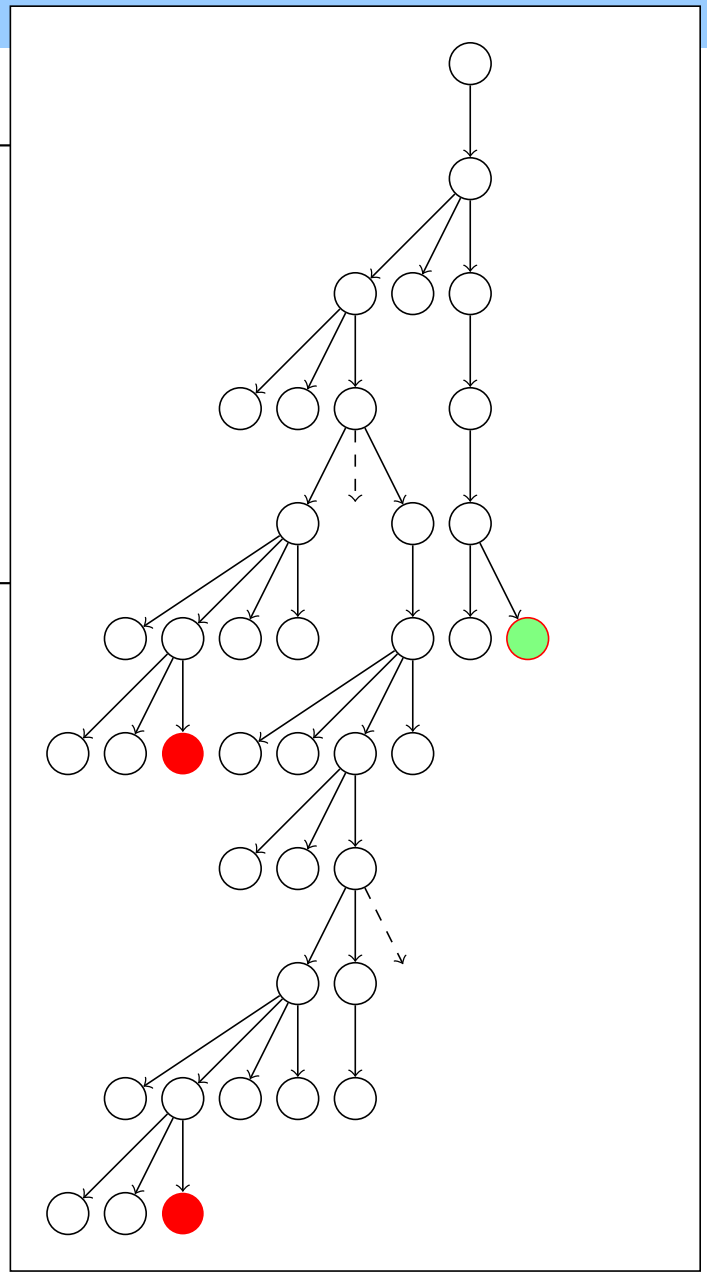
```
def aliased_geometric(p):
    g = geometric(p)
    return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

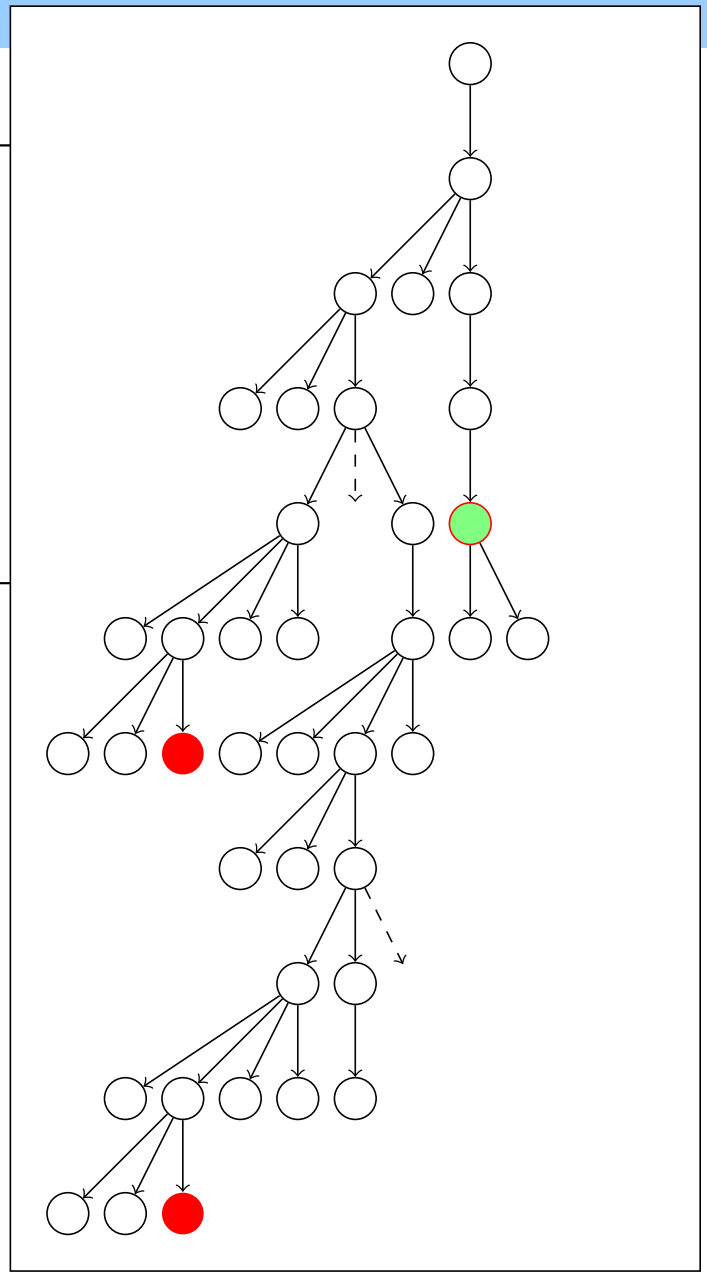
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

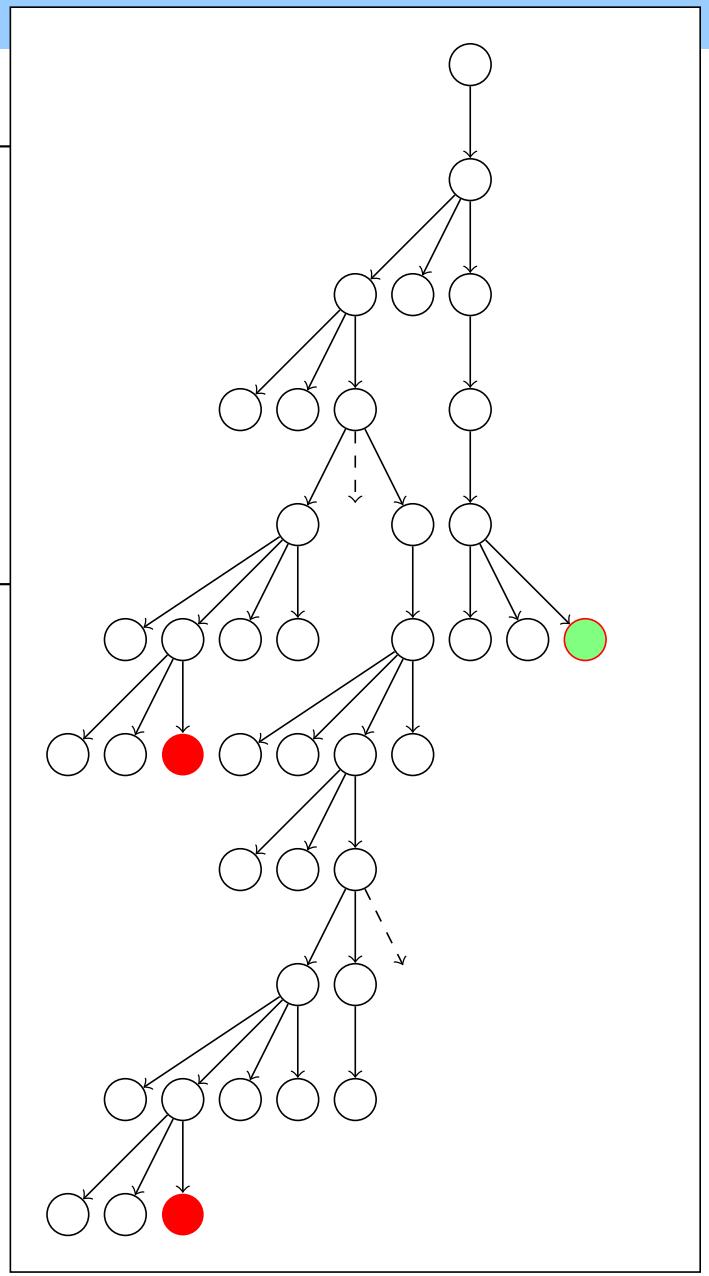
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

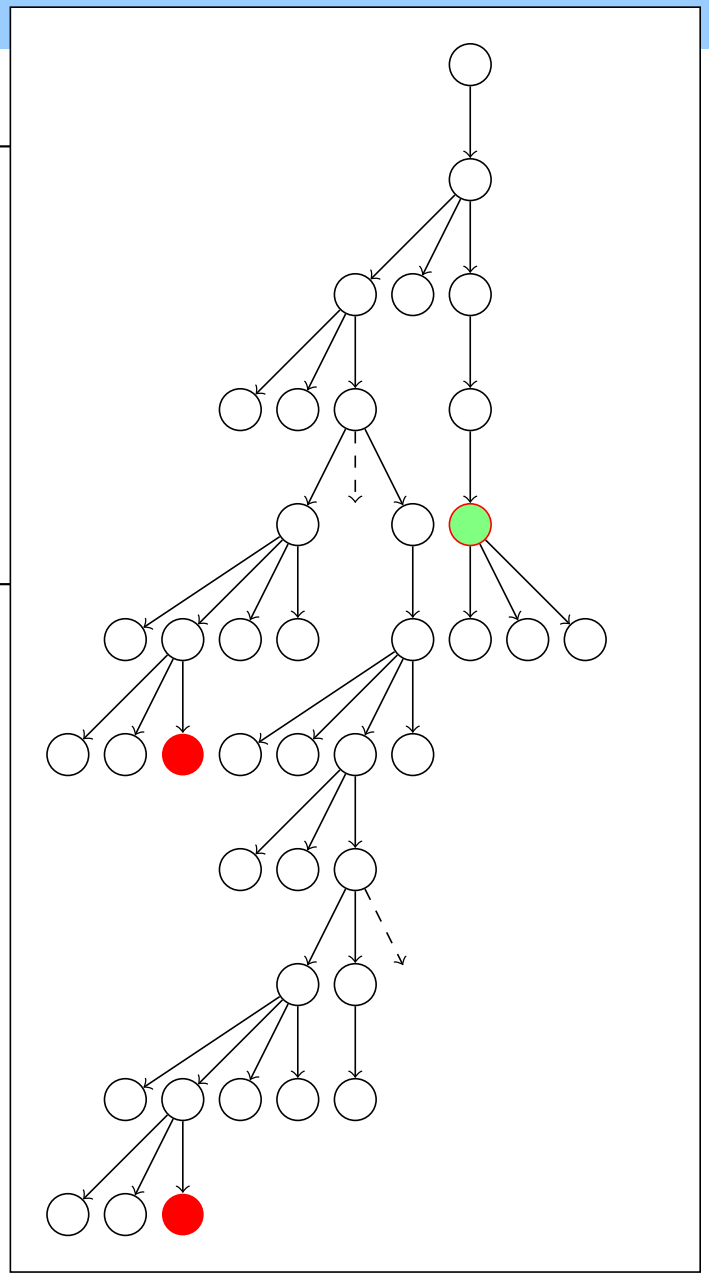
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
def geometric(p):
    if bernoulli(p) == 1: return 1
    else: return 1 + geometric(p)
```

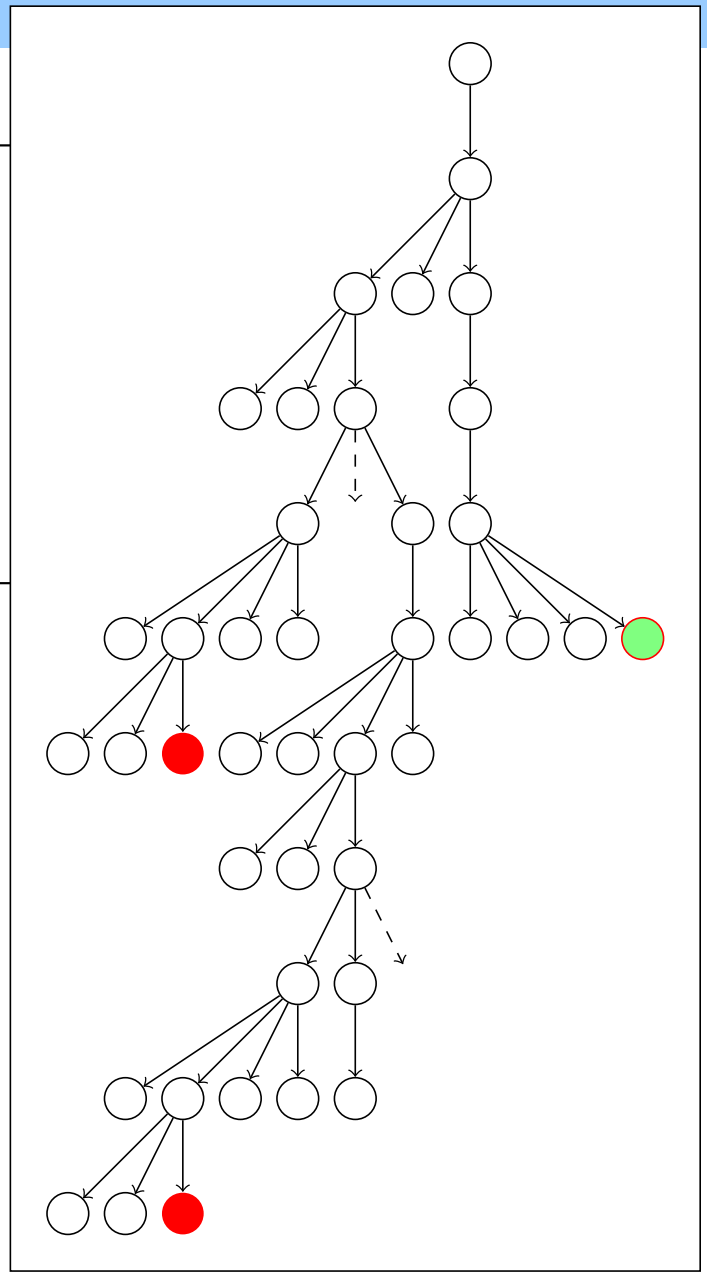
```
def aliased_geometric(p):
    g = geometric(p)
    return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

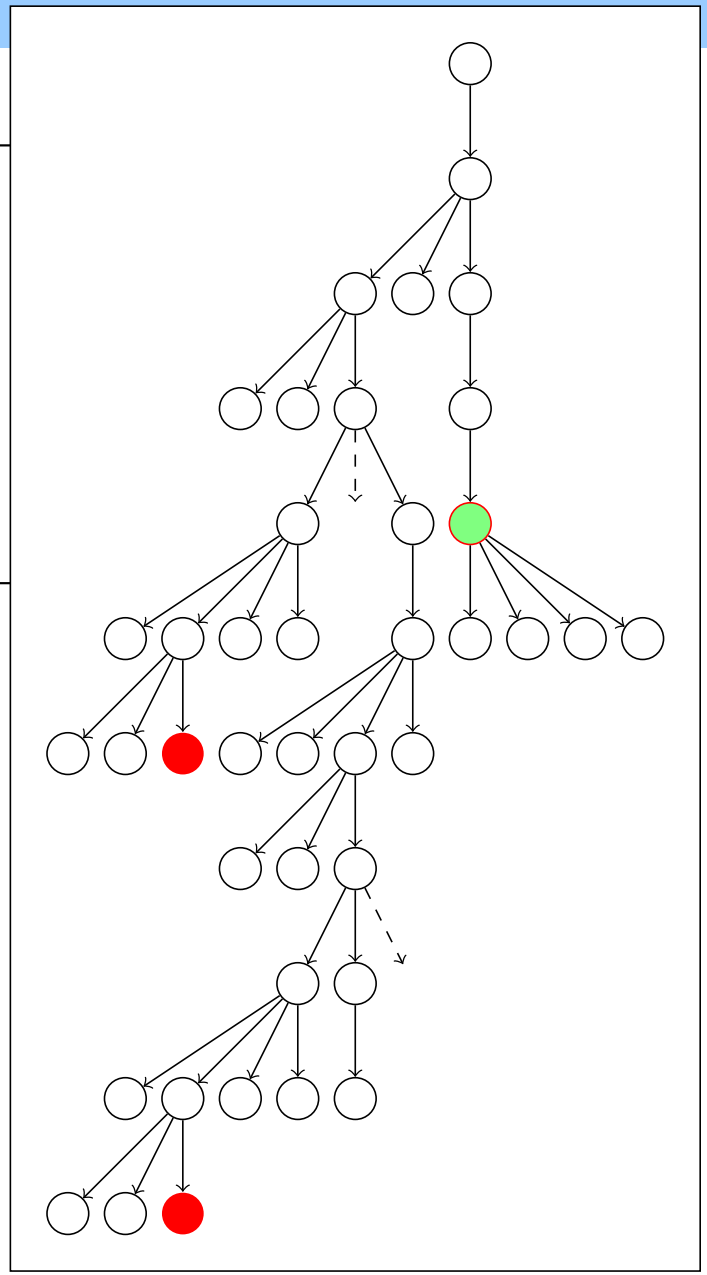
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

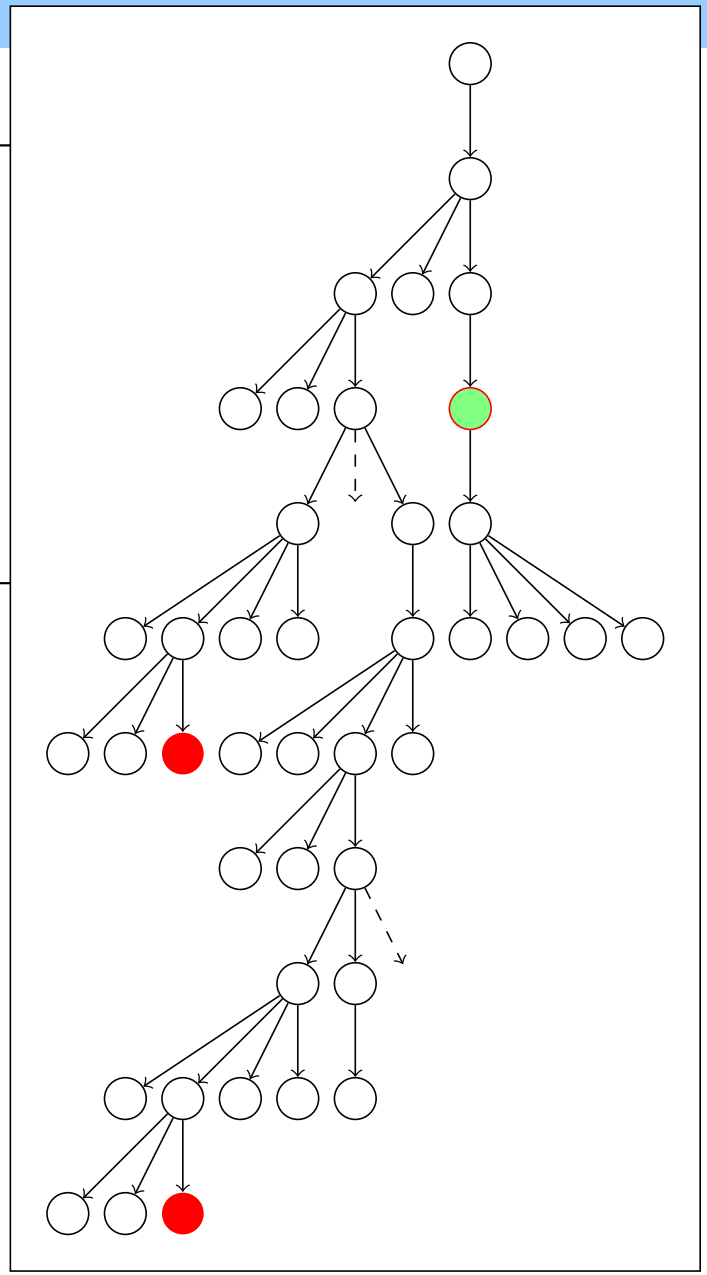
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

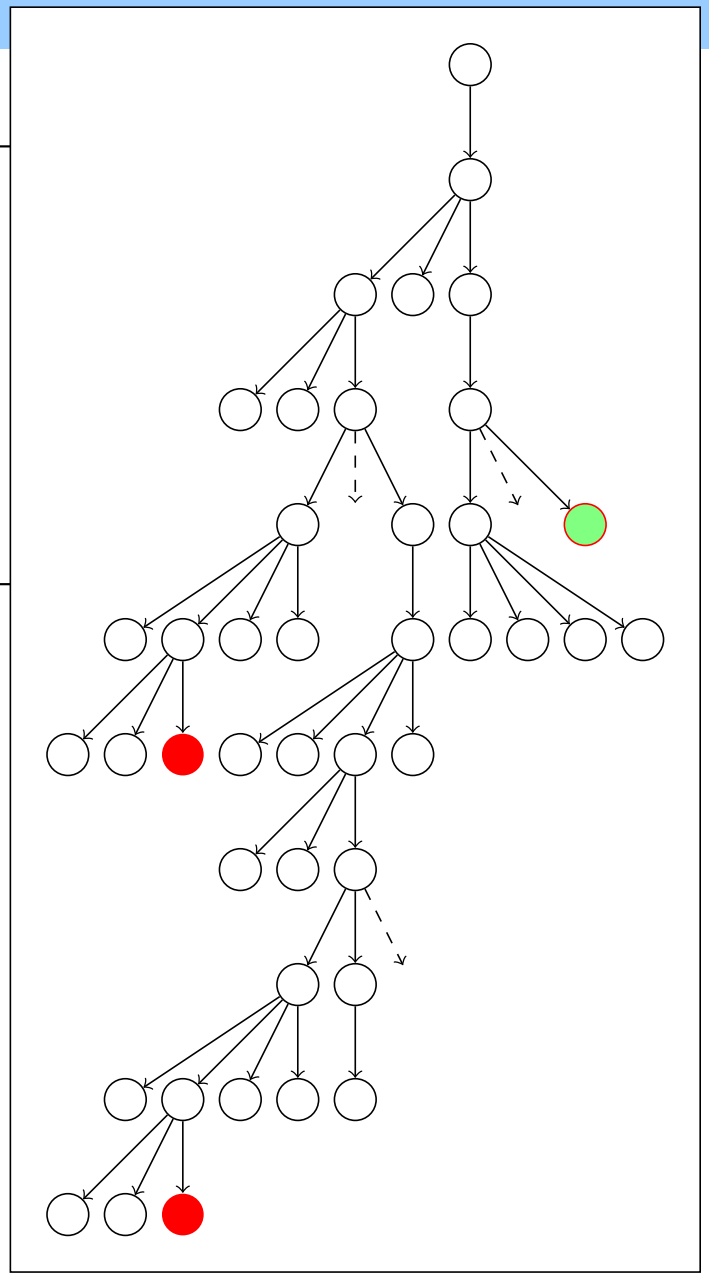
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

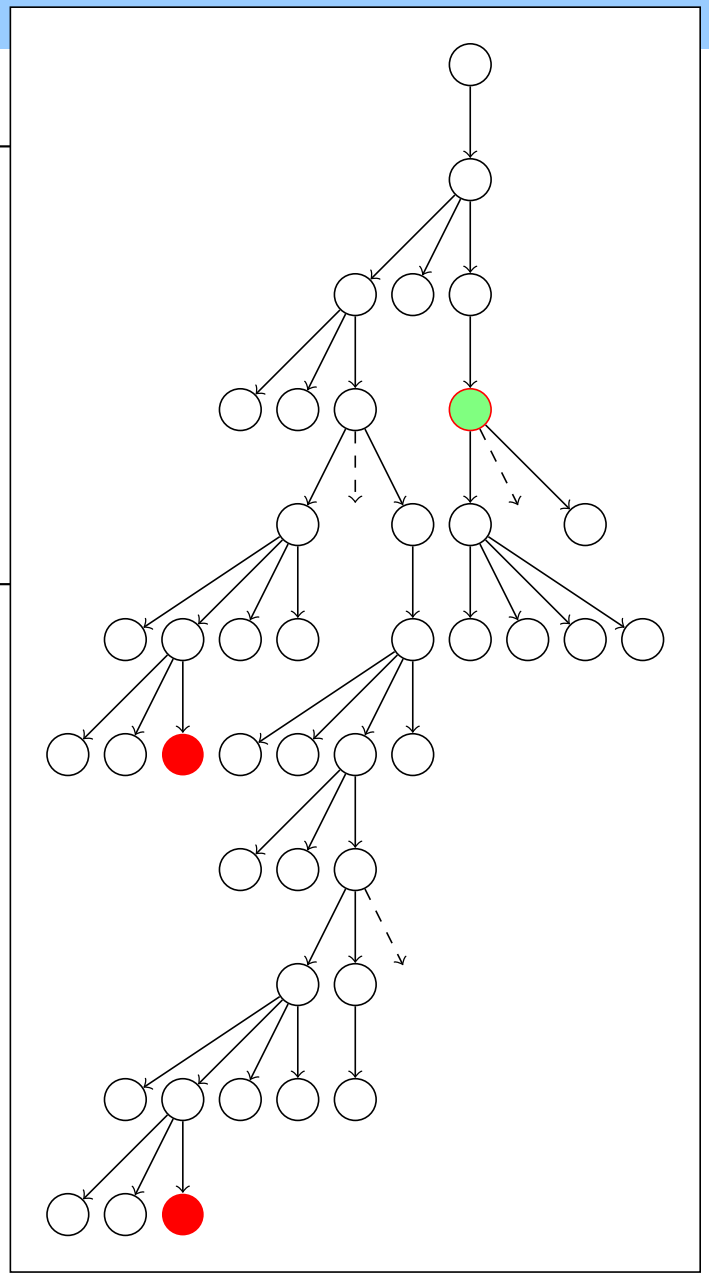
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

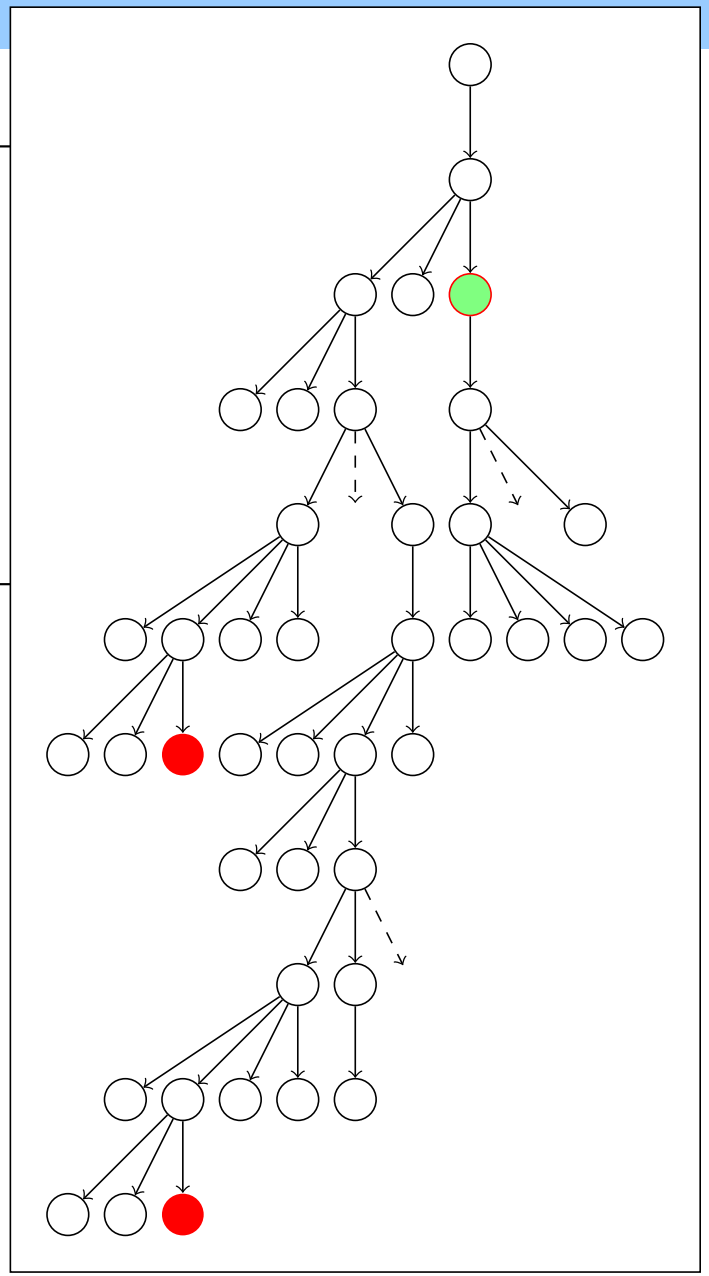
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

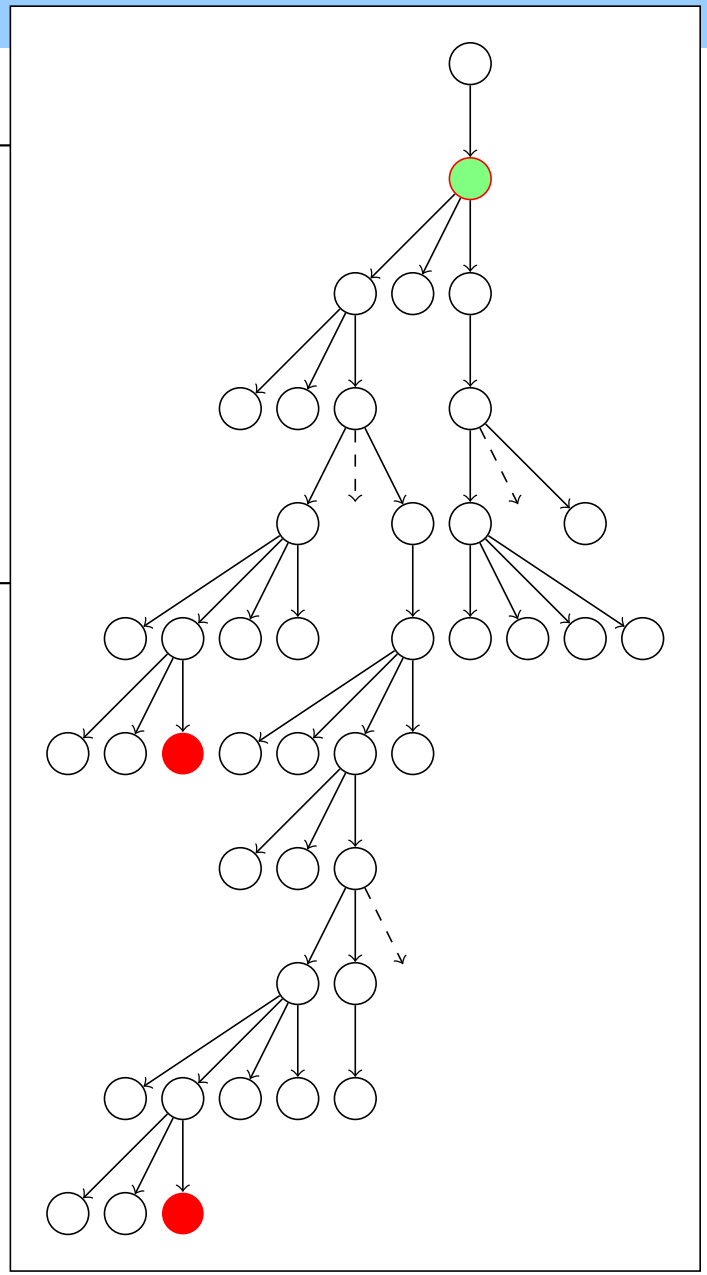
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

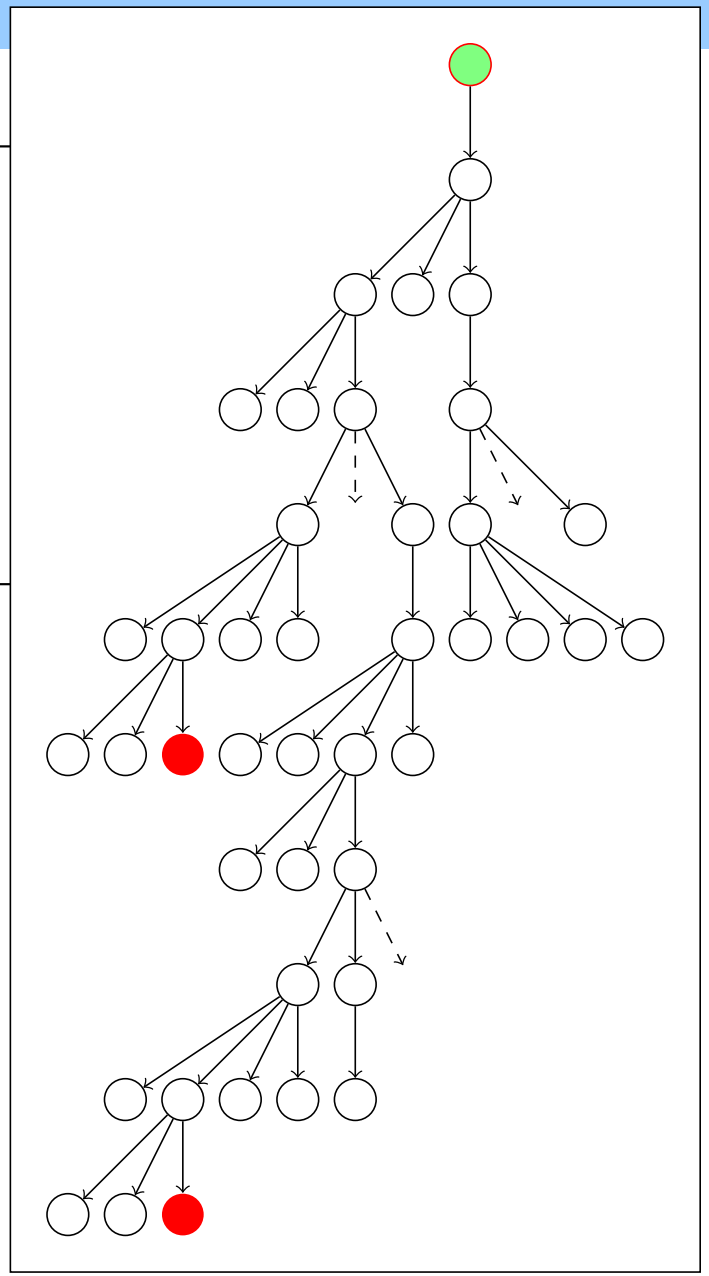
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

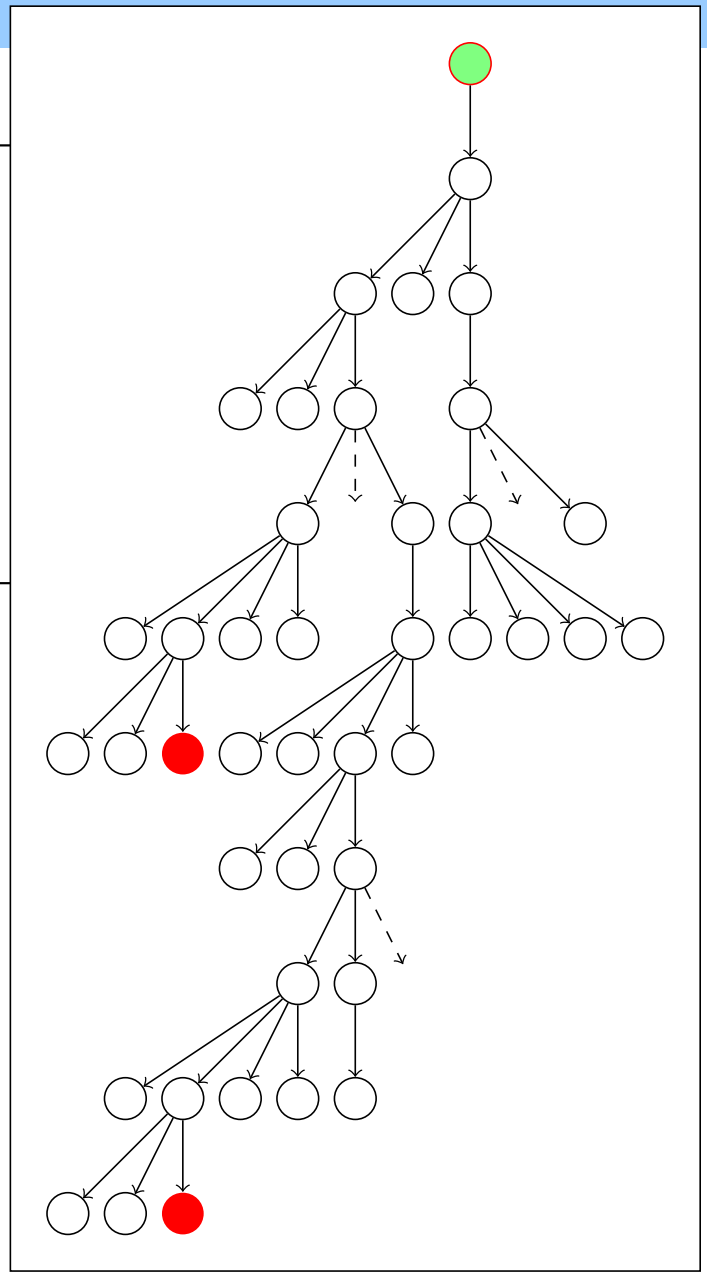
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

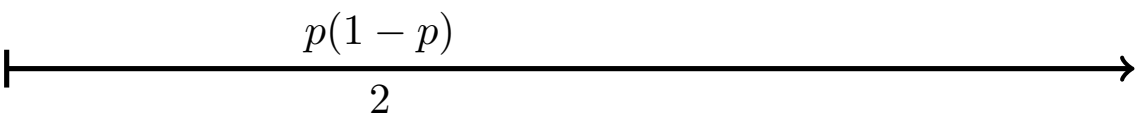
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```

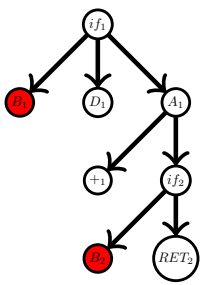


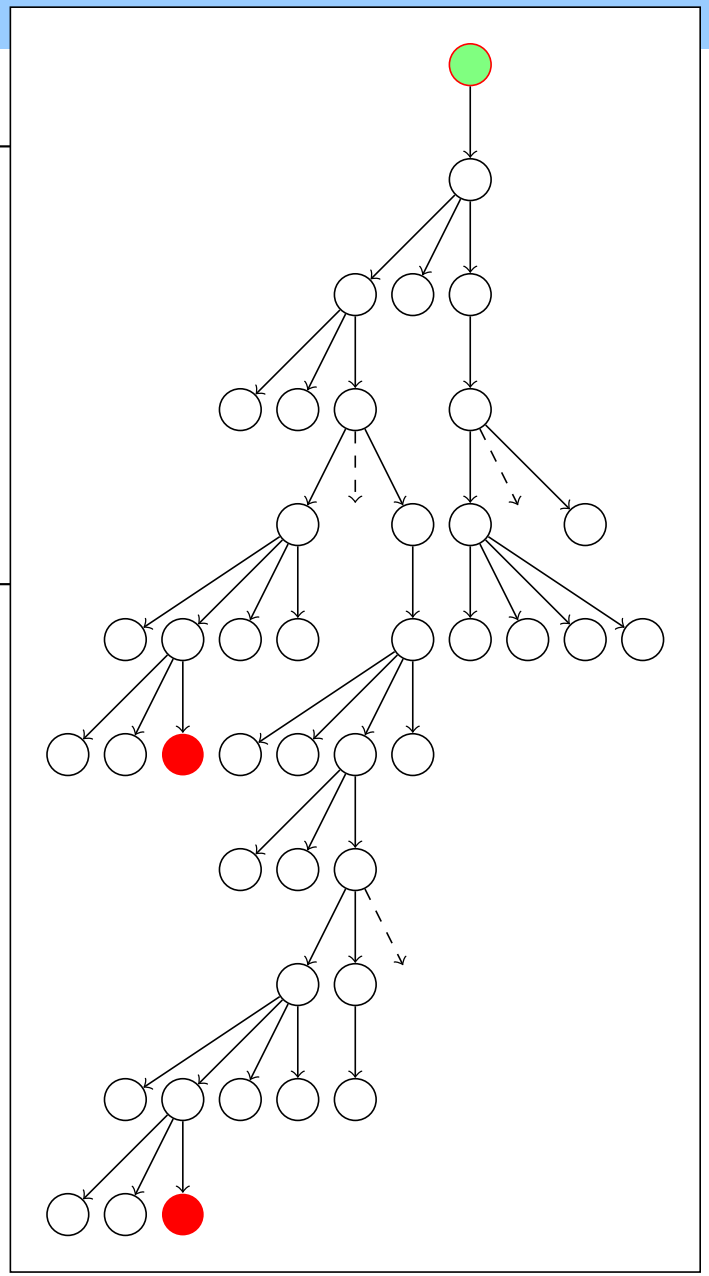
MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



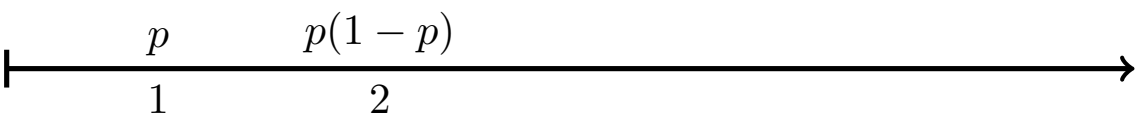


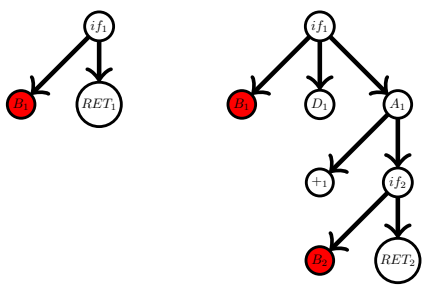


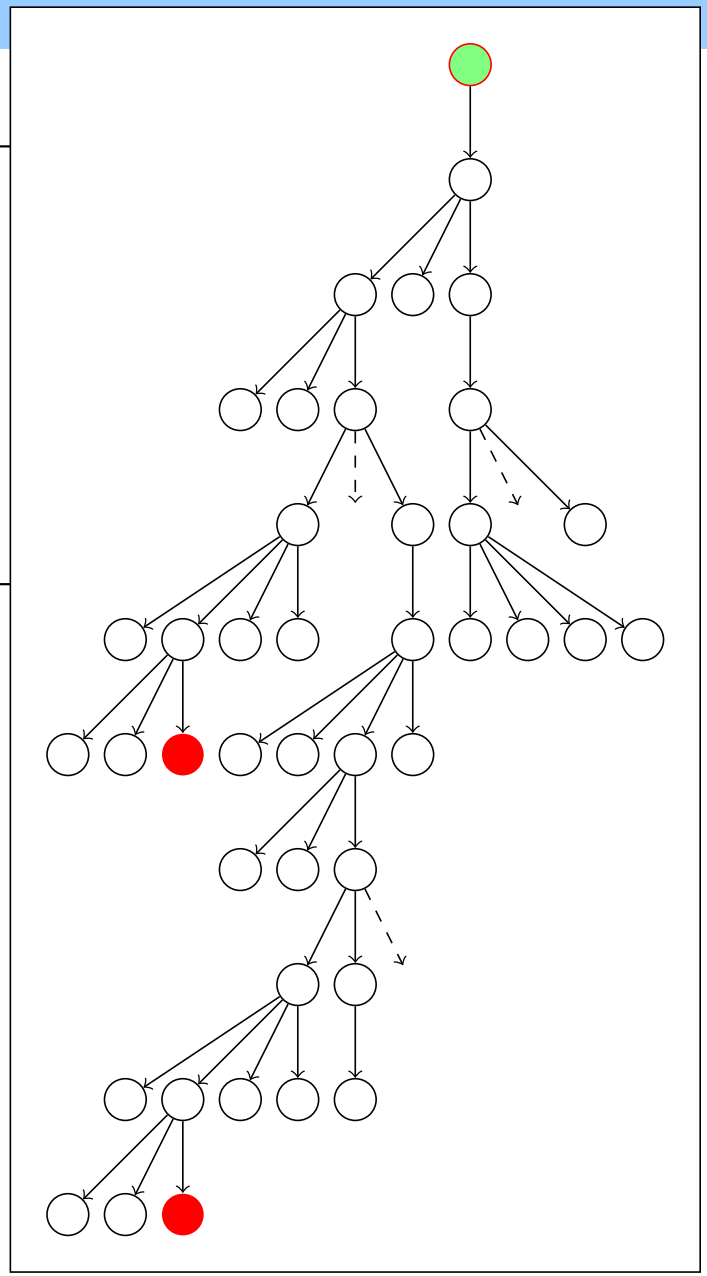
MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```



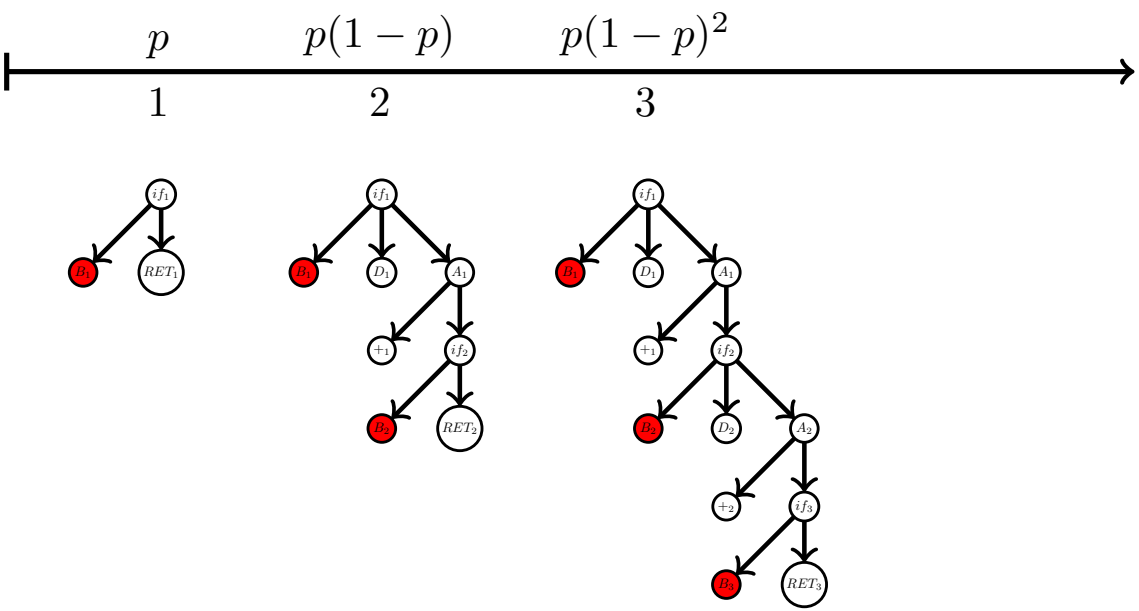


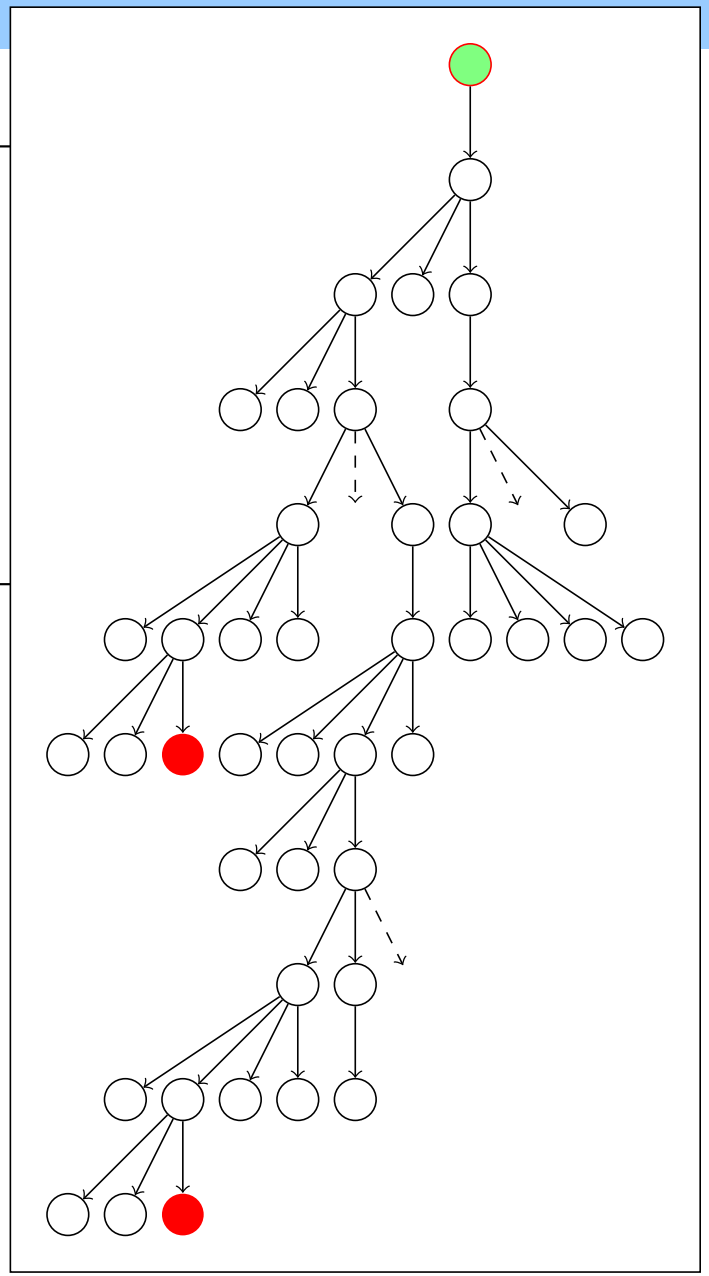


MIT-Church aka Trace-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```

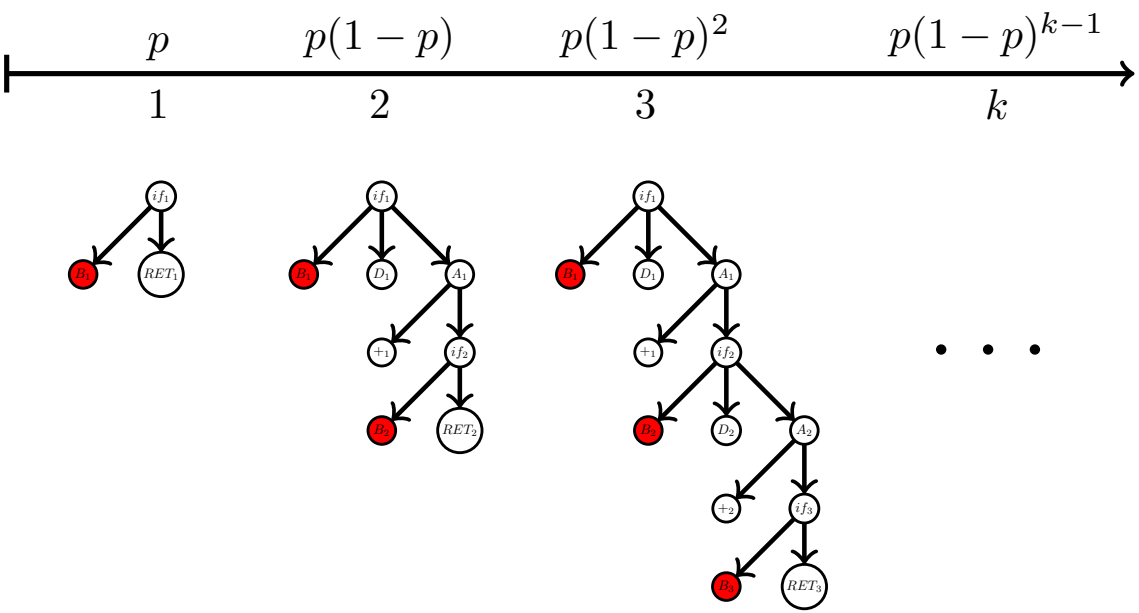


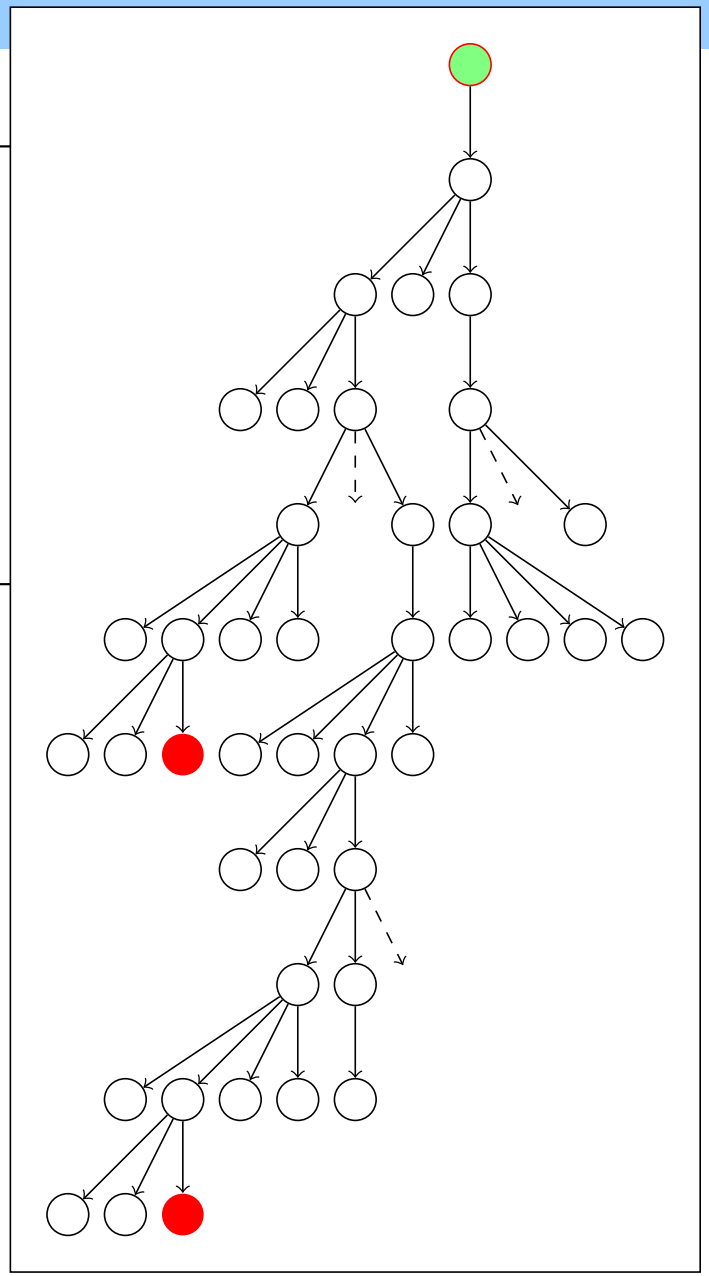


MIT-CHURCH AKA TRACE-MCMC

```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```

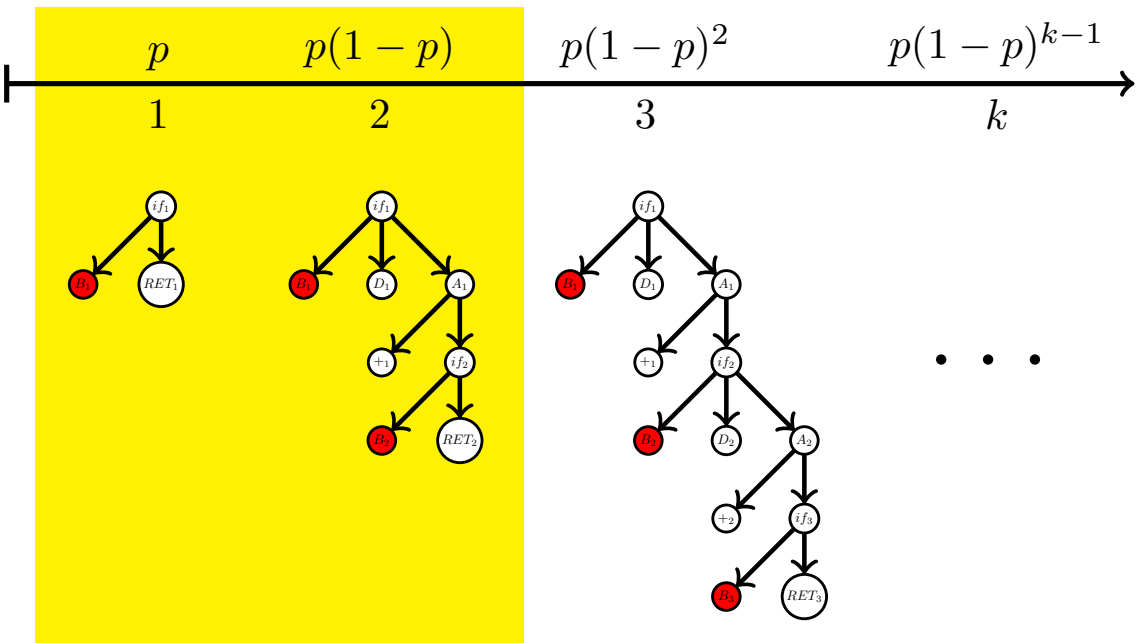


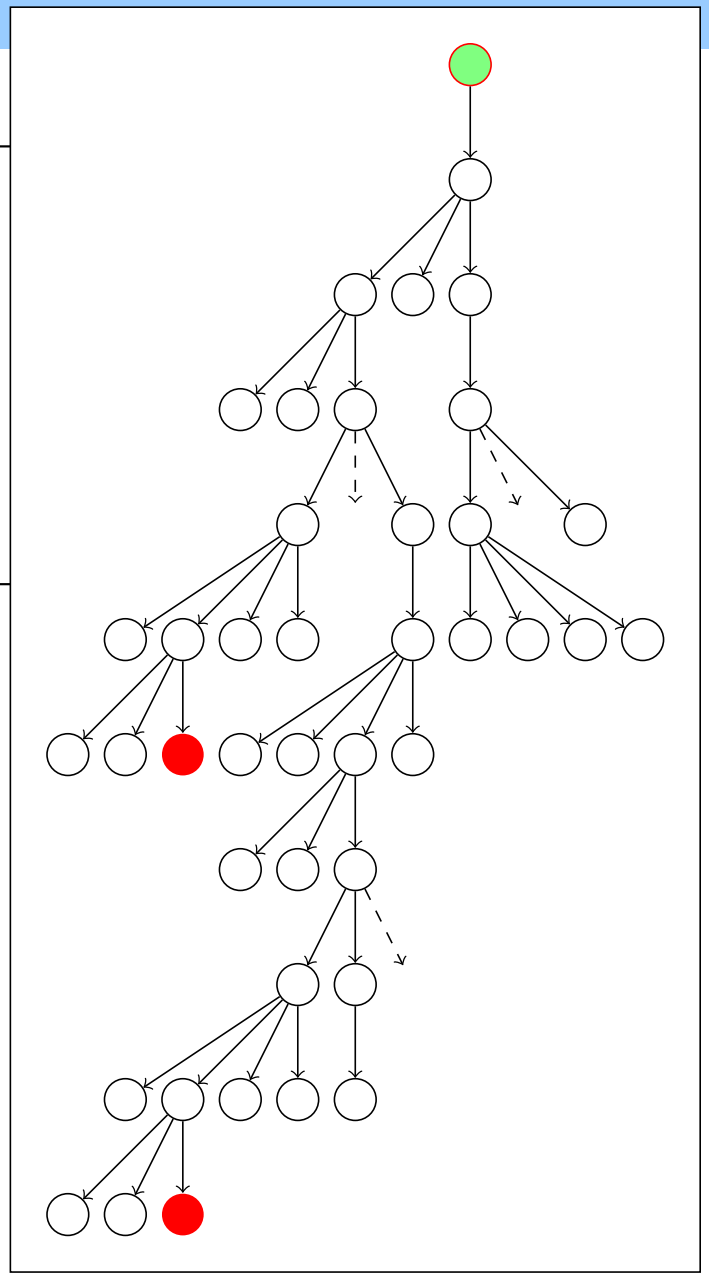


MIT-Church aka Trace-MCMC

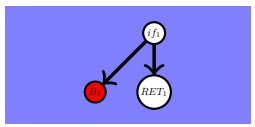
```
1  def geometric(p):
2      if bernoulli(p) == 1: return 1
3      else: return 1 + geometric(p)
```

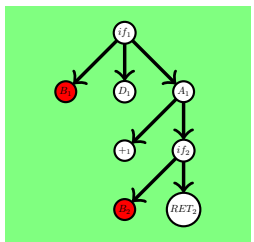
```
1  def aliased_geometric(p):
2      g = geometric(p)
3      return 1 if g < 3 else 0
```

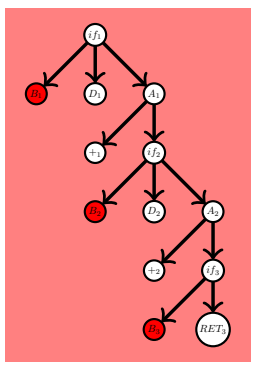




MIT-Church aka Trace-MCMC





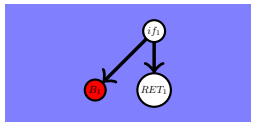
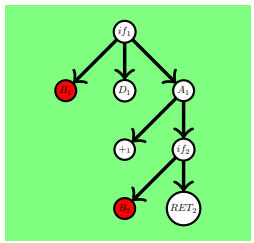
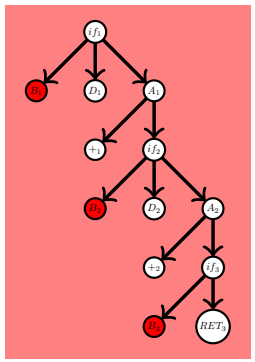


4

5

6

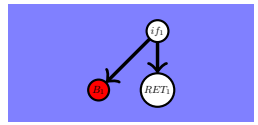
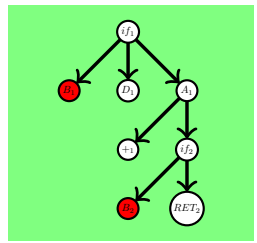
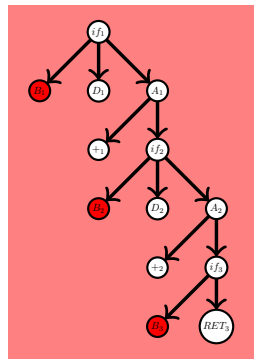
MIT-Church aka Trace-MCMC

4

5

6

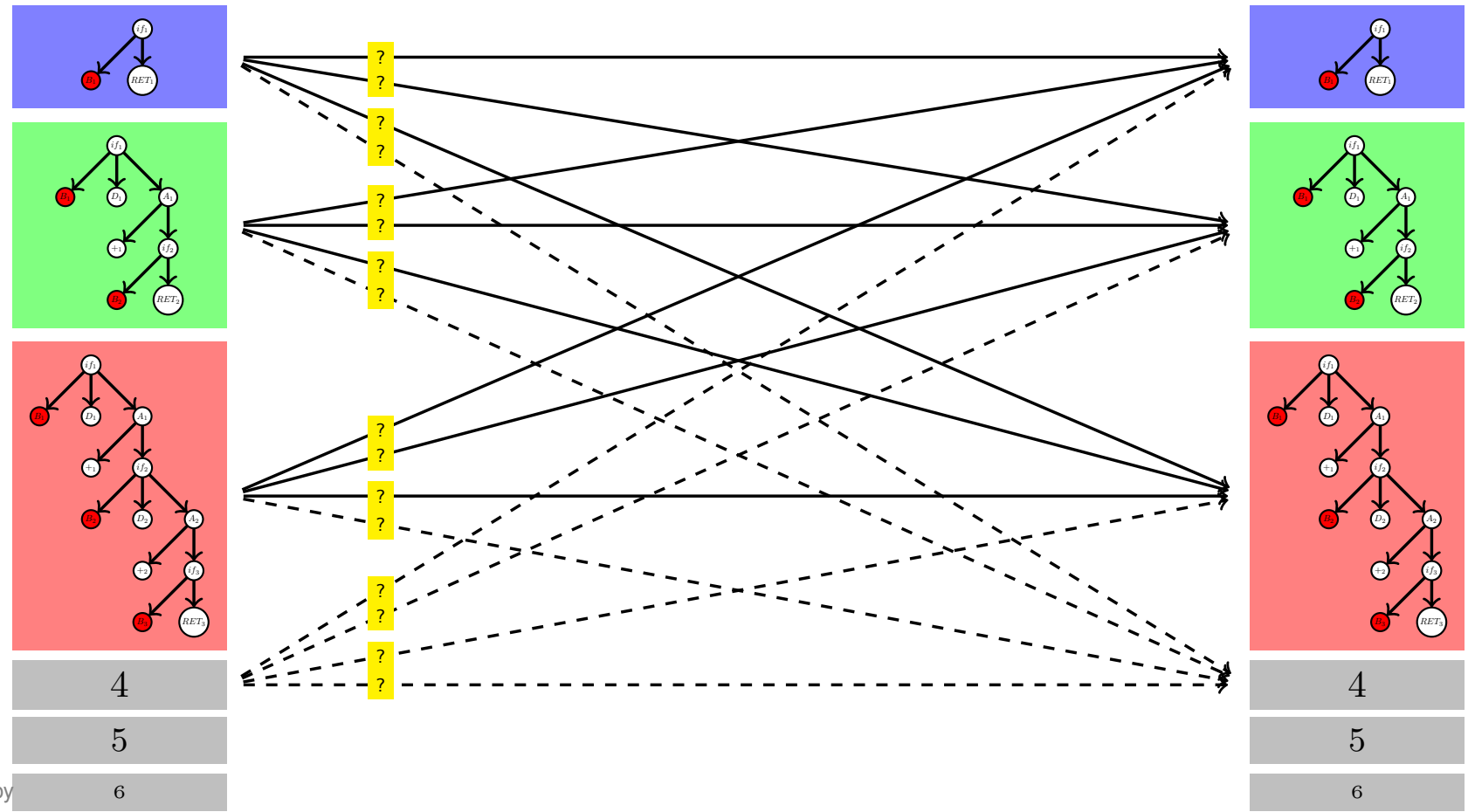




4

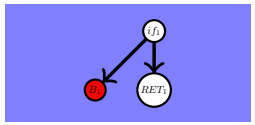
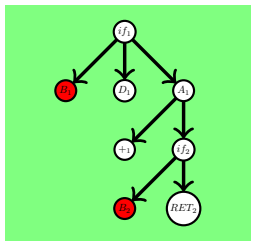
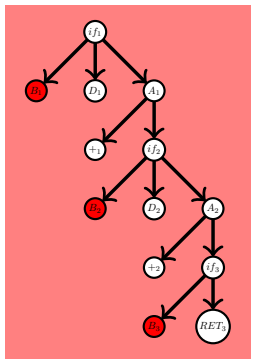
5

6

MIT-Church aka Trace-MCMC



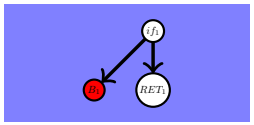
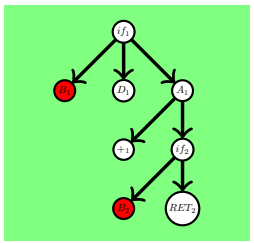
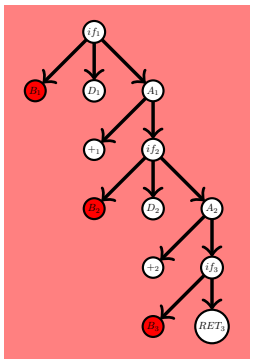
MIT-Church aka Trace-MCMC

4

5

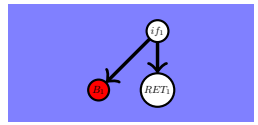
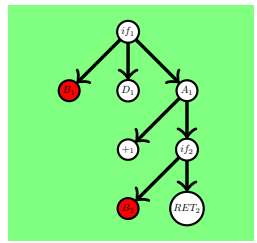
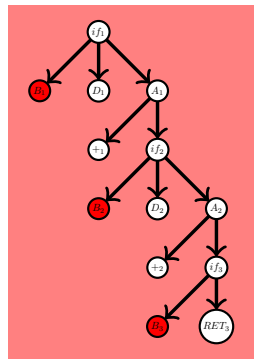
6

4

5

6

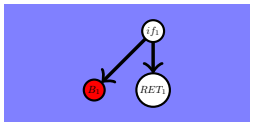
4

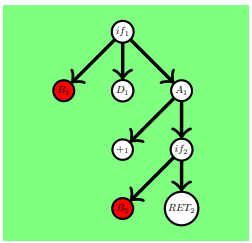
5

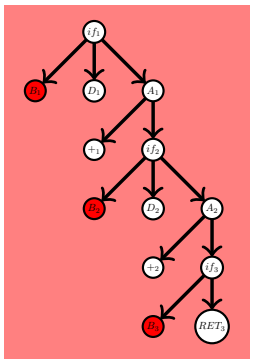
6

MIT-Church aka Trace-MCMC

Proposal



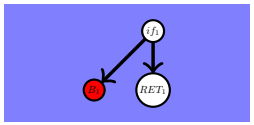


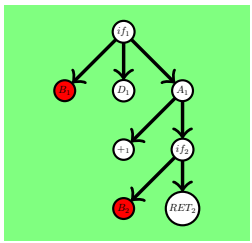


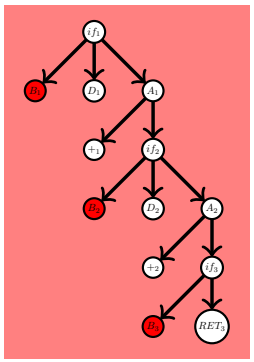
4

5

6



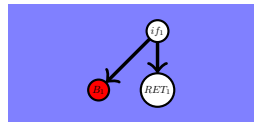


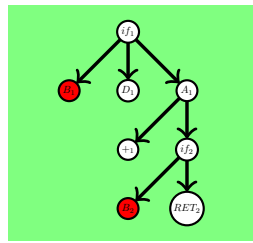


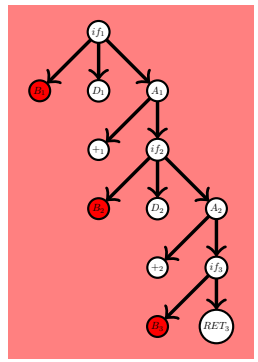
4

5

6





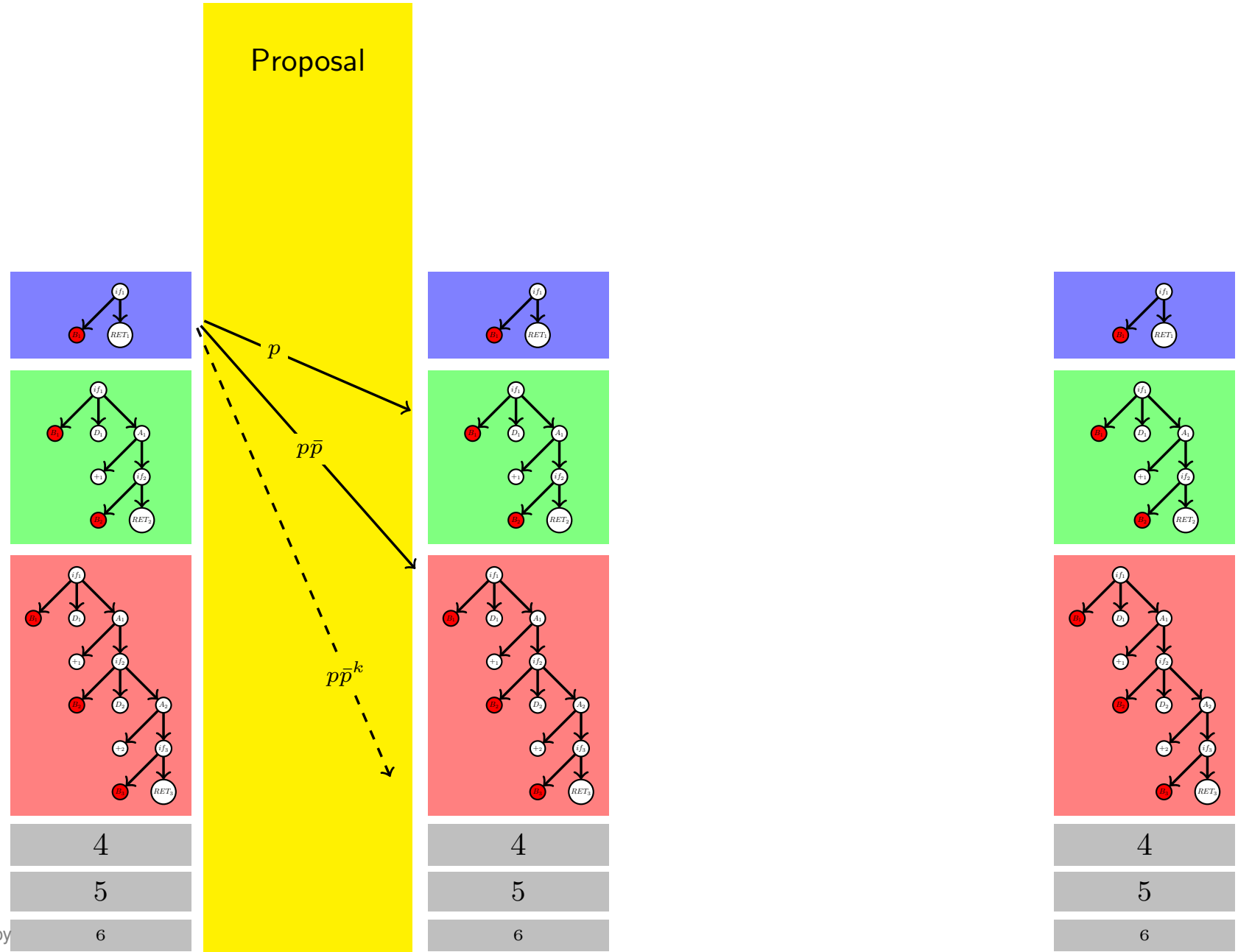


4

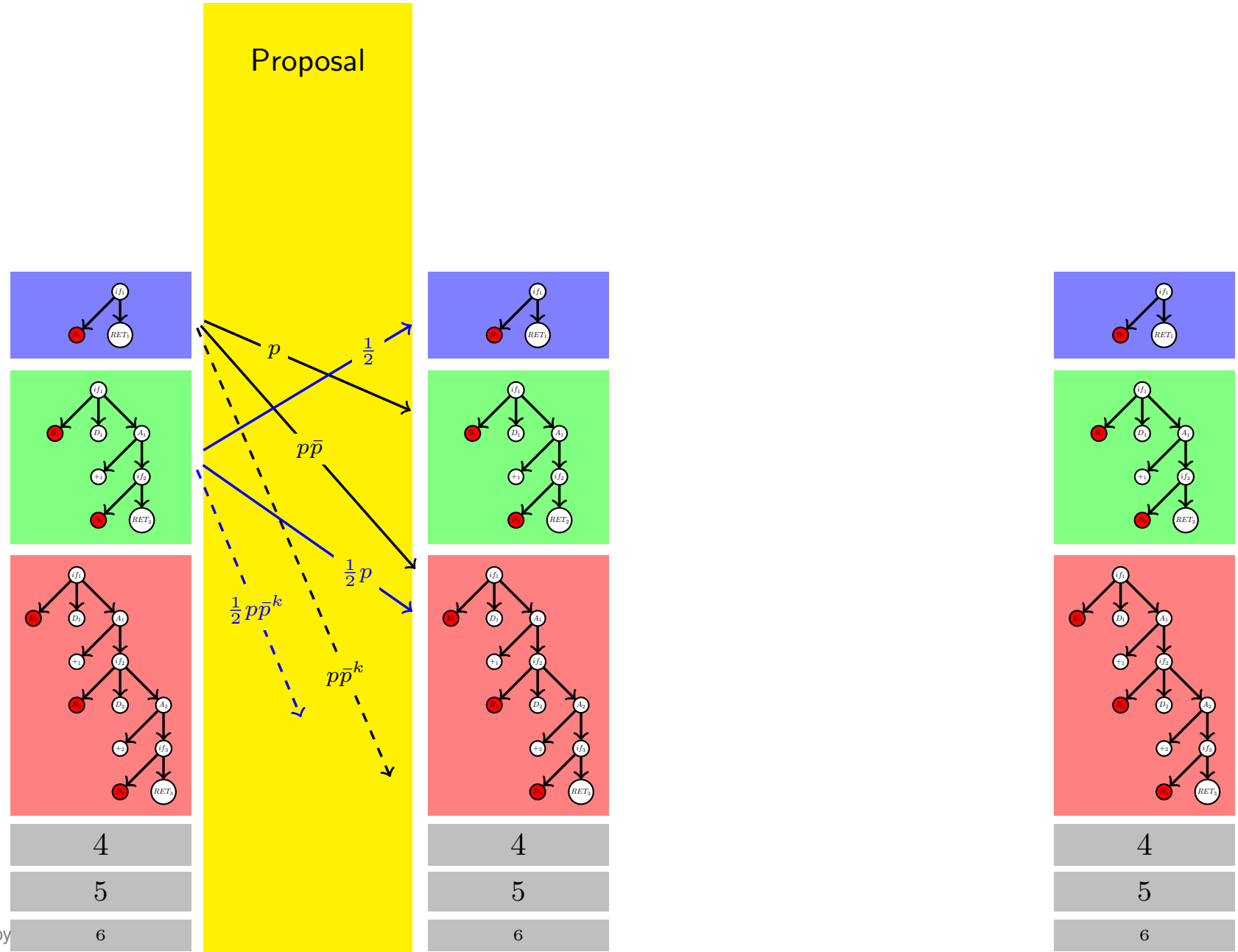
5

6

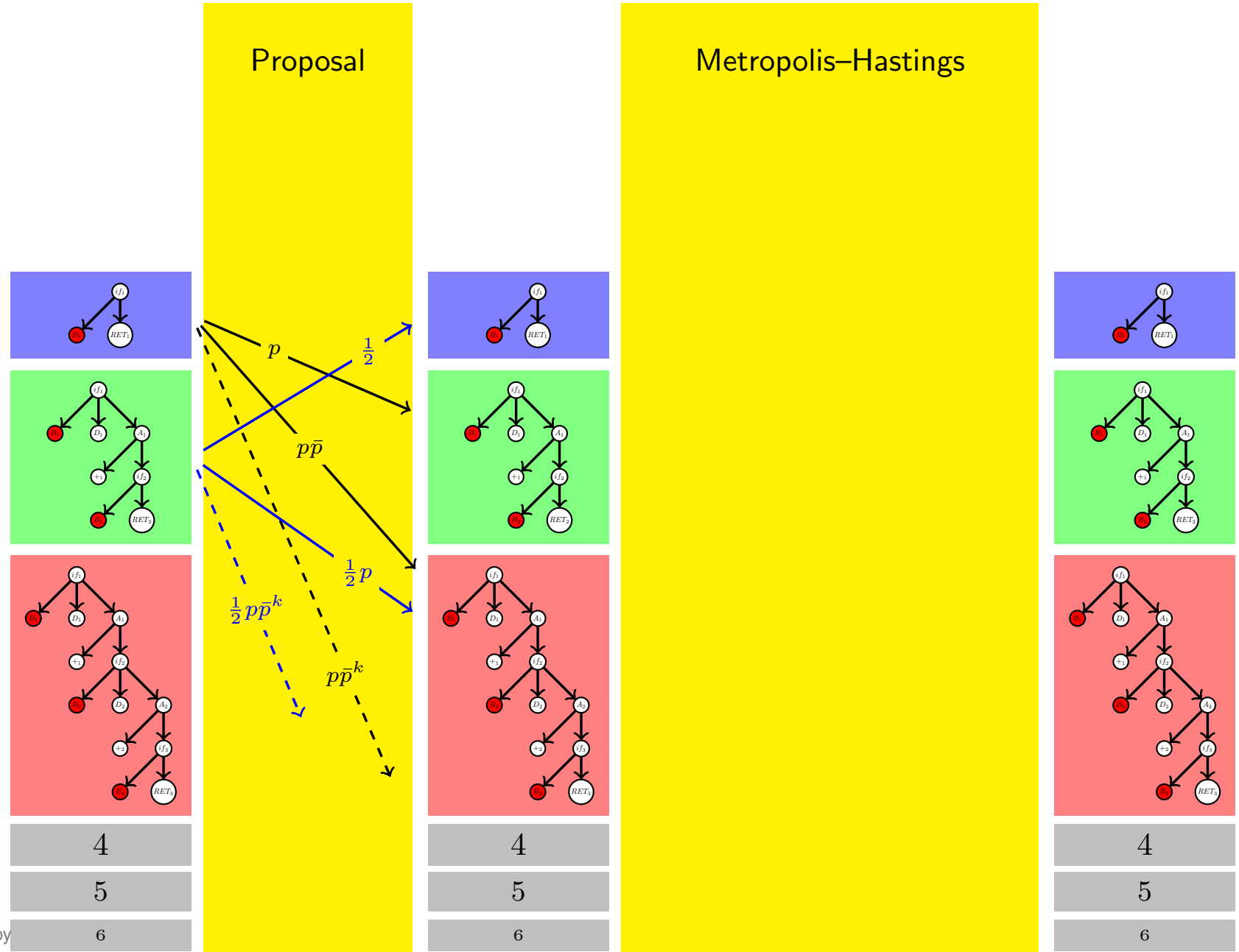
MIT-Church aka Trace-MCMC



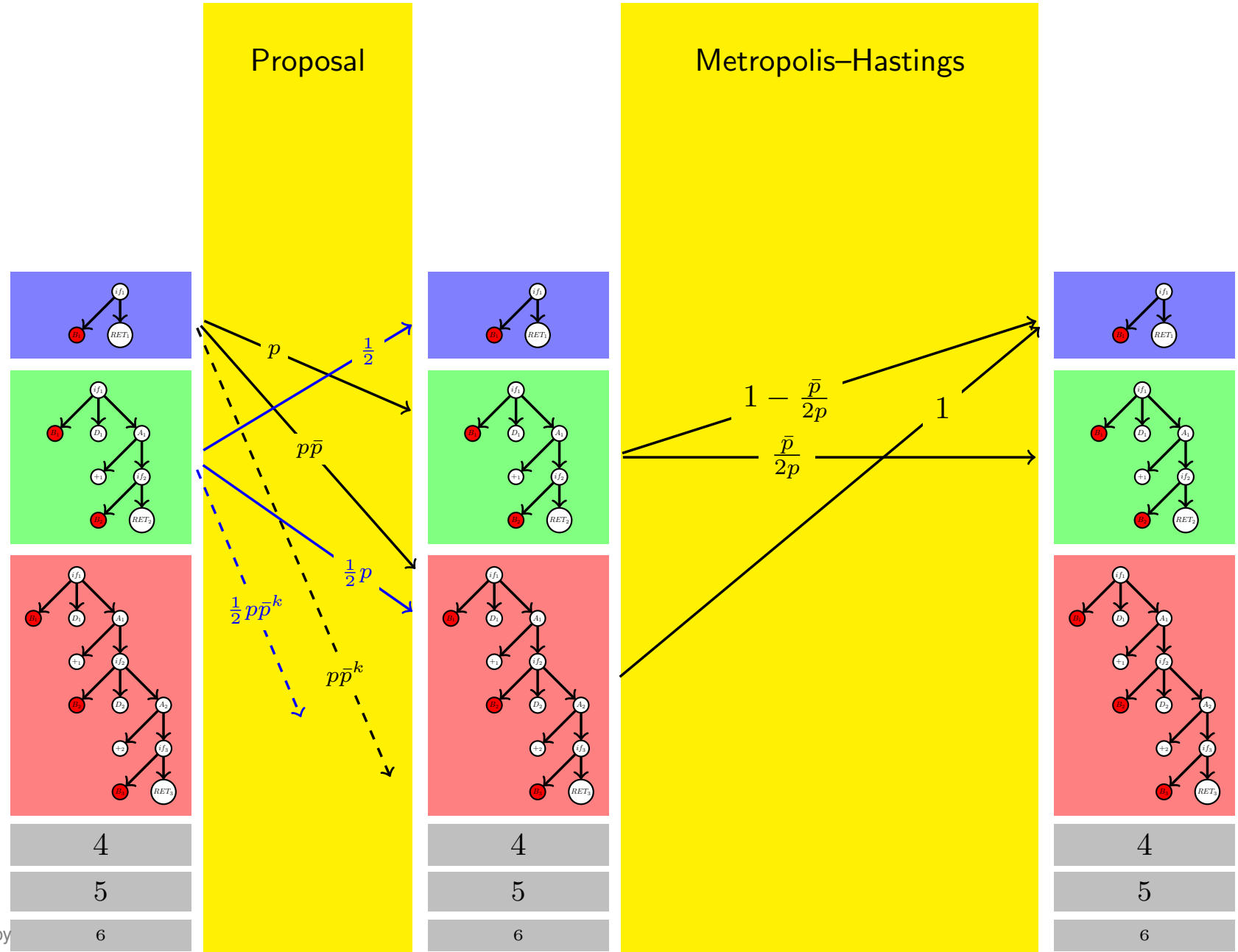
MIT-Church aka Trace-MCMC



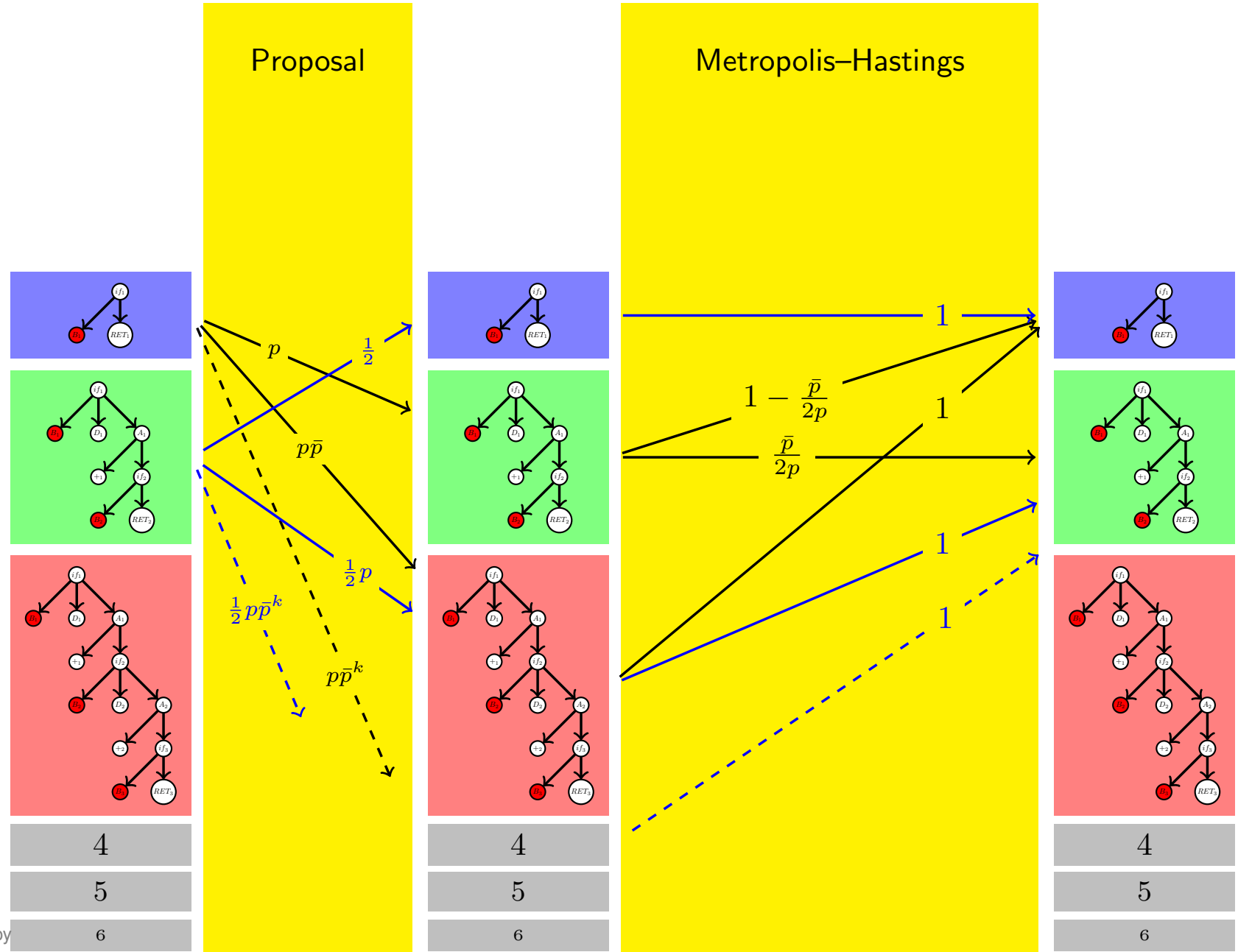
MIT-Church aka Trace-MCMC



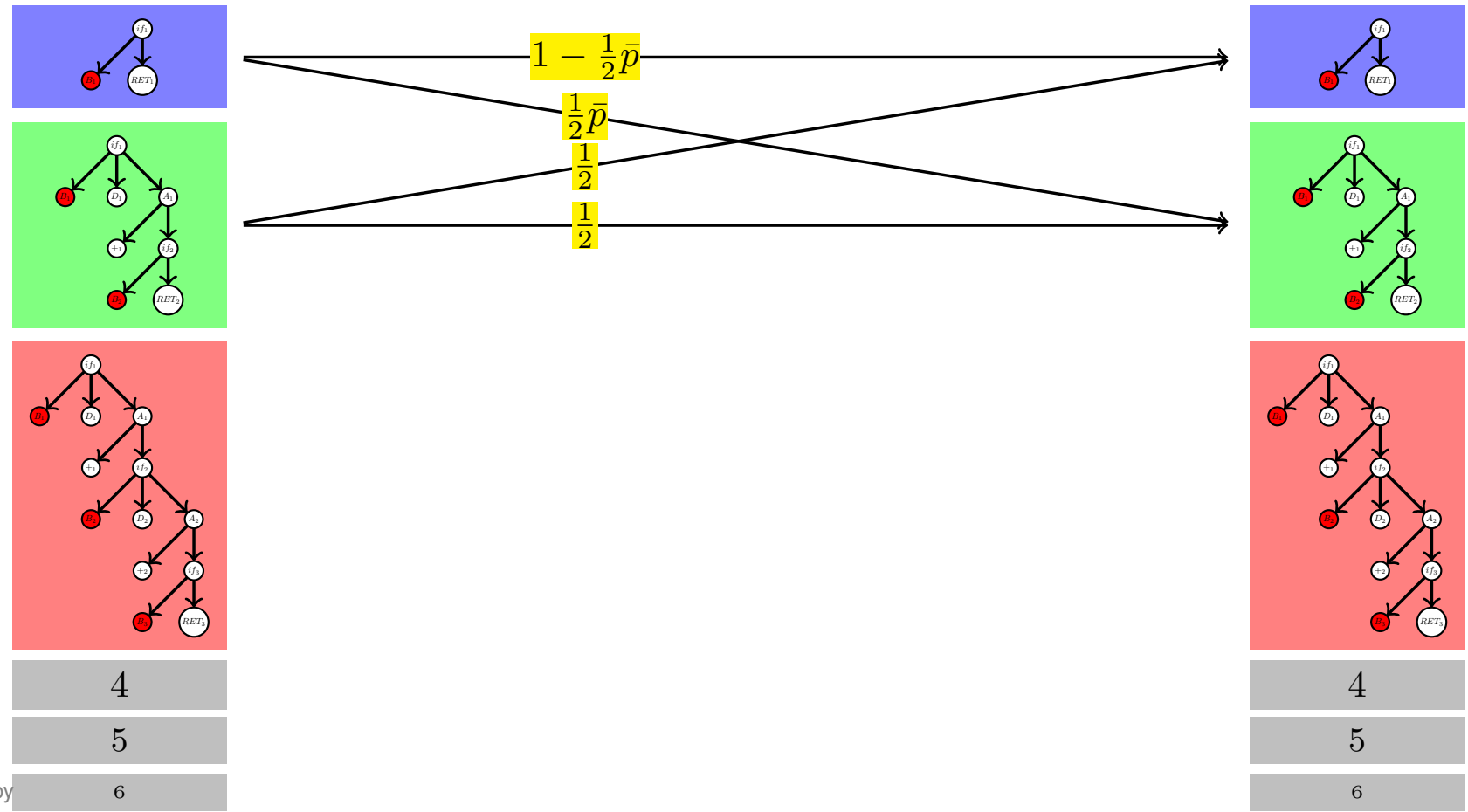
MIT-Church aka Trace-MCMC



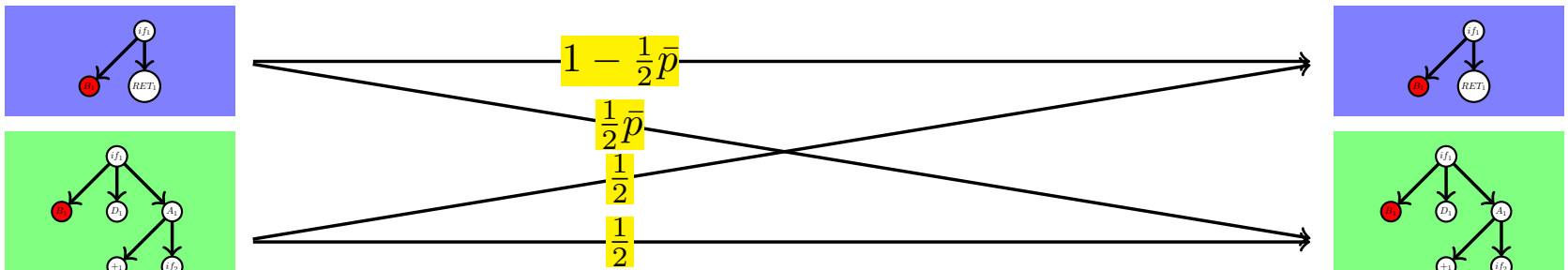
MIT-Church aka Trace-MCMC



MIT-Church aka Trace-MCMC



MIT-Church aka Trace-MCMC



$$\begin{pmatrix} 1-\frac{1}{2}\bar{p} & \frac{1}{2}\bar{p} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}^k \to \begin{pmatrix} \frac{p}{p+p\bar{p}} & \frac{\bar{p}}{p+p\bar{p}} \\ \frac{p}{p+p\bar{p}} & \frac{\bar{p}}{p+p\bar{p}} \end{pmatrix} \quad \text{as } k \to \infty$$

4

5

6

QUERY CAN CAPTURE A WIDE RANGE OF AI BEHAVIORS

Despite the apparent simplicity of the QUERY construct, we will see that it captures the essential structure of a range of common-sense inferences. We now demonstrate the power of the QUERY formalism by exploring its behavior in a medical diagnosis example.

MEDICAL DIAGNOSIS MODEL DESCRIPTION, PART I

The remainder of the tutorial will use a medical diagnosis task as a running example. The goal is to link observed symptoms to (unobserved) diseases. The stochastic inference problem is:

```
QUERY(diseasesAndSymptoms,checkSymptoms)
```

All that remains is to define the two procedures that define the model. `diseasesAndSymptoms()` will produce a random (possibly empty) set of diseases and associated symptoms, modeling the distribution of diseases and symptoms of a random chosen patient arriving at a clinic. `checkSymptoms(...)` checks the hypothesized symptoms against the list of observed symptoms, accepting if there is a match

MEDICAL DIAGNOSIS MODEL DESCRIPTION, PART II

The prior program `diseasesAndSymptoms()` proceeds as follows:

(1) For each disease n, sample an independent binary random variable D_n with mean p_n where

n	Disease	p_n
1	Arthritis	0.06
2	Asthma	0.04
3	Diabetes	0.11
4	Epilepsy	0.002
5	Giardiasis	0.006
6	Influenza	0.08
7	Measles	0.001
8	Meningitis	0.003
9	MRSA	0.001
10	Salmonella	0.002
11	Tuberculosis	0.003

D_n indicates whether or not a patient has disease n.

MEDICAL DIAGNOSIS MODEL DESCRIPTION, PART III

(2) For each symptom m, sample an independent binary random variable L_m with mean ℓ_m where

m	Symptom	ℓ_m
1	Fever	0.06
2	Cough	0.04
3	Hard breathing	0.001
4	Insulin resistant	0.15
5	Seizures	0.002
6	Aches	0.2
7	Sore neck	0.006

L_m indicates whether or not a patient spontaneously presents symptom m.

MEDICAL DIAGNOSIS MODEL DESCRIPTION, PART IV

(3) For each pair of disease n and symptom m, sample an independent binary random variable $C_{n,m}$ with mean $c_{n,m}$ where

$c_{n,m}$	1	2	3	4	5	6	7
1	.1	.2	.1	.2	.2	.5	.5
2	.1	.4	.8	.3	.1	.0	.1
3	.1	.2	.1	.9	.2	.3	.5
4	.4	.1	.0	.2	.9	.0	.0
5	.6	.3	.2	.1	.2	.8	.5
6	.4	.2	.0	.2	.0	.7	.4
7	.5	.2	.1	.2	.1	.6	.5
8	.8	.3	.0	.3	.1	.8	.9
9	.3	.2	.1	.2	.0	.3	.5
10	.4	.1	.0	.2	.1	.1	.2
11	.3	.2	.1	.2	.2	.3	.5

Conditioned on having disease n, $C_{n,m}$ indicates whether or not disease n *causes* the patient to present symptom m.

MEDICAL DIAGNOSIS MODEL DESCRIPTION, PART V

For each symptom m, we then define

$$S_m = \max\{L_m, D_1 \cdot C_{1,m}, \ldots, D_{11} \cdot C_{11,m}\},$$

hence $S_m \in \{0, 1\}$. (The $\max$ operator is playing the role of a logical OR operation.)

S_m indicates whether or not the patient presents symptom m.

Every term of the form $D_n \cdot C_{n,m}$ is interpreted as indicating whether (or not) the patient has disease n *and* disease n has caused symptom m. The term L_m captures the possibility that the symptom may present itself despite the patient having none of the listed diseases.

Finally, define the output of `diseasesAndSymptoms` to be the vector $(D_1, \ldots, D_{11}, S_1, \ldots, S_7)$.

EXPLORING THE MODEL

If we repeatedly evaluate `diseasesAndSymptoms`, we might see outputs like those in the following array:

	Diseases											Symptoms						
	1	2	3	4	5	6	7	8	9	10	11	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
4	0	0	1	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The rows model eight patients chosen randomly:

1. diseases and symptom free
2. same
3. suffering from diabetes and presenting insulin resistance;
4. suffering from diabetes and influenza, and presenting a fever and insulin resistance;
5. suffering from unexplained aches;
6. free from disease and symptoms;
7. suffering from diabetes, and presenting insulin resistance and aches;
8. disease and symptom free.

This model is a toy version of the real diagnostic model QMR-DT [Shwe et al., 1991], built from the Quick Medical Reference (QMR) knowledge base of hundreds of diseases and thousands of findings (such as symptoms or test results). A key aspect of this model is the disjunctive relationship between the diseases and the symptoms, known as a "noisy-OR".

Shortcomings

As a model of natural patterns of diseases and symptoms in a random patient, leaves much to desire:

- model assumes that the presence or absence of any two diseases is independent, *although*, as we will see later on in our analysis, diseases are (as expected) typically not independent conditioned on symptoms.
  - diseases may cause other diseases, and symptoms may cause diseases
- QMR-DT, like our toy model, was major advance over earlier expert systems and probabilistic models, allowing simultaneous occurrence of multiple diseases Shwe et al. [1991].

These caveats notwithstanding, a close inspection of this simplified model will demonstrate a surprising range of common-sense reasoning phenomena.

INCORPORATING OBSERVED SYMPTOMS

Consider the predicate that accepts if and only if $S_1 = 1$ and $S_7 = 1$, i.e., if and only if the patient presents the symptoms of a fever and a sore neck, ignoring other symptoms.

```
def checkSymptoms(D_1,...,D_11,S_1,...,S_7):
    return S_1 == 1 and S_7 == 1
```

What does $\texttt{QUERY}(\texttt{diseasesAndSymptoms}, \texttt{checkSymptoms})$ produce? We can just run it! (Right?)

Let μ denote output distribution of `diseasesAndSymptoms`, and let $A = \{(d_1, \ldots, d_{11}, s_1, \ldots, s_7) : s_1 = s_7 = 1\}$. Then $\texttt{QUERY}(\texttt{diseasesAndSymptoms}, \texttt{checkSymptoms})$ generates samples from the conditioned distribution $\mu(\cdot \mid A)$.

We will study the conditional distributions of the diseases given the symptoms. The following calculations may be very familiar to some readers, but will be less so to others, and so we present them here to give a more complete picture of the behavior of `QUERY`.

Posterior calculations given $S_1 = S_7 = 1$, Part I

Consider a $\{0,1\}$-assignment d_n for each disease n, and write $D = d$ to denote the event that $D_n = d_n$ for every such n. We'd like to be able to compute $\mathbb{P}\{D = d | S_1 = S_7 = 1\}$.

How? We know that output of `diseasesAndSymptoms` is produced from a collection of independent random variables. We will use this fact to compute $\mathbb{P}\{D = d\}$ and $\mathbb{P}\{S_1 = S_7 = 1 \,|\, D = d\}$. We will then employ the following identities

$$\mu(B)\mu(A \,|\, B) = \mu(A \cap B) = \mu(B|A)\mu(A) \tag{2}$$

Rearranging, we obtain **Bayes rule**,

$$\mu(A \,|\, B) = \frac{\mu(B|A)\mu(A)}{\mu(B)}. \tag{3}$$

Posterior calculations given $S_1 = S_7 = 1$, Part II

Assume for the moment that $D = d$. Then what is the probability that `checkSymptoms` accepts? The probability we are seeking is the conditional probability

$$\mathbb{P}(S_1 = S_7 = 1 \,|\, D = d) \tag{4}$$

$$= \mathbb{P}(S_1 = 1 \,|\, D = d) \cdot \mathbb{P}(S_7 = 1 \,|\, D = d), \tag{5}$$

where the equality follows from the observation that once the D_n variables are fixed, the variables S_1 and S_7 are independent. To see this, recall that

$$S_m = \max\{L_m, D_1 \cdot C_{1,m}, \ldots, D_{11} \cdot C_{11,m}\},$$

and see there's no overlap in variables determining each S_m once D is fixed.

Posterior calculations given $S_1 = S_7 = 1$, Part III

Note that $S_m = 1$ if and only if $L_m = 1$ or $C_{n,m} = 1$ for some n such that $d_n = 1$. (Equivalently, $S_m = 0$ if and only if $L_m = 0$ and $C_{n,m} = 0$ for all n such that $d_n = 1$.) By the independence of each of these variables, it follows that

$$\mathbb{P}(S_m = 1 | D = d) = 1 - (1 - \ell_m) \prod_{n \,:\, d_n = 1} (1 - c_{n,m}). \tag{6}$$

It's difficult to visualize a distribution on a 11-dimensional vector, such as D. Instead, let d, d' be $\{0,1\}$-assignment specifying different patterns of diseases. We can characterize the *a posteriori* odds

$$\frac{\mathbb{P}(D = d \,|\, S_1 = S_7 = 1)}{\mathbb{P}(D = d' \,|\, S_1 = S_7 = 1)}$$

of the assignment d versus the assignment d' in order to understand how much more likely we are to see d as an explanation versus d'.

Posterior calculations given $S_1 = S_7 = 1$, Part IV

By Bayes' rule, this can be rewritten as

$$\frac{\mathbb{P}(S_1 = S_7 = 1 \,|\, D = d) \cdot \mathbb{P}(D = d)}{\mathbb{P}(S_1 = S_7 = 1 \,|\, D = d') \cdot \mathbb{P}(D = d')}, \tag{7}$$

where $\mathbb{P}(D = d) = \prod_{n=1}^{11} \mathbb{P}(D_n = d_n)$ by independence. Using (5), (6) and (7), one may calculate that

$$\frac{\mathbb{P}(\text{Patient only has influenza} \,|\, S_1 = S_7 = 1)}{\mathbb{P}(\text{Patient has no listed disease} \,|\, S_1 = S_7 = 1)} \approx 42,$$

i.e., it is forty-two times more likely that an execution of `diseasesAndSymptoms` satisfies the predicate `checkSymptoms` via an execution that posits the patient only has the flu than an execution which posits that the patient has no disease at all.

Posterior calculations given $S_1 = S_7 = 1$, Part V

On the other hand,

$$\frac{\mathbb{P}(\text{Patient only has meningitis} \,|\, S_1 = S_7 = 1)}{\mathbb{P}(\text{Patient has no listed disease} \,|\, S_1 = S_7 = 1)} \approx 6,$$

and so

$$\frac{\mathbb{P}(\text{Patient only has influenza} \,|\, S_1 = S_7 = 1)}{\mathbb{P}(\text{Patient only has meningitis} \,|\, S_1 = S_7 = 1)} \approx 7,$$

and hence we would expect to see, over many executions of

$$\texttt{QUERY}(\texttt{diseasesAndSymptoms}, \texttt{checkSymptoms}), \tag{8}$$

roughly seven times as many explanations positing only influenza than positing only meningitis.

Explaining away

Further investigation reveals some subtle aspects of the model. For example,

$$\frac{\mathbb{P}(\text{Patient only has meningitis and influenza} \,|\, S_1 = S_7 = 1)}{\mathbb{P}(\text{Patient has meningitis, maybe influenza, but nothing else} \,|\, S_1 = S_7 = 1)}$$
$$= 0.09 \approx \mathbb{P}(\text{Patient has influenza}),$$

Observation 1

Once we have observed some symptoms, diseases are no longer independent.

Observation 2

Once the symptoms have been "explained" (e.g., as coming from meningitis), there is little pressure to posit further causes (the *posterior* probability of influenza is essentially the *prior* probability of influenza).

This phenomenon is well-known and is called ***explaining away***; it is also known to be linked to the computational hardness of computing probabilities (and generating samples as `QUERY` does) in models of this variety.

SIMPLE MODELS CAN YIELD COMPLEX BEHAVIOR, PART I

Despite the simple model and simple query, `QUERY(diseasesAndSymptoms, checkSymptoms)` yields a collection of diagnostic inferences with tremendous internal complexity.

Many more behaviors available through different predicates:

- The model naturally handles missing data. `checkSymptoms` leads to different conclusions than

```
def checkSymptoms(D_1,...,D_11,S_1,...,S_7):
    return S_1 == 1 and S_7 == 1 and sum_{m=1}^{7} S_m == 2
```

SIMPLE MODELS CAN YIELD COMPLEX BEHAVIOR, PART II

We need not limit ourselves to reasoning about diseases given symptoms.

- Imagine that we perform a diagnostic test that rules out meningitis. We could represent our new knowledge using a predicate capturing the condition

$$(D_8 = 0) \wedge (S_1 = S_7 = 1) \wedge (S_2 = \cdots = S_6 = 0).$$

  Of course this approach would not take into consideration our uncertainty regarding the accuracy or mechanism of the diagnostic test itself, and so, ideally, we might expand the `diseasesAndSymptoms` model to account for how the outcomes of diagnostic tests are affected by the presence of other diseases or symptoms. Later, we will discuss how such an extended model might be learned from data, rather than constructed by hand.

SIMPLE MODELS CAN YIELD COMPLEX BEHAVIOR, PART III

We can also reason in the other direction, about symptoms given diseases.

- For example, public health officials might wish to know about how frequently those with influenza present no symptoms. This is captured by the conditional probability

$$\mathbb{P}(S_1 = \cdots = S_7 = 0 \,|\, D_6 = 1),$$

  and, via `QUERY`, by the predicate for the condition $D_6 = 1$. Unlike the earlier examples where we reasoned backwards from effects (symptoms) to their likely causes (diseases), here we are reasoning in the same forward direction as the model `diseasesAndSymptoms` is expressed.

The possibilities are effectively inexhaustible, including more complex states of knowledge such as, *there are at least two symptoms present*, or *the patient does not have both salmonella and tuberculosis*. Later, we will consider the vast number of predicates and the resulting inferences supported by `QUERY` and `diseasesAndSymptoms`, and contrast this with the compact size of `diseasesAndSymptoms` and the table of parameters.

In this section, we illustrated the basic behavior of `QUERY`, and began to explore how `QUERY` can be used to update beliefs in light of observations.

- Inferences need not be explicitly described in terms of rules, but can arise implicitly via other mechanisms, like `QUERY`, paired with an appropriate models and predicates.
  - In the working example, the diagnostic rules were determined by the definition of `diseasesAndSymptoms` and the table of its parameters.
- The inference, however, are fixed.
  - We will examine how the underlying table of probabilities might be learned from data.
  - The structure of `diseasesAndSymptoms` itself encodes strong structural relationships among the diseases and symptoms. We will study how to learn this in part 2.
  - Finally, many common-sense reasoning tasks involve making a *decision*, and not just determining what to believe. Towards the end, we will describe how to use `QUERY` to make decisions under uncertainty.

Conditional independence and compact representations

In this section, we return to the medical diagnosis example, and explain the way in which conditional independence leads to compact representations, and conversely, the fact that efficient probabilistic programs, like `diseasesAndSymptoms`, exhibit many conditional independencies. We will do so through connections with the Bayesian network formalism, whose introduction by Pearl [1988] was a major advancement in AI.

THE COMBINATORICS OF QUERY

Common-sense reasoning seems to encompass an unbounded range of responses / behaviors. How are these compactly represented?

In fact, the small number of diseases and symptoms considered in our simple medical diagnosis model already leads to a combinatorial number of potential scenarios: among 11 potential diseases and 7 potential symptoms there are

$$3^{11} \cdot 3^{7} = 387\,420\,489$$

partial assignments to a subset of variables. All of these must be assigned probabilities!
Luckily, these 387 420 489 probabilities are determined by

$$2^{11} \cdot 2^{7} - 1 = 262\,143$$

probabilities, one each for every complete assignment. Still, this number is exponential in the number of diseases and symptoms. Even if we discretize the probabilities to some fixed accuracy, a simple counting argument shows that most such distributions have no short description.

Few distributions have compact representations

Like `diseasesAndSymptoms`, every probability distribution on 18 binary variables implicitly defines a large set of probabilities.

- Not feasible to store these probabilities explicitly.
- Necessary to exploit *structure* to devise more compact representations.

`diseasesAndSymptoms` is a small efficient program acting on three tables with

$$11 + 7 + 11 \cdot 7 = 95$$

probabilities. In contrast, a generic distribution on 2^{18} possibilities has no short description.

- `diseasesAndSymptoms` implicitly represents $2^{18} - 1$ probabilities via an efficient (and short) simulator.
  - What structure suffices to yield compact representations?
  - What structure is necessary given efficient representations?

CONDITIONAL INDEPENDENCE, PART I

The answer to both questions is *conditional independence*.

Recall that a collection of random variables $\{X_i : i \in I\}$ is **independent** when, for all finite subsets $J \subseteq I$ and measurable sets A_i where $i \in J$, we have

$$\mathbb{P}\Big(\bigwedge_{i \in J} X_i \in A_i\Big) = \prod_{i \in J} \mathbb{P}(X_i \in A_i). \tag{9}$$

If X and Y were binary random variables, then specifying their distribution would require 3 probabilities in general, but only 2 if they were independent. While those savings are small, consider instead m binary random variables X_j, $j = 1, \ldots, m$, and note that, while a generic distribution over these random variables would require the specification of $2^m - 1$ probabilities, only m probabilities are needed in the case of full independence.

Full independence is rare and so this factorization is not the whole story.

CONDITIONAL INDEPENDENCE, PART II

Conditional independence is arguably more fundamental.

We consider a special case of conditional independence, restricting our attention to conditional independence with respect to a discrete random variable N taking values in some countable or finite set $\mathcal{N}$.

We say that a collection of random variables $\{X_i : i \in I\}$ is **conditionally independent** given N when, for all $n \in \mathcal{N}$, finite subsets $J \subseteq I$ and measurable sets A_i, for $i \in J$, we have

$$\mathbb{P}\Big(\bigwedge_{i \in J} X_i \in A_i \,|\, N = n\Big) = \prod_{i \in J} \mathbb{P}(X_i \in A_i \,|\, N = n).$$

To illustrate the potential savings that can arise from conditional independence, consider m binary random variables that are conditionally independent given a discrete random variable taking k values. In general, the joint distribution over these $m + 1$ variables is specified by $k \cdot 2^m - 1$ probabilities, but, in light of the conditional independence, we need specify only $km + k - 1$ probabilities.

Conditional independence in diseasesAndSymptoms, Part I

Conditional independence gives rise to compact representations. Indeed, `diseasesAndSymptoms` exhibits many conditional independencies.

To begin to understand the compactness of `diseasesAndSymptoms`, note that the 95 variables

$$\{D_1, \ldots, D_{11};\ L_1, \ldots, L_7;\ C_{1,1}, C_{1,2}, C_{2,1}, C_{2,2}, \ldots, C_{11,7}\}$$

are independent, and thus their joint distribution is determined by specifying only 95 probabilities (in particular, those in the tables).

Conditional independence in diseasesAndSymptoms, Part II

Each symptom S_m is a deterministic function of a 23-variable subset

$$\{D_1, \ldots, D_{11};\, L_m;\, C_{1,m}, \ldots, C_{11,m}\}.$$

The variables $L_m;\, C_{1,m}, \ldots, C_{11,m}$ are not shared across symptoms, implying symptoms are conditionally independent given diseases.

However, these facts alone do not fully explain the compactness of `diseasesAndSymptoms`. In particular, there are

$$2^{2^{23}} > 10^{10^6}$$

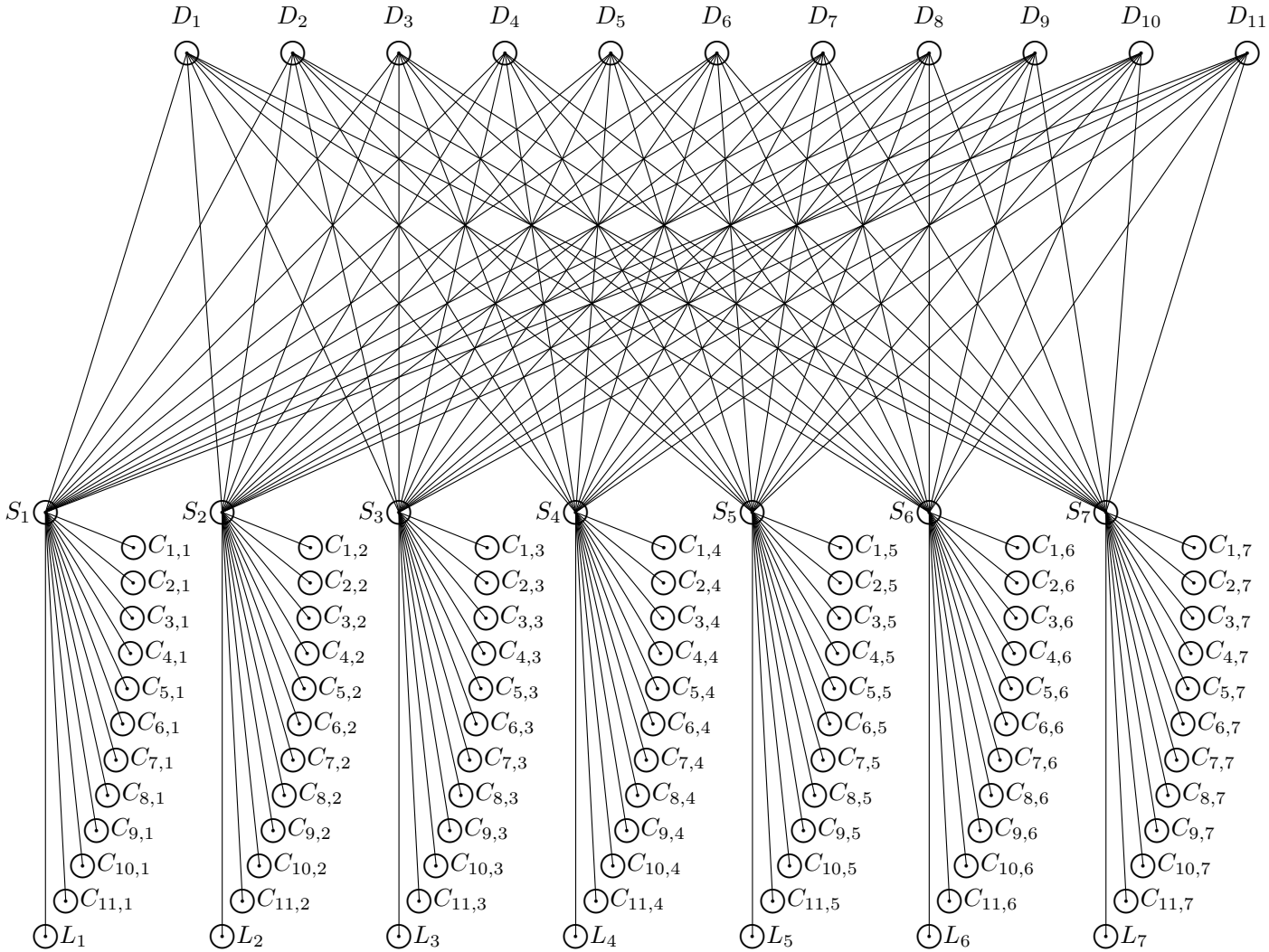
binary functions of 23 binary inputs, and so by a counting argument, most have no short description. On the other hand, the $\max$ operation that defines S_m does have a compact *and efficient* implementation.

What's the connection?

Representations of conditional independence

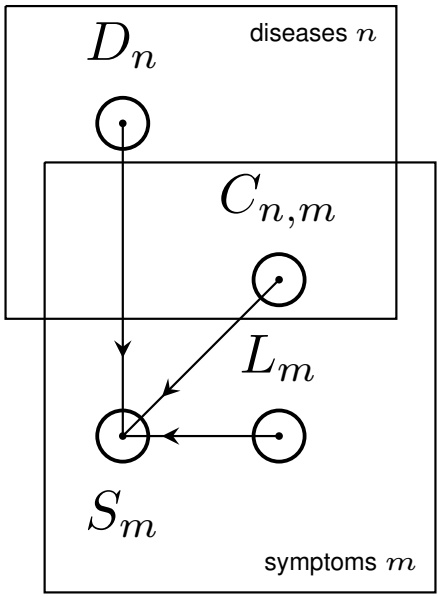
A useful way to represent conditional independence among a collection of random variables is in terms of a directed acyclic graph, where the vertices stand for random variables, and the collection of edges indicates the presence of certain conditional independencies. Such a graph is known as a directed graphical model or Bayesian network. (For more details on Bayesian networks, see the survey by Pearl [2004].)

BAYES NET FOR DISEASESANDSYMPTOMS



Bayes net for the medical diagnosis example. (Note that the directionality of the arrows is not rendered for clarity. All arrows point to the symptoms S_m.)

PROTO-LANGUAGES FOR BAYES NETS



The repetitive structure can be partially captured by so-called “plate notation”, which can be interpreted as a primitive `for`-loop construct. Practitioners have adopted a number of strategies like plate notation for capturing complicated structures.

D-SEPARATION IN BAYES NETS I

In order to understand exactly which conditional independencies are formally encoded in such a graph, we must introduce the notion of d-separation.

d-separation

A pair (x, y) of vertices are d-separated by a subset of vertices $\mathcal{E}$ as follows: First, mark each vertex in $\mathcal{E}$ with a $\times$, which we will indicate by the symbol $\otimes$. If a vertex with (any type of) mark has an unmarked parent, mark the parent with a $+$, which we will indicate by the symbol $\oplus$. Repeat until a fixed point is reached. Let $\odot$ indicate unmarked vertices.

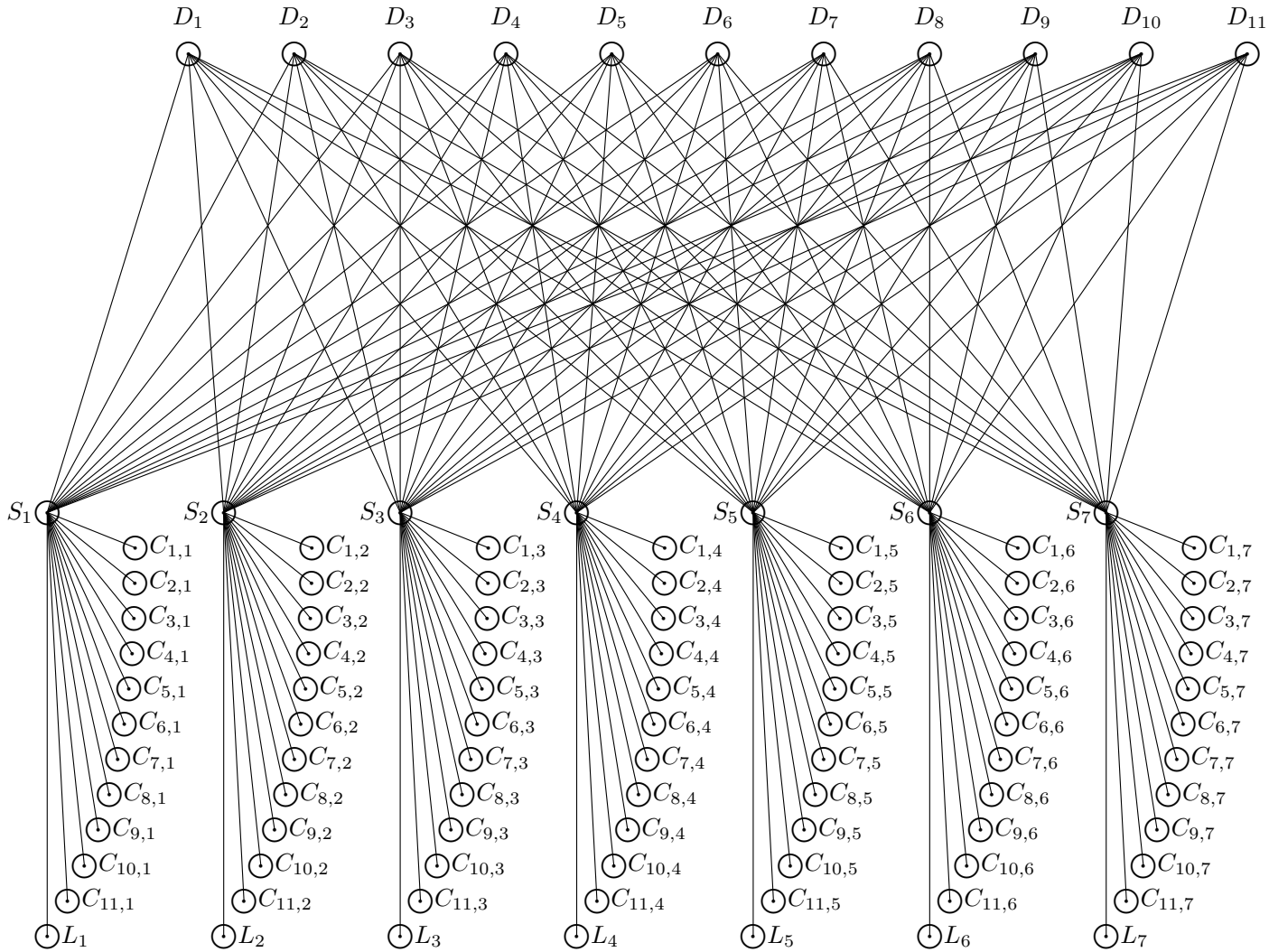
Definition. x and y are d-separated by $\mathcal{E}$ if, for all (undirected) paths from x to y, one of the following patterns appears:

$$\odot \to \otimes \to \odot \qquad \odot \leftarrow \otimes \leftarrow \odot$$
$$\odot \leftarrow \otimes \to \odot \qquad \odot \to \odot \leftarrow \odot$$

D-SEPARATION IN BAYES NETS II

More generally, if $\mathcal{X}$ and $\mathcal{E}$ are disjoint sets of vertices, then the graph encodes the conditional independence of the vertices $\mathcal{X}$ given $\mathcal{E}$ if every pair of vertices in $\mathcal{X}$ is d-separated by $\mathcal{E}$.

If we fix a collection V of random variables, then we say that a directed acyclic graph G over V is a *Bayesian network* when the random variables in V indeed posses all of the conditional independencies implied by the graph by d-separation. Note that a Bayes net G says nothing about which conditional independencies do *not* exist among its vertex set.



Deciding d-separation in the Bayes net, we can determine that (1) the diseases $\{D_1, \ldots, D_{11}\}$ are independent (i.e., conditionally independent given $\mathcal{E} = \emptyset$) and that (2) the symptoms $\{S_1, \ldots, S_7\}$ are conditionally independent given the diseases $\{D_1, \ldots, D_{11}\}$ and many more.

CONDITIONAL INDEPENDENCE AND FACTORIZATION I

Bayes nets specify a factorization of the joint distribution of the vertex set.

It is a basic fact from probability that

$$\begin{aligned}
&\mathbb{P}(X_1 = x_1, \ldots, X_k = x_k) \\
&= \mathbb{P}(X_1 = x_1) \cdot \mathbb{P}(X_2 = x_2 \,|\, X_1 = x_1) \cdots \mathbb{P}(X_k = x_k \,|\, X_j = x_j,\ j < k) \\
&= \prod_{j=1}^{k} \mathbb{P}(X_j = x_j \,|\, X_i = x_i,\ i < j).
\end{aligned}$$

Such a factorization provides no advantage when seeking a compact representation, as the j'th conditional distribution is determined by 2^{j-1} probabilities.

CONDITIONAL INDEPENDENCE AND FACTORIZATION II

On the other hand, if we have a Bayes net over the same variables, then we may have a much more concise factorization.

Let G be a Bayes net over $\{X_1, \ldots, X_k\}$, and write $\mathrm{Pa}(X_j)$ for the set of indices i such that $(X_i, X_j) \in G$, i.e., $\mathrm{Pa}(X_j)$ indexes the parent vertices of X_j. Then the joint p.m.f. may be expressed as

$$\mathbb{P}(X_1 = x_1, \ldots, X_k = x_k) = \prod_{j=1}^{k} \mathbb{P}(X_j = x_j \,|\, X_i = x_i, i \in \mathrm{Pa}(X_j)).$$

This factorization is determined by only $\sum_{j=1}^{k} 2^{|\mathrm{Pa}(X_j)|}$ probabilities, an exponential savings potentially.

Efficient representations and conditional independence, Part I

As we saw at the beginning of this section, models with only a moderate number of variables can have enormous descriptions. Having introduced the Bayesian network formalism, we can use `diseasesAndSymptoms` as an example to explain why, roughly speaking, the output distributions of efficient probabilistic programs exhibit many conditional independencies.

Efficient representations and conditional independence, Part II

What does the efficiency of `diseasesAndSymptoms` imply about the structure of its output distribution?

Assuming the 95 tabulated probabilities are dyadics of the form $\frac{k}{2^m}$, we may represent `diseasesAndSymptoms` as a small boolean circuit whose inputs are random bits and whose 18 output lines represent the diseases and symptom indicators. Each circuit elements can be restricted to constant-fan-in, and the total number of circuit elements grows only linearly in the number of diseases and in the number of symptoms, assuming fixed accuracy for the bae probabilities.

Efficient representations and conditional independence, Part III

If we view the input lines as random variables, then the output lines of the logic gates are also random variables, and so we may ask: what conditional independencies hold among the circuit elements?

It is straightforward to show that the circuit diagram, viewed as a directed acyclic graph, is a Bayes net capturing conditional independencies among the inputs, outputs, and internal gates of the circuit implementing `diseasesAndSymptoms`. For every gate, the conditional probability mass function is characterized by the (constant-size) truth table of the logical gate.

Therefore, if an efficient prior program samples from some distribution over a collection of binary random variables, then those random variables exhibit many conditional independencies, in the sense that we can introduce a polynomial number of additional boolean random variables (representing intermediate computations) such that there exists a constant-fan-in Bayes net over all the variables.

GRAPHICAL MODELS IN AI

Graphical models, and, in particular, Bayesian networks, played a critical role in popularizing probabilistic techniques within AI in the late 1980s and early 1990s.
Two developments were central to this shift:

1. Researchers introduced compact, computer-readable representations of distributions on large (but still finite) collections of random variables, and did so by explicitly representing a graph capturing conditional independencies and exploiting the factorization (**??**).
2. Researchers introduced efficient graph-based algorithms that operated on these representations, exploiting the factorization to compute conditional probabilities.

For the first time, a large class of distributions were given a formal representation that enabled the design of general purpose algorithms to compute useful quantities.

Learning parameters via probabilistic inference I

The 95 tabulated probabilities induce a distribution over $262\,144$ outcomes.

- Where did these numbers come from?
- And are they any good?

In practice, these 95 "parameters" would themselves be subject to a great deal of uncertainty, and one might hope to use data from actual diagnostic situations to learn appropriate values.

Learning as probabilistic inference, Part I

The Bayesian approach to learning the 95 parameters is to place a prior distribution on them. Whereas different individuals diseases and symptoms were independent before, they are no longer independent.
Thus, this change affects `diseasesAndSymptoms` in two ways:

1. Rather than using the fixed 95 tabulated probabilities, the program will start by randomly generating the entries of the table. (Concretely, assume they are i.i.d. uniform in $[0, 1]$.)
2. The new program, dubbed `allDiseasesAndSymptoms`, then evaluates `diseasesAndSymptoms()` $n + 1$ times, returning the resulting $(n + 1) \times 18$ array.
3. Similarly, we modify `checkSymptoms` to obtain `checkAllSymptoms` by accepts the $n + 1$ diagnoses generated by `allDiseasesAndSymptoms` if and only if the first n agree with n historical records, and last one, the current patient, is accepted by `checkSymptoms`.

LEARNING AS PROBABILISTIC INFERENCE, PART II

The changes to `allDiseasesAndSymptoms` may sound quite surprising, and unlikely to work very well. Indeed, `allDiseasesAndSymptoms` will produce unnatural patterns. The key is to consider the effect of `checkAllSymptoms`. What are typical outputs from `QUERY(allDiseasesAndSymptoms, checkAllSymptoms)`?

For large n,

- hypothesized marginal probability of a disease relatively close to the frequency observed in the *simulated disease–symptom data*.

Thus, when a simulation is eventually accepted by `checkAllSymptoms`,

- the hypothesized marginal probabilities will closely match the relative frequencies *in the data*.

The concentration around the data occurs at a $n^{-1/2}$ rate, and so we would expect that the typical accepted sample would soon correspond with a latent table of probabilities that roughly matches the historical record.

EXACT POSTERIOR DISTRIBUTION OF p_j

Fix a disease j and recall that $p_j \sim \text{Uniform}[0,1]$.

The probability that the n sampled values of D_j match the historical record is

$$p_j^k \cdot (1-p_j)^{n-k}, \tag{10}$$

where k stands for the number of records where disease j is present.

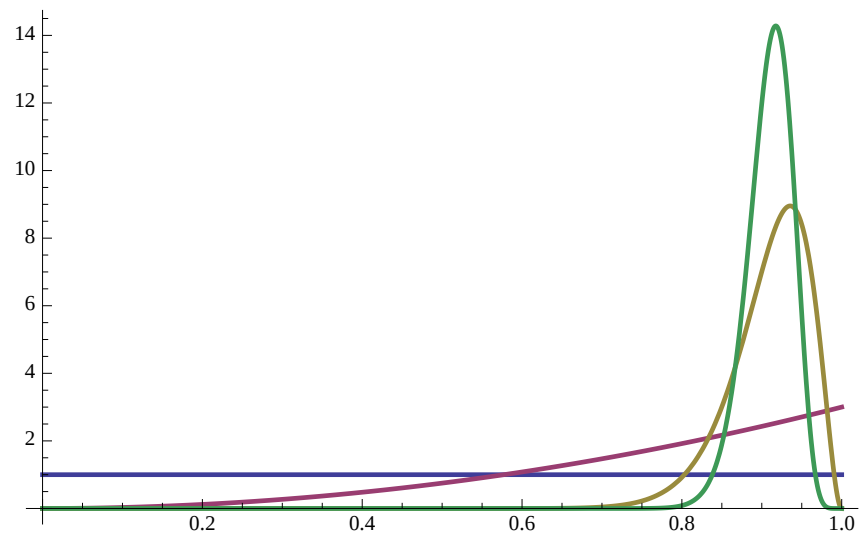
By Bayes' theorem, in the special case of a uniform prior distribution on p_j, the density of the conditional distribution of p_j given the historical evidence is proportional to the likelihood (10). This implies that, conditionally on the historical record, p_j has a so-called $\text{Beta}(\alpha_1, \alpha_0)$ distribution with mean

$$\frac{\alpha_1}{\alpha_1 + \alpha_0} = \frac{k+1}{n+2}$$

and concentration parameter $\alpha_1 + \alpha_0 = n + 2$.

EXACT POSTERIOR DISTRIBUTION OF p_j

Beta distributions under varying parameterizations, highlighting the fact that, as the concentration grows, the distribution begins to concentrate around its mean. As n grows, predictions made by $\texttt{QUERY}(\texttt{allDiseasesAndSymptoms}, \texttt{checkAllSymptoms})$ will likely be those of runs where each disease marginals p_j falls near the observed frequency of the jth disease. In effect, the historical record data *determines* the values of the marginals p_j.



Plots of the probability density of $\mathrm{Beta}(a_1, a_0)$ distributions with density $f(x; \alpha_1, \alpha_0) = \frac{\Gamma(\alpha_1+\alpha_0)}{\Gamma(\alpha_1)\Gamma(\alpha_0)} x^{\alpha_1-1}(1-x)^{\alpha_0-1}$ for parameters $(1,1)$, $(3,1)$, $(30,3)$, and $(90,9)$ (respectively, in height). For parameters $\alpha_1, \alpha_0 > 1$, the distribution is unimodal with mean $\alpha_1/(\alpha_1+\alpha_0)$.

POSTERIOR CONVERGENCE

A similar analysis can be made of the dynamics of the posterior distribution of the latent parameters ℓ_m and $c_{n,m}$.

Abstractly speaking, in finite dimensional Bayesian models like this one satisfying certain regularity conditions, it is possible to show that the predictions of the model converge to those made by the best possible approximation within the model to the distribution of the data. (For a discussion of these issues, see, e.g., Barron [1998].)

Bayesian learning, Part I

- Original `diseasesAndSymptoms` program makes the same inferences in each case, even in face of large data.
- `allDiseasesAndSymptoms` learns from data.

The key: the latent table of probabilities, modeled as random variables.

Similar approaches can be used when the patients come from multiple distinct populations where you do not expect the patterns of diseases and symtoms to agree.

BAYESIAN LEARNING, PART II

Note

- `allDiseasesAndSymptoms` is even more compact than `diseasesAndSymptoms`
  - the specification of the distribution of the random table is logarithmic in the size of the table
- On the other hand, `allDiseasesAndSymptoms` relies on data to help it reduce its substantial *a priori* uncertainty regarding these values.

This tradeoff—between, on the one hand, the flexibility and complexity of a model and, on the other, the amount of data required in order to make sensible predictions—is seen throughout statistical modeling. We will return to this point later.

Here we have seen how the parameters in prior programs can be modeled as random, and thereby learned from data by conditioning on historical diagnoses. In the next section, we consider the problem of learning not only the parameterization but the structure of the model's conditional independence itself.

Learning conditional independences I

Key limitation of `diseasesAndSymptoms`

`diseasesAndSymptoms` uses a fixed noisy-OR network to perform inference

Solution

`allDiseasesAndSymptoms` performs probabilistic inference over the probabilities to learn the best noisy-OR network from data.

Key limitation of `allDiseasesAndSymptoms`

Irrespective of how much historical data we have, `allDiseasesAndSymptoms` cannot go beyond the conditional independence assumptions implicit in the structure of the prior program.

Solution

Identify the correct structure of the dependence between symptoms and disease by probabilistic inference over random conditional independence *structure* among the model variables.

STRUCTURAL LEARNING VIA PROBABILISTIC INFERENCE

Learning a probabilistic model from data is a quintessential example of *unsupervised learning*. Learning conditional independence relationships among model variables is known as *structure learning*.

Need to learn the components of this factorization:

$$\mathbb{P}(X_1 = x_1, \ldots, X_k = x_k) = \prod_{j=1}^{k} \mathbb{P}(X_j = x_j \,|\, X_i = x_i, i \in \mathrm{Pa}(X_j)).$$

Approach

Be “Bayesian”... put prior distributions on graphs and conditional probabilities.

A RANDOM PROBABILITY DISTRIBUTION I

Consider a prior program, which we will call `RPD` (for *Random Probability Distribution*) that takes two positive integer inputs, n and D, and produces as output n independent samples drawn from a random probability distribution on $\{0,1\}^D$.

We will then perform inference as usual

```
QUERY(RPD(n+1,18),checkAllSymptoms)
```

A RANDOM PROBABILITY DISTRIBUTION II

Intuitively, RPD works in the following way:

1. RPD generates a random directed acyclic graph G over the vertex set $\{X_1, \ldots, X_D\}$.
2. For each vertex j, for each setting $v \in \{0,1\}^{\mathrm{Pa}(X_j)}$, RPD generates the value of the conditional probabilitiy $p_{j|v} = \mathbb{P}(X_j = 1 \,|\, X_i = v_i, i \in \mathrm{Pa}(X_j))$ uniformly at random.
3. Generate n samples, each a vector of D binary values with the same distributions as $X_1, \ldots, X_D$.

The first two steps produce a graph G and *random* probabliity distribution for which G is a valid Bayes net.

General Bayesian learning machine

With `RPD` fully specified, we'd like to understand the output of

$$\texttt{QUERY}(\texttt{RPD}(n+1, 18), \texttt{checkAllSymptoms}) \tag{11}$$

where `checkAllSymptoms` is defined earlier, accepting $n+1$ diagnoses if and only if the first n agree with historical records, and the symptoms associated with the $n+1$'st agree with the current patient's symptoms. (Note that we are identifying each output $(X_1, \ldots, X_{11}, X_{12}, \ldots, X_{18})$ with a diagnosis $(D_1, \ldots, D_{11}, S_1, \ldots, S_7)$, and have done so in order to highlight the generality of `RPD`.)

GRAPH-CONDITIONAL POSTERIOR, PART I

Assume we condition on the graph G produced on step 1. **Now we're essentially back to our earlier analysis.**

The expected value of $p_{j|v} = \mathbb{P}(X_j = 1 \,|\, X_i = v_i, i \in \mathrm{Pa}(X_j))$ on those runs accepted by `QUERY` is

$$\frac{k_{j|v} + 1}{n_{j|v} + 2}$$

where $n_{j|v}$ is the number of times in the historical data where the event $\{X_i = x_i, i \in \mathrm{Pa}(X_j)\}$ occurs; and $k_{j|v}$ is the number of times when, moreover, $X_j = 1$. This is simply the "smoothed" empirical frequency. In fact, the probability $p_{j|v}$ is conditionally Beta distributed with concentration $n_{j|v} + 2$.

GRAPH-CONDITIONAL POSTERIOR, PART II

The variance of $p_{j|v}$ is one characterization of our residual uncertainty, and for each probability j, the variance is easily shown to scale as $n_{j|v}^{-1}$, i.e., the number of times in the historical data when $\{X_i = x_i, i \in \mathrm{Pa}(X_j)\}$ occurs.

Informally,

- the smaller the parental sets (a property of G), the more certain we are likely to be regarding the correct parameterization
- in terms of `QUERY`, reflected in smaller the range of values of $p_{j|v}$ on accepted runs.

This is our first glimpse at a subtle balance between the simplicity (sparsity) of the graph G and how well it captures hidden structure in the data.

Aspects of the posterior distribution of the graphical structure

There are a number of practical roadblocks

- The space of directed acyclic graphs on 18 variables is enormous
- Computational hardness results abound [Cooper, 1990, Dagum and Luby, 1993, Chandrasekaran et al., 2008].
- Indeed, `QUERY` would not halt in reasonable time for even small n and D because the probability of generating the structure that fits the data is astronomically small.

State of the art structure learning algorithms operate in special subclasses of distributions and are very sophisticated.
Our goal here is understanding / intuition:

- we can aim to understand conceptual structure of the posterior distribution of the graph G, perhaps in simple cases
- this example reveals an important aspect of hierarchical Bayesian models with regard to their ability to avoid "overfitting", and gives some insight into why we might expect "simpler" explanations/theories to win out in the short term over more complex ones.

LIKELIHOOD OF A GRAPH I

Fix a graph G. Then any distribution of the form

$$\mathbb{P}(X_1 = x_1, \ldots, X_k = x_k) = \prod_{j=1}^{k} \mathbb{P}(X_j = x_j \,|\, X_i = x_i, i \in \mathrm{Pa}(X_j)).$$

will be called a **model in** G.

Lemma. If edges in G' are a strict subset of those in G, then the models of G' are a strict subset of those of G.

Corollary. The best fitting distribution in G is never worse than that in G'.

Are samples from $\texttt{QUERY}(\texttt{RPD}(n+1, 18), \texttt{checkAllSymptoms})$ more likely to come from G than G'?

LIKELIHOOD OF A GRAPH II

Key observations:

- posterior probability of a particular graph G does not reflect the best-fitting model in G, but rather reflects the *average* ability of models in G to explain the historical data.
- average is over $\mathrm{Uniform}[0,1]$ distribution of $p_{j|v}$.
- if a spurious edge exists in a graph G', a typical distribution from G' is less likely to explain the data than a typical distribution from the graph with that edge removed.

LIKELIHOOD OF A GRAPH III

In order to characterize the posterior distribution of the graph, we must identify the likelihood that a sample from the prior program is accepted given a particular graph G.

Every time the pattern $\{X_i = v_i, i \in \mathrm{Pa}(X_j)\}$ arises in historical data, the generative process produces the historical value X_j with probability $p_{j|v}$ if $X_j = 1$ and $1 - p_{j|v}$ if $X_j = 0$.

Conditional on the $p_{j|v}$*'s*, the probability that the historical data is reproduced is

$$\prod_{j=1}^{D} \prod_{v} p_{j|v}^{k_{j|v}} (1 - p_{j|v})^{n_{j|v} - k_{j|v}},$$

where v ranges over the possible $\{0, 1\}$ assignments to $\mathrm{Pa}(X_j)$ and $k_{j|v}$ and $n_{j|v}$ are defined as above.

LIKELIHOOD OF A GRAPH IV

However, we don't know $p_{j|v}$, and so the likelihood of the graph G is

$$\begin{aligned}\mathrm{score}(G) &= E\left[\prod_{j=1}^{D}\prod_{v} p_{j|v}^{k_{j|v}}(1-p_{j|v})^{n_{j|v}-k_{j|v}}\right] \\ &= \prod_{j=1}^{D}\prod_{v}(n_{j|v}+1)^{-1}\binom{n_{j|v}}{k_{j|v}}^{-1}\end{aligned}$$

where E takes expectations with respect to $\mathrm{Uniform}[0,1]$.

Because the graph G was chosen uniformly at random, it follows that the posterior probability of a particular graph G is proportional to $\mathrm{score}(G)$.

GRAPH POSTERIOR

We can study the preference for one graph G over another G' by studying the ratio of their scores:

$$\frac{\mathrm{score}(G)}{\mathrm{score}(G')}.$$

This score ratio is known as the *Bayes factor*, which I.J. Good termed the *Bayes–Jeffreys–Turing factor* [Good, 1968, 1975], and which Turing himself called the *factor in favor of a hypothesis* (see [Good, 1968], [Zabell, 2012, §1.4], and [Turing, 2012]). Its logarithm is sometimes known as the *weight of evidence* [Good, 1968].

The form of the score is a product over the local structure of the graph, thus the Bayes factor will depend only on the contributions from those parts of the graphs G and G' that differ.

Simple special case of graph posterior, Part I

Consider the following simplified scenario, which captures several features of learning structure from data: Fix two graphs, G and G', over the same collection of random variables, but assume that in G, two of these random variables, X and Y, have no parents and are thus independent, and in G' there is an edge from X to Y, and so they are almost surely dependent.

The Bayes factor in factor of independence is

$$\frac{(n_1+1)(n_0+1)}{(n+1)} \frac{\binom{n_1}{k_1}\binom{n_0}{k_0}}{\binom{n}{k}}, \tag{12}$$

where

- n counts the total number of observations;
- k counts $Y=1$;
- n_1 counts $X=1$;
- k_1 counts $X=1$ and $Y=1$;
- n_0 counts $X=0$; and
- k_0 counts $X=0$ and $Y=1$.

SIMPLE SPECIAL CASE OF GRAPH POSTERIOR, PART II

First consider the case where G' is the true underlying graph, i.e., when Y is indeed dependent on X.

By the law of large numbers, and Stirling's approximation, we can reason that the evidence for G' accumulates rapidly, satisfying

$$\log \frac{\mathrm{score}(G)}{\mathrm{score}(G')} \sim -C \cdot n, \quad \text{a.s.},$$

for some constant $C > 0$ that depends only on the joint distribution of (X, Y).

Simple special case of graph posterior, Part III

Now consider that G is the true underlying graph, i.e., X, Y are independent.

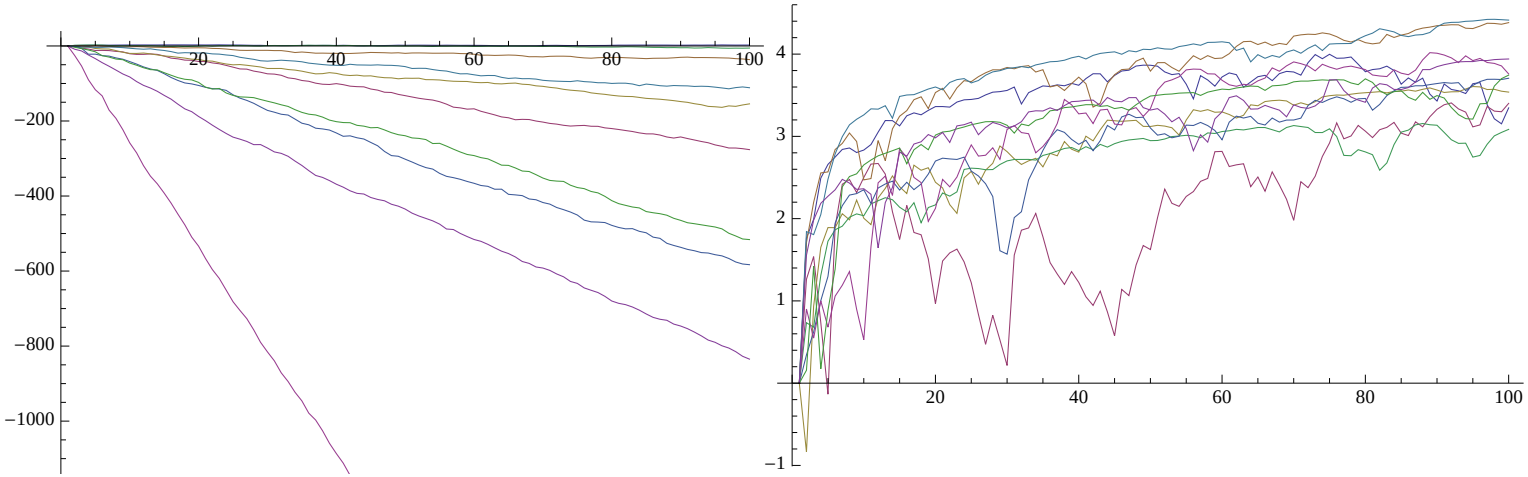
Using a similar approach, we have

$$\log \frac{\text{score}(G)}{\text{score}(G')} \sim \frac{1}{2} \log n, \quad \text{a.s.}$$

and thus evidence accumulates much more slowly.

The following plots show typical the evolution of the Bayes factors under G' and under G. Evidence accumulates much more rapidly for G'.

SIMPLE SPECIAL CASE OF GRAPH POSTERIOR, PART IV



Weight of evidence for dependence versus independence (positive values support independence) of a sequence of pairs of random variables sampled from $\texttt{RPD}(n, 2)$.

(left) When presented with data from a distribution where (X, Y) are indeed dependent, the weight of evidence rapidly accumulates for the dependent model, at an asymptotically linear rate in the amount of data.

(right) When presented with data from a distribution where (X, Y) are independent, the weight of evidence slowly accumulates for the independent model, at an asymptotic rate that is logarithmic in the amount of data.

Note that the dependent model can imitate the independent model, but, on average over random parameterizations of the conditional probability mass functions, the dependent model is worse at modeling independent data.

Simple special case of graph posterior, Part V

Some observations

- In both cases, evidence accumulates for the correct model. In fact, it can be shown that the expected weight of evidence is always non-negative for the true hypothesis, a result due to Turing himself [Good, 1991, p. 93].
- the prior distributions on the $p_{j|v}$ are fixed and do not vary with the amount of data, thus the weight of evidence will eventually eclipse any prior information and determine the posterior probability
- however, evidence accumulates rapidly for dependence and much more slowly for independence: should choose prior distribution on graph to reflect this imbalance, preferring graphs with fewer edges *a priori*.

Bayes' Occam's razor

The following two statements may seem contradictory

- When X and Y are independent, evidence stochastically accumulates for the simpler graph over the more complex graph
- There is, with high probability, always a parameterization of the more complex graph that assigns higher likelihood to the data than any parametrization of the simpler graph.

Bayes' Occam's razor [MacKay, 2003, Ch. 28]

The natural way in which hierarchical models choose explanations of the data with intermediate complexity, avoiding overfitting.

- many degrees of modeling freedom, then each configuration must be assigned, on average, less probability than it would under a simpler model with fewer degrees of freedom.
- graph with additional edges has more degrees of freedom

CHOOSING MODELS USING DATA

Which model should we use to diagnose: `diseasesAndSymptoms`, `allDiseasesAndSymptoms` or `RPD`?

We can use the same Bayes' Occam's Razor perspective:

- `RPD` model has many more degrees of freedom than both `diseasesAndSymptoms` and `allDiseasesAndSymptoms`.
  - Given enough data, `RPD` can fit any distribution on a finite collection of binary variables, as opposed to `allDiseasesAndSymptoms`, which cannot because it makes strong and immutable assumptions.
- With only a small amount of training data, `RPD` model expected to have high posterior uncertainty.
  - One would expect better predictions from `allDiseasesAndSymptoms` than `RPD`, if both were fed data generated by `diseasesAndSymptoms`.

State of the art

- There is a challenge bridging the gap between `allDiseasesAndSymptoms` and `RPD`.
- A lot of focus in Bayesian statistics is on building scalable approximations to `QUERY`.
- Key family of algorithms are so-called “variational approximations”.
- There’s also active research on Monte Carlo approximations to `QUERY`.
- Bespoke hand-crafted models in machine learning are being replaced by hybrid deep learning ones.
- Probabilistic programming and differentiable programming frameworks massively accelerate certain approaches.
- There’s also interest in bridging the frequentist–Bayesian divide.

F. R. Bach and M. I. Jordan. Learning graphical models with Mercer kernels. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, pages 1009–1016. The MIT Press, Cambridge, MA, 2003.

A. R. Barron. Information-theoretic characterization of Bayes performance and the choice of priors in parametric and nonparametric problems. In J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, editors, *Bayesian Statistics 6: Proceedings of the Sixth Valencia International Meeting*, pages 27–52, 1998.

V. Chandrasekaran, N. Srebro, and P. Harsha. Complexity of inference in graphical models. In *Proceedings of the Twenty Fourth Conference on Uncertainty in Artificial Intelligence (UAI 2008)*, pages 70–78, Corvalis, Oregon, 2008. AUAI Press.

G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990.

P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153, 1993. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/0004-3702(93)90036-B.

P. Dagum, R. Karp, M. Luby, and S. Ross. An optimal algorithm for Monte Carlo estimation. *SIAM Journal on Computing*, 29(5):1484–1496, 2000. doi: 10.1137/S0097539797315306. URL http://epubs.siam.org/doi/abs/10.1137/S0097539797315306.

M. H. DeGroot. *Optimal Statistical Decisions*. Wiley Classics Library. Wiley, 2005. ISBN 9780471726142. URL http://books.google.co.uk/books?id=dtVieJ245z0C.

I. J. Good. Corroboration, explanation, evolving probability, simplicity and a sharpened razor. *The British Journal for the Philosophy of Science*, 19(2): 123–143, 1968.

I. J. Good. Explicativity, corroboration, and the relative odds of hypotheses. *Synthese*, 30(1):39–73, 1975.

I. J. Good. Weight of evidence and the Bayesian likelihood ratio. In C. G. G. Aitken and D. A. Stoney, editors, *The Use Of Statistics In Forensic Science*. Ellis Horwood, Chichester, 1991.

N. D. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: A language for generative models. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI 2008)*, pages 220–229, Corvalis, Oregon, 2008. AUAI Press.

T. L. Griffiths and J. B. Tenenbaum. Structure and strength in causal induction. *Cognitive Psychology*, 51(4):334–384, 2005. ISSN 0010-0285. doi: 10.1016/j.cogpsych.2005.05.004. URL http://www.sciencedirect.com/science/article/pii/S0010028505000459.

T. L. Griffiths and J. B. Tenenbaum. Optimal predictions in everyday cognition. *Psychological Science*, 17(9):767–773, 2006. URL http://web.mit.edu/cocosci/Papers/Griffiths-Tenenbaum-PsychSci06.pdf.

T. L. Griffiths and J. B. Tenenbaum. Theory-based causal induction. *Psychological Review*, 116(4):661–716, 2009. URL http://cocosci.berkeley.edu/tom/papers/tbci.pdf.

T. L. Griffiths, C. Kemp, and J. B. Tenenbaum. Bayesian models of cognition. In *Cambridge Handbook of Computational Cognitive Modeling*. Cambridge University Press, 2008. URL http://cocosci.berkeley.edu/tom/papers/bayeschapter.pdf.

O. Kallenberg. *Foundations of modern probability*. Probability and its Applications. Springer, New York, 2nd edition, 2002. ISBN 0-387-95313-2.

C. Kemp and J. B. Tenenbaum. The discovery of structural form. *Proceedings of the National Academy of Sciences*, 105(31):10687–10692, 2008. URL http://www.psy.cmu.edu/~ckemp/papers/kempt08.pdf.

C. Kemp, P. Shafto, A. Berke, and J. B. Tenenbaum. Combining causal and similarity-based reasoning. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19 (NIPS 2006)*, pages 681–688. The MIT Press, Cambridge, MA, 2007.

R. D. Luce. *Individual Choice Behavior*. John Wiley, New York, 1959.

R. D. Luce. The choice axiom after twenty years. *Journal of Mathematical Psychology*, 15(3):215–233, 1977.

D. J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, Cambridge, UK, 2003.

V. K. Mansinghka. *Natively Probabilistic Computation*. PhD thesis, Massachusetts Institute of Technology, 2009.

V. K. Mansinghka, C. Kemp, J. B. Tenenbaum, and T. L. Griffiths. Structured priors for structure learning. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence (UAI 2006)*, pages 324–331, Arlington, Virginia, 2006. AUAI Press. URL http://cocosci.berkeley.edu/tom/papers/structure.pdf.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, 1988.

J. Pearl. Graphical models for probabilistic and causal reasoning. In A. B. Tucker, editor, *Computer Science Handbook*. CRC Press, 2nd edition, 2004.

M. A. Shwe, B. Middleton, D. E. Heckerman, M. Henrion, E. J. Horvitz, H. P. Lehmann, and G. F. Cooper. Probabilistic diagnosis using a reformulation of the INTERNIST-1/QMR knowledge base. *Methods of Information in Medicine*, 30:241–255, 1991.

J. B. Tenenbaum, T. L. Griffiths, and C. Kemp. Theory-based Bayesian models of inductive learning and reasoning. *Trends in Cognitive Sciences*, 10:309–318, 2006. doi: $10.1016/\mathrm{j.tics.2006.05.009}$.

M. Toussaint, S. Harmeling, and A. Storkey. Probabilistic inference for solving (PO)MDPs. Technical Report EDI-INF-RR-0934, University of Edinburgh, School of Informatics, 2006.

A. M. Turing. *The Applications of Probability to Cryptography, c. 1941*. UK National Archives, HW 25/37. 2012.

S. L. Zabell. Commentary on Alan M. Turing: The applications of probability to cryptography. *Cryptologia*, 36(3):191–214, 2012. doi: $10.1080/01611194.2012.697811$.