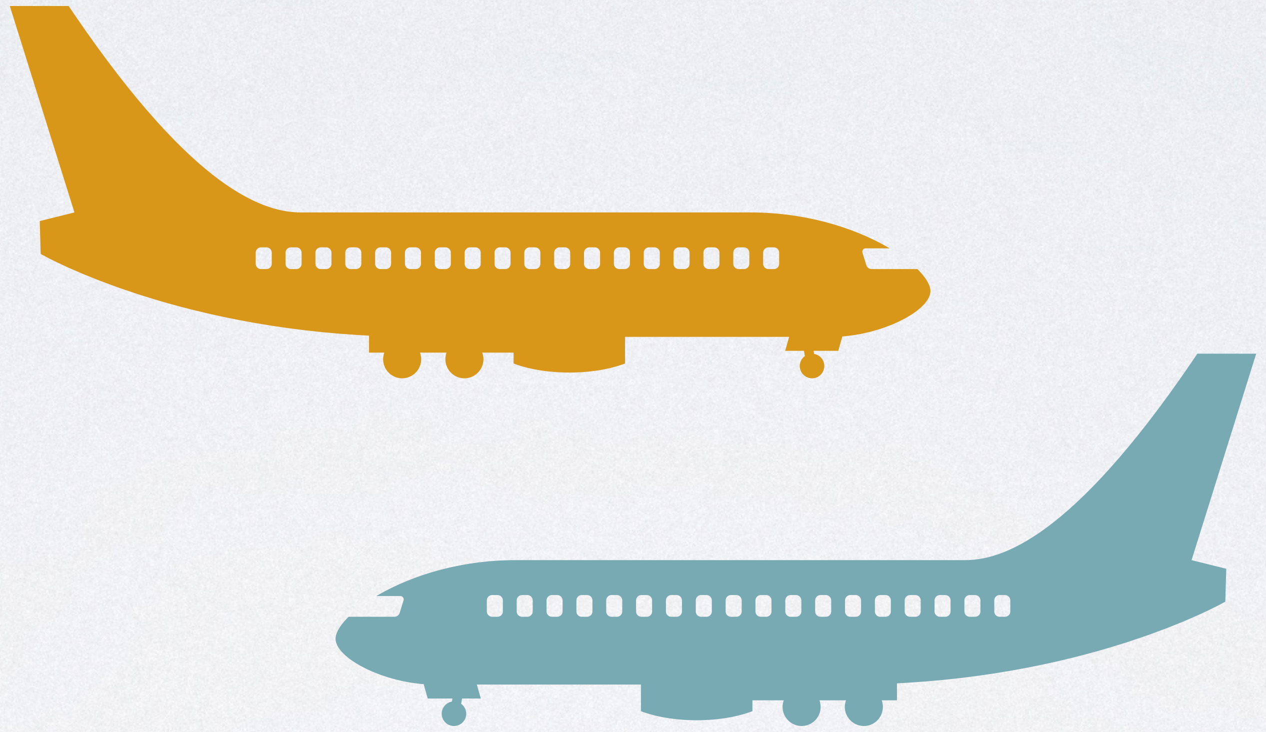
Algebraic Program Analysis of Probabilistic Programs

Di Wang
Peking University
wangdi95@pku.edu.cn
Jun 14, 2025

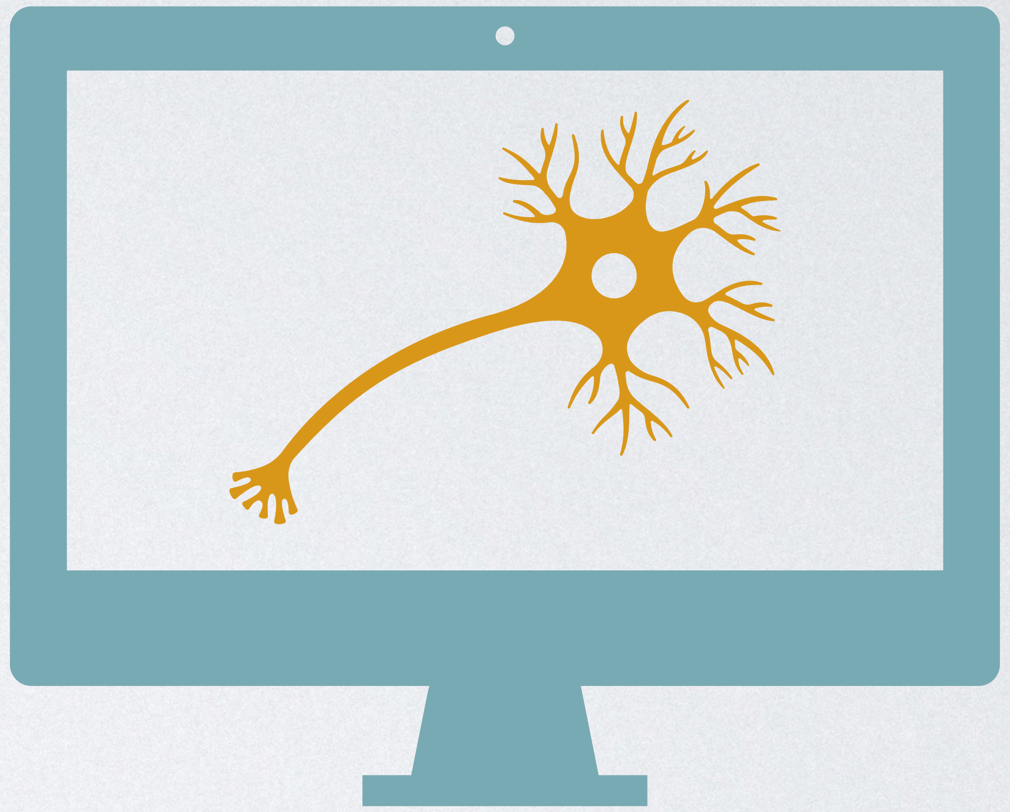
Joint Work with **Jan Hoffmann** and **Thomas Reps**

Probabilistic Systems are Becoming Pervasive

Randomized Algorithms
(improve efficiency)



Cyber-Physical Systems
(model uncertainty)



Artificial Intelligence
(describe statistical models)

Application: Randomized Quicksort

- Improve efficiency
- From $\Theta(n^2)$ to $\Theta(n \log n)$ (expected)
- Samplesort
  - Use >1 random samples as pivots

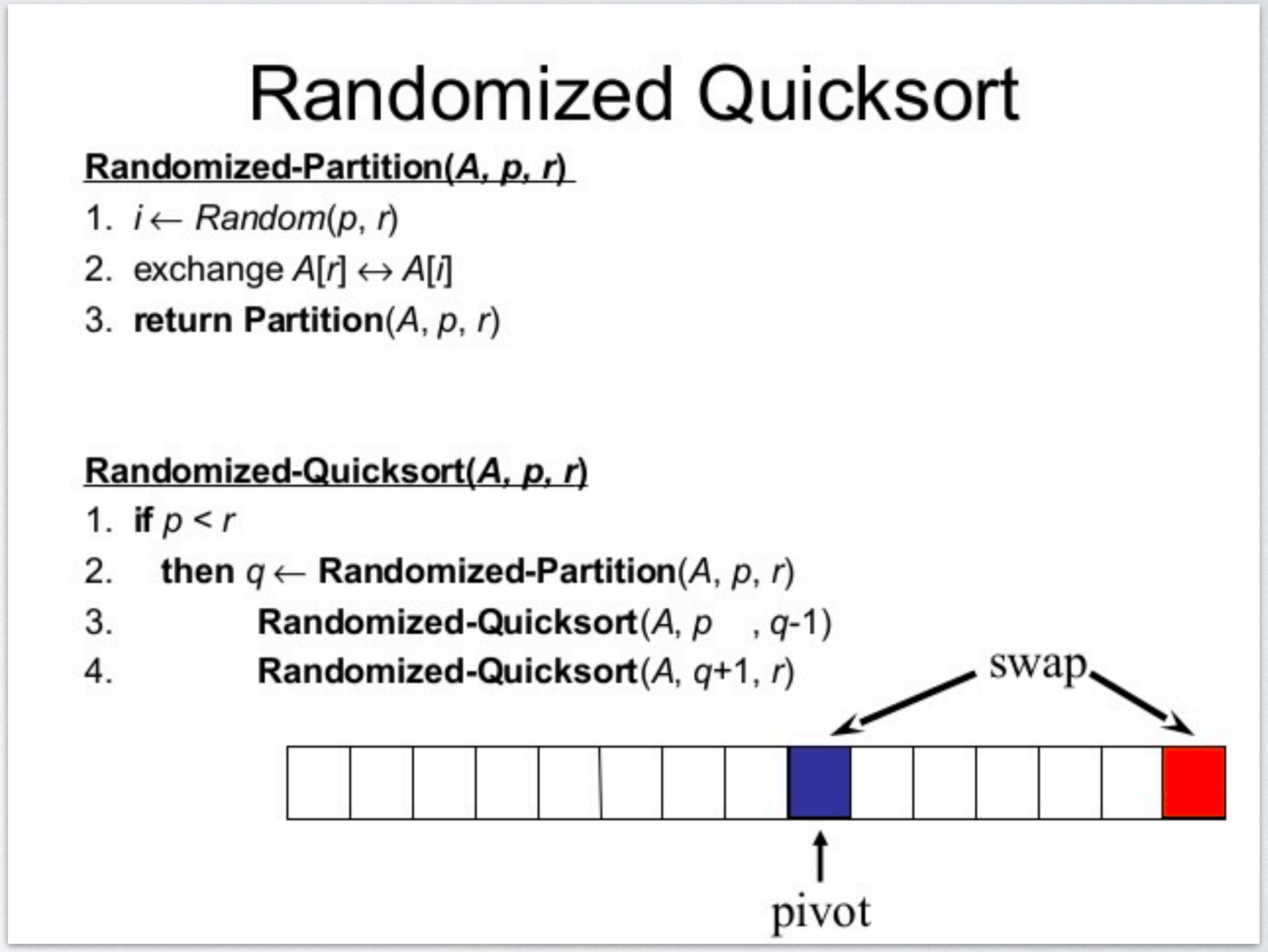


Image source: https://geekfactorial.blogspot.com/2016/08/randomized-quick-sort-algorithm.html.

Application: Airborne Collision Avoidance

- Model uncertainty
- Probabilistic dynamics
- Probabilistic sensors
- High-confidence system

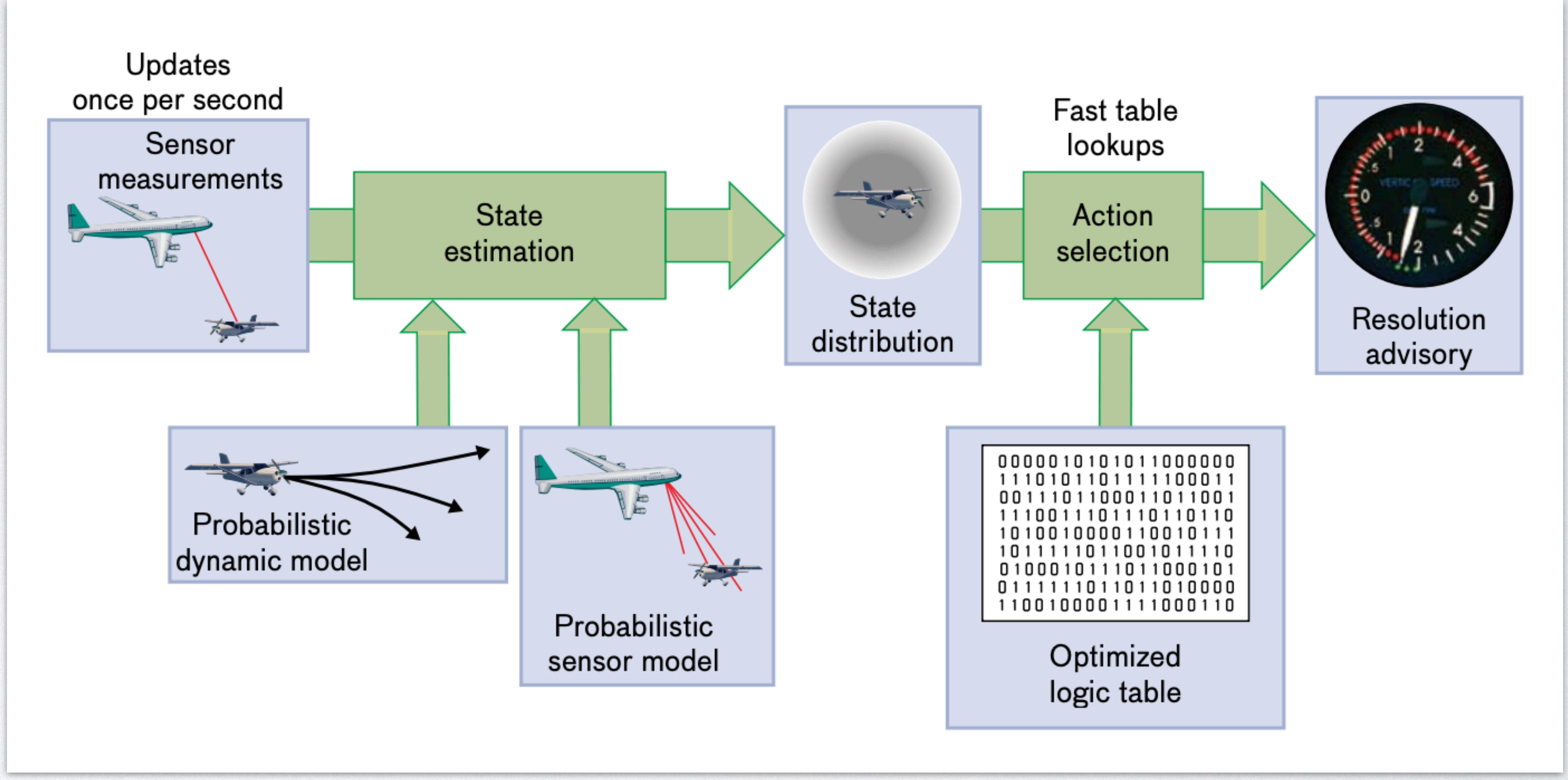
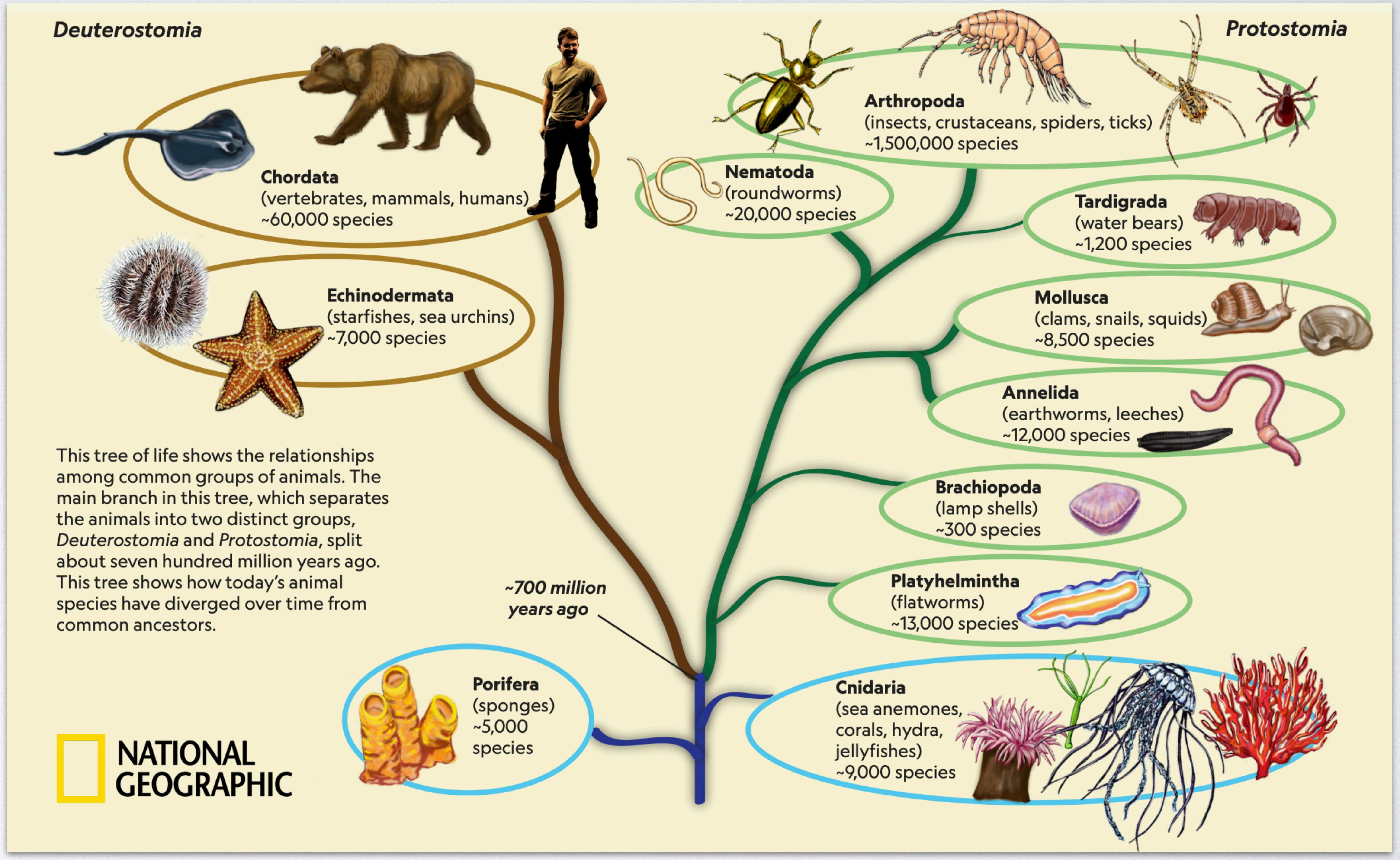


Image source: Kochenderfer, et al. "Next-Generation Airborne Collision Avoidance System." Lincoln Laboratory Journal (2012).

Application: Statistical Phylogenetics

- Describe statistical models
- Automated reasoning
- Apply Bayesian inference to infer evolutionary history
- Solve previously intractable problems



Ronquist, et al. "Universal probabilistic programming offers a powerful approach to statistical phylogenetics." Communications Biology (2021).
Image source: https://www.nationalgeographic.org/media/tree-life/.

Probabilistic Programs

Draw random **data** from distributions

Change **control-flow** at random

Image sources: https://www.libertystorch.info/2022/02/21/the-grab-bag/; https://www.stockvault.net/photo/179872/at-a-crossroads-decisions-and-choices-concept-with-large-arrow-signs.

Probabilistic Programs

- True randomness
- A distribution on execution paths
- Probabilistic nondeterminism

```
if
| prob(1/3) → choice ≔ 1
| prob(1/3) → choice ≔ 2
| prob(1/3) → choice ≔ 3
fi
```

Probabilistic Programs

- True randomness
- A distribution on execution paths
- Probabilistic nondeterminism

```
if
| prob(1/3) → choice ≔ 1
| prob(1/3) → choice ≔ 2
| prob(1/3) → choice ≔ 3
fi
```

```
choice :∈p (1 @ 1/3 | 2 @ 1/3 | 3 @ 1/3)
```

Demonic Programs

- Dijkstra's **Guarded Command Language** (GCL)
- A set of execution paths
- Demonic nondeterminism

```
if
| true → prize ≔ 1
| true → prize ≔ 2
| true → prize ≔ 3
fi
```

Demonic Programs

- Dijkstra's **Guarded Command Language** (GCL)
- A set of execution paths
- Demonic nondeterminism

```
if
| true → prize ≔ 1
| true → prize ≔ 2
| true → prize ≔ 3
fi
```

$$\boxed{\texttt{prize} :\in_d \{1,2,3\}}$$

Example: Monty Hall

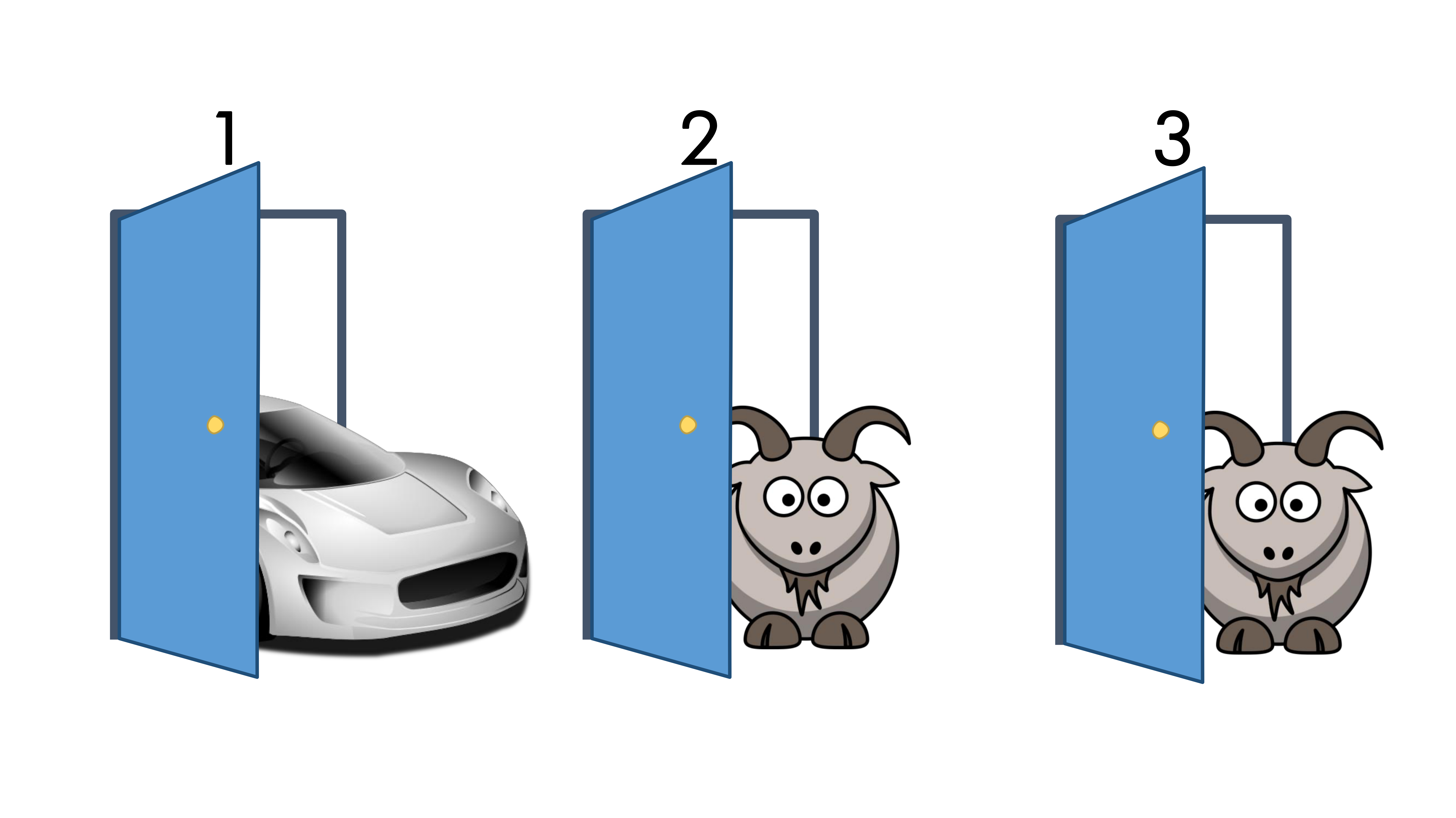


Image source: Maria Gorinova's Advances in Programming Languages (Guest Lecture) slides on Probabilistic Programming.

Example: Monty Hall

Example: Monty Hall

- McIver and Morgan's **probabilistic Guarded Command Language** (pGCL)
- Combine two forms of nondeterminism:
  - Probabilistic
  - Demonic

```
prize :∈d {1,2,3};
choice :∈p (1 @ 1/3 | 2 @ 1/3 | 3 @ 1/3);
host :∈d {1,2,3} \ {prize,choice};
if switch then
  choice :∈d {1,2,3} \ {choice,host}
fi
```

Example: Monty Hall

- McIver and Morgan's **probabilistic Guarded Command Language** (pGCL)
- Combine two forms of nondeterminism:
  - Probabilistic
  - Demonic

```
prize :∈d {1,2,3};
choice :∈p (1 @ 1/3 | 2 @ 1/3 | 3 @ 1/3);
host :∈d {1,2,3} \ {prize,choice};
if switch then
  choice :∈d {1,2,3} \ {choice,host}
fi
```

$\mathbb{P}(choice = prize) = ?$

Example: Monty Hall

- McIver and Morgan's **probabilistic Guarded Command Language** (pGCL)
- Combine two forms of nondeterminism:
  - Probabilistic
  - Demonic
- "Demons" minimize the probability

```
prize :∈d {1,2,3};
choice :∈p (1 @ 1/3 | 2 @ 1/3 | 3 @ 1/3);
host :∈d {1,2,3} \ {prize,choice};
if switch then
  choice :∈d {1,2,3} \ {choice,host}
fi
```

$$\mathbb{P}(choice = prize) = ?$$

Example: Failure Modeling

Example: Failure Modeling

- An example of **probabilistic modeling checking**
- Send c messages, each with a failure probability 0.1

```
fail := FALSE;
c :∈d {0,1,2};
while not(fail) and c > 0 do
  fail :∈p (TRUE @ 0.1 | FALSE @ 0.9 );
  c := c - 1
od
```

Example: Failure Modeling

- An example of **probabilistic modeling checking**
- Send c messages, each with a failure probability 0.1
- What is the probability of success?

```
fail := FALSE;
c :∈d {0,1,2};
while not(fail) and c > 0 do
  fail :∈p (TRUE @ 0.1 | FALSE @ 0.9 );
  c := c - 1
od
```

$\mathbb{P}(\mathit{fail} = \mathit{FALSE}) = ?$

Example: Abstraction

Example: Abstraction

- Program analysis introduces **abstraction**
- **Predicate Abstraction**
  - [c=0] is a Boolean variable

```
fail := FALSE;
[c=0] :∈d {TRUE,FALSE};
while not(fail) and not([c=0]) do
  fail :∈p (TRUE @ 0.1 | FALSE @ 0.9 );
  [c=0] :∈a {TRUE,FALSE}
od;
```

Example: Abstraction

- Program analysis introduces **abstraction**
- **Predicate Abstraction**
  - [c=0] is a Boolean variable

```
fail := FALSE;
[c=0] :∈d {TRUE,FALSE};
while not(fail) and not([c=0]) do
  fail :∈p (TRUE @ 0.1 | FALSE @ 0.9 );
  [c=0] :∈a {TRUE,FALSE}
od;
```

$$\mathbb{P}(\mathit{fail} = \mathit{FALSE}) = ?$$

Example: Abstraction

- Program analysis introduces **abstraction**
- **Predicate Abstraction**
  - [c=0] is a Boolean variable
- Abstraction nondeterminism
  - Maximize —> Upper bound
  - Minimize —> Lower bound

```
fail := FALSE;
[c=0] :∈d {TRUE,FALSE};
while not(fail) and not([c=0]) do
  fail :∈p (TRUE @ 0.1 | FALSE @ 0.9 );
  [c=0] :∈a {TRUE,FALSE}
od;
```

$\mathbb{P}(fail = FALSE) = ?$

How to automate such $\boxed{\text{quantitative}}$ reasoning about probabilistic programs?

How to automate such $\boxed{\text{quantitative}}$ reasoning about probabilistic programs?

Examples

What is the probability that an assertion holds?

How to automate such $\boxed{\text{quantitative}}$ reasoning about probabilistic programs?

Examples

What is the probability that an assertion holds?

What is the expected value of an expression?

How to automate such $\boxed{\text{quantitative}}$ reasoning about probabilistic programs?

Examples

What is the probability that an assertion holds?

What is the expected value of an expression?

What is the expected time complexity of a program?

Challenge I:
How to support multiple confluence operations?

… $: \in_p$ …

… $: \in_d$ …

… $: \in_a$ …

Semantic Algebras

- **Kleene Algebras**: A **compositional** and **flexible** framework for program semantics

Program Construct	Algebraic Representation
A program S	An interpretation $\mathbb{S}$ of S into the algebra
Branching between A and B	$\mathbb{A} \oplus \mathbb{B}$
Sequencing of A and B	$\mathbb{A} \otimes \mathbb{B}$
Iteration (i.e., loop) of A	$\mathbb{A}^*$
"`abort`", "`skip`"	$\underline{0}, \underline{1}$

Do Kleene Algebras Suffice?

Do Kleene Algebras Suffice?

```
if
| true → x ≔ 1
| true → x ≔ 2
| true → x ≔ 3
fi
```

Do Kleene Algebras Suffice?

```
if
| true → x := 1
| true → x := 2
| true → x := 3
fi
```

$$
\begin{aligned}
&([\mathbf{true}] \otimes \mathsf{x} \coloneqq 1) \\
\oplus\ &([\mathbf{true}] \otimes \mathsf{x} \coloneqq 2) \\
\oplus\ &([\mathbf{true}] \otimes \mathsf{x} \coloneqq 3)
\end{aligned}
$$

Do Kleene Algebras Suffice?

```
if
| true → x ≔ 1
| true → x ≔ 2
| true → x ≔ 3
fi
```

$$
\begin{aligned}
&([\mathbf{true}] \otimes \mathrm{x} \coloneqq 1) \\
\oplus\ &([\mathbf{true}] \otimes \mathrm{x} \coloneqq 2) \\
\oplus\ &([\mathbf{true}] \otimes \mathrm{x} \coloneqq 3)
\end{aligned}
$$

```
if
| prob(1/3) → x ≔ 1
| prob(1/3) → x ≔ 2
| prob(1/3) → x ≔ 3
fi
```

Do Kleene Algebras Suffice?

```
if
| true → x ≔ 1
| true → x ≔ 2
| true → x ≔ 3
fi
```

$$
\begin{aligned}
&([\mathbf{true}] \otimes x \coloneqq 1)\\
\oplus\ &([\mathbf{true}] \otimes x \coloneqq 2)\\
\oplus\ &([\mathbf{true}] \otimes x \coloneqq 3)
\end{aligned}
$$

```
if
| prob(1/3) → x ≔ 1
| prob(1/3) → x ≔ 2
| prob(1/3) → x ≔ 3
fi
```

$$
\begin{aligned}
&([\mathbf{prob}(1/3)] \otimes x \coloneqq 1)\\
\oplus\ &([\mathbf{prob}(1/3)] \otimes x \coloneqq 2)\\
\oplus\ &([\mathbf{prob}(1/3)] \otimes x \coloneqq 3)
\end{aligned}
$$

Do Kleene Algebras Suffice?

```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```

Do Kleene Algebras Suffice?

```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```

$$(([\mathbf{prob}(1/2)] \otimes x \coloneqq 1) \oplus ([\mathbf{prob}(1/2)] \otimes x \coloneqq 2))$$
$$\oplus\ (([\mathbf{prob}(1/2)] \otimes x \coloneqq 3) \oplus ([\mathbf{prob}(1/2)] \otimes x \coloneqq 4))$$

Do Kleene Algebras Suffice?

```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```

$$(([\mathbf{prob}(1/2)] \otimes x \coloneqq 1) \oplus ([\mathbf{prob}(1/2)] \otimes x \coloneqq 2))$$
$$\oplus\ (([\mathbf{prob}(1/2)] \otimes x \coloneqq 3) \oplus ([\mathbf{prob}(1/2)] \otimes x \coloneqq 4))$$

$$= ([\mathbf{prob}(1/2)] \otimes x \coloneqq 1)$$
$$\oplus\ ([\mathbf{prob}(1/2)] \otimes x \coloneqq 2)$$
$$\oplus\ ([\mathbf{prob}(1/2)] \otimes x \coloneqq 3)$$
$$\oplus\ ([\mathbf{prob}(1/2)] \otimes x \coloneqq 4)$$

Do Kleene Algebras Suffice?

```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```

$$((\mathbf{[prob(1/2)]} \otimes x := 1) \oplus (\mathbf{[prob(1/2)]} \otimes x := 2))$$
$$\oplus ((\mathbf{[prob(1/2)]} \otimes x := 3) \oplus (\mathbf{[prob(1/2)]} \otimes x := 4))$$

$$= (\mathbf{[prob(1/2)]} \otimes x := 1)$$
$$\oplus (\mathbf{[prob(1/2)]} \otimes x := 2)$$
$$\oplus (\mathbf{[prob(1/2)]} \otimes x := 3)$$
$$\oplus (\mathbf{[prob(1/2)]} \otimes x := 4)$$

Probabilities sum up to 2!

Our Approach: Markov Algebras

- Key observation: Probabilistic programs have **multiple confluence operations**

$$\left\langle M, \sqsubseteq, \otimes, {}_{\phi}\oplus, \sqcap, \underline{0}, \underline{1} \right\rangle$$

Our Approach: Markov Algebras

- Key observation: Probabilistic programs have **multiple confluence operations**

$$\left\langle \underline{M, \sqsubseteq}, \otimes, {}_{\phi}\oplus, \sqcap, \underline{0}, \underline{1} \right\rangle$$

Program denotations form a CPO

Our Approach: Markov Algebras

- Key observation: Probabilistic programs have **multiple confluence operations**

$$\left\langle M, \sqsubseteq, \otimes, {}_{\phi}\oplus, \sqcap, \underline{0}, \underline{1} \right\rangle$$

Program denotations form a CPO

Sequencing, branching, and nondeterministic-choice

Our Approach: Markov Algebras

- Key observation: Probabilistic programs have **multiple confluence operations**

$$\left\langle \underbrace{M, \sqsubseteq}_{\text{Program denotations form a CPO}}, \underbrace{\otimes, {}_{\phi}\oplus, \sqcap}_{\text{Sequencing, branching, and nondeterministic-choice}}, \underline{0}, \underline{1} \right\rangle$$

Program denotations form a CPO

Sequencing, branching, and nondeterministic-choice

Easy to extend with more confluence operations!

Our Approach: Markov Algebras

- Key observation: Probabilistic programs have **multiple confluence operations**

$$\left\langle \underline{M, \sqsubseteq}, \underline{\otimes, {}_{\phi}\oplus, \sqcap}, \underline{\underline{0}, \underline{1}} \right\rangle$$

Program denotations form a CPO

Sequencing, branching, and nondeterministic-choice

$\underline{0}$ interprets **`abort`**
$\underline{1}$ interprets **`skip`**

Easy to extend with more confluence operations!

Our Approach: Markov Algebras

- Key observation: Probabilistic programs have **multiple confluence operations**

$$\left\langle \underline{M, \sqsubseteq}, \underline{\otimes, {}_{\phi}\oplus, \sqcap}, \underline{0}, \underline{1} \right\rangle$$

Program denotations form a CPO

Sequencing, branching, and nondeterministic-choice

Easy to extend with more confluence operations!

Our Approach: Markov Algebras

- Key observation: Probabilistic programs have **multiple confluence operations**

$$\langle \underbrace{M, \sqsubseteq}, \underbrace{\otimes, {}_{\phi}\oplus, \sqcap}, \underline{0}, \underline{1} \rangle$$

Program denotations form a CPO

Sequencing, branching, and nondeterministic-choice

Easy to extend with more confluence operations!

$$(a \otimes b) \otimes c = a \otimes (b \otimes c)$$

$$a \otimes \underline{1} = \underline{1} \otimes a = a$$

$$a\, {}_{\phi}\oplus\, b = b\, {}_{\overline{\phi}}\oplus\, a$$

$$a \sqcap a = a$$

...

Markov Algebras Suffice!

Markov Algebras Suffice!

```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```

Markov Algebras Suffice!

```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```

$$(x \coloneqq 1 \,{}_{1/2}\!\oplus\, x \coloneqq 2) \sqcap (x \coloneqq 3 \,{}_{1/2}\!\oplus\, x \coloneqq 4)$$

Markov Algebras Suffice!

```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```

$$(x := 1 \,{}_{1/2}\!\oplus\, x := 2) \sqcap (x := 3 \,{}_{1/2}\!\oplus\, x := 4)$$

```
while x>0 do
  x :∈p (x+1 @ 1/2 | x-1 @ 1/2)
od
```

Markov Algebras Suffice!

```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```

$$(x := 1\ {}_{1/2}\!\oplus\ x := 2) \sqcap (x := 3\ {}_{1/2}\!\oplus\ x := 4)$$

```
while x>0 do
  x :∈p (x+1 @ 1/2 | x-1 @ 1/2)
od
```

$$\mu S\,.\,((x := x+1\ {}_{1/2}\!\oplus\ x := x\text{-}1) \otimes S)\ {}_{[x>0]}\!\oplus\ \mathbf{skip}$$

Markov Algebras Suffice!

```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```

$$(x := 1 \,{}_{1/2}\!\oplus x := 2) \sqcap (x := 3 \,{}_{1/2}\!\oplus x := 4)$$

```
while x>0 do
  x :∈p (x+1 @ 1/2 | x-1 @ 1/2)
od
```

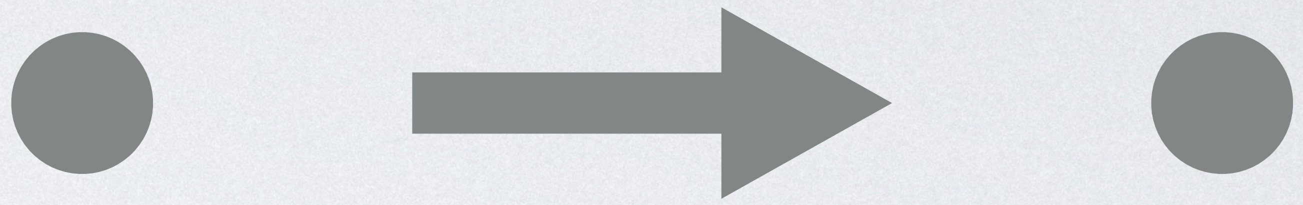
$$\mu S\,.\,((x := x{+}1 \,{}_{1/2}\!\oplus x := x{-}1) \otimes S)\,{}_{[x>0]}\!\oplus \mathbf{skip}$$

Recursive Program Scheme

Flexibility: Different Semantics

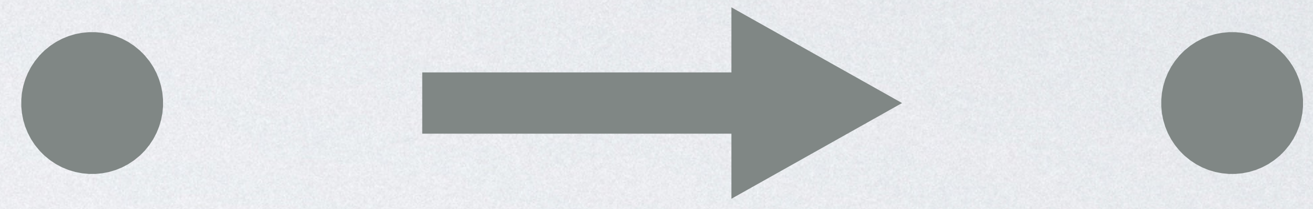
Flexibility: Different Semantics

Standard: $State \to State$



Flexibility: Different Semantics

Standard: $State \rightarrow State$



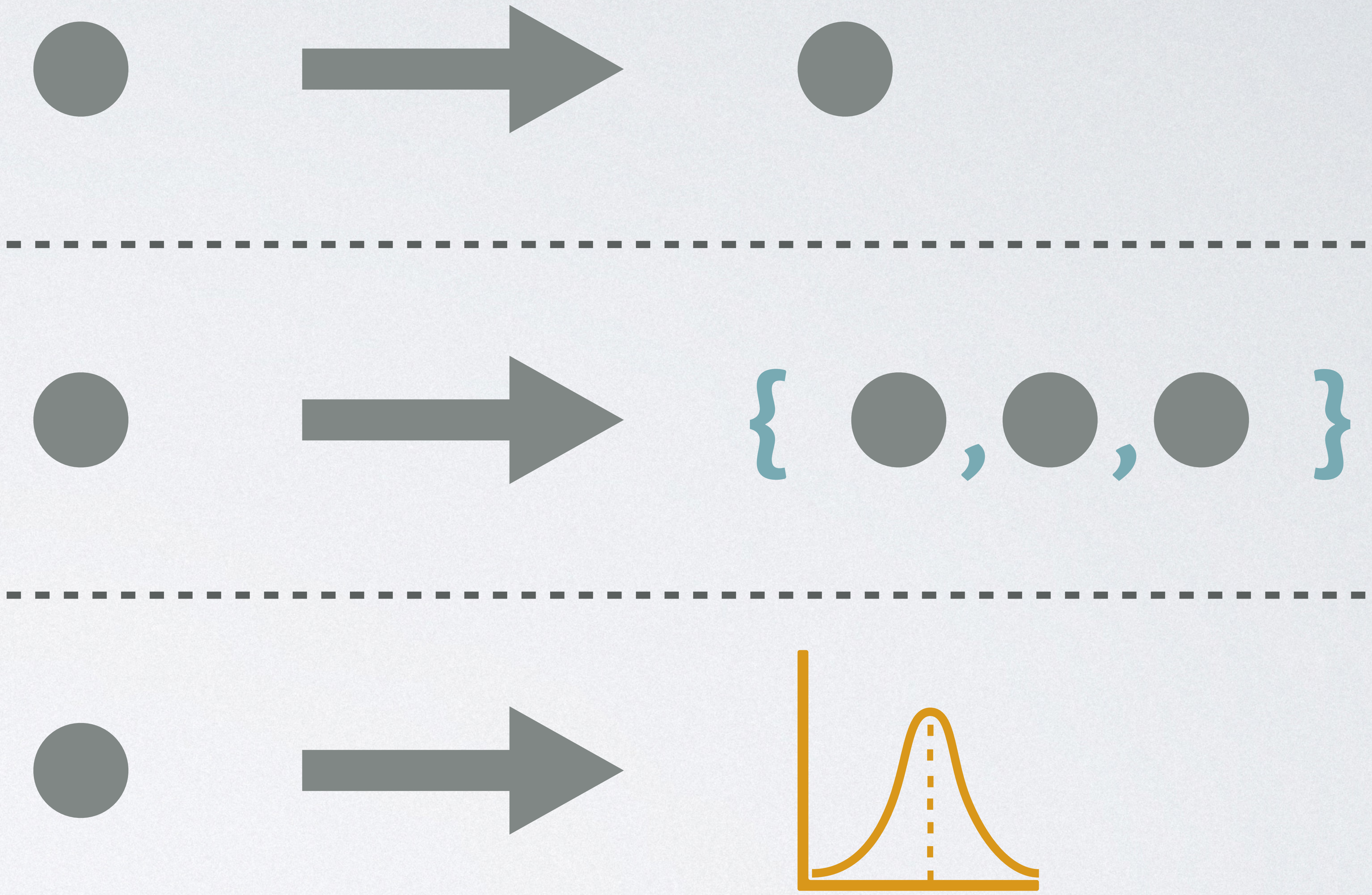
GCL: $State \rightarrow \wp(State)$

Flexibility: Different Semantics

Standard: $\mathit{State} \rightarrow \mathit{State}$

GCL: $\mathit{State} \rightarrow \wp(\mathit{State})$

Probabilistic: $\mathit{State} \rightarrow \mathbb{D}(\mathit{State})$



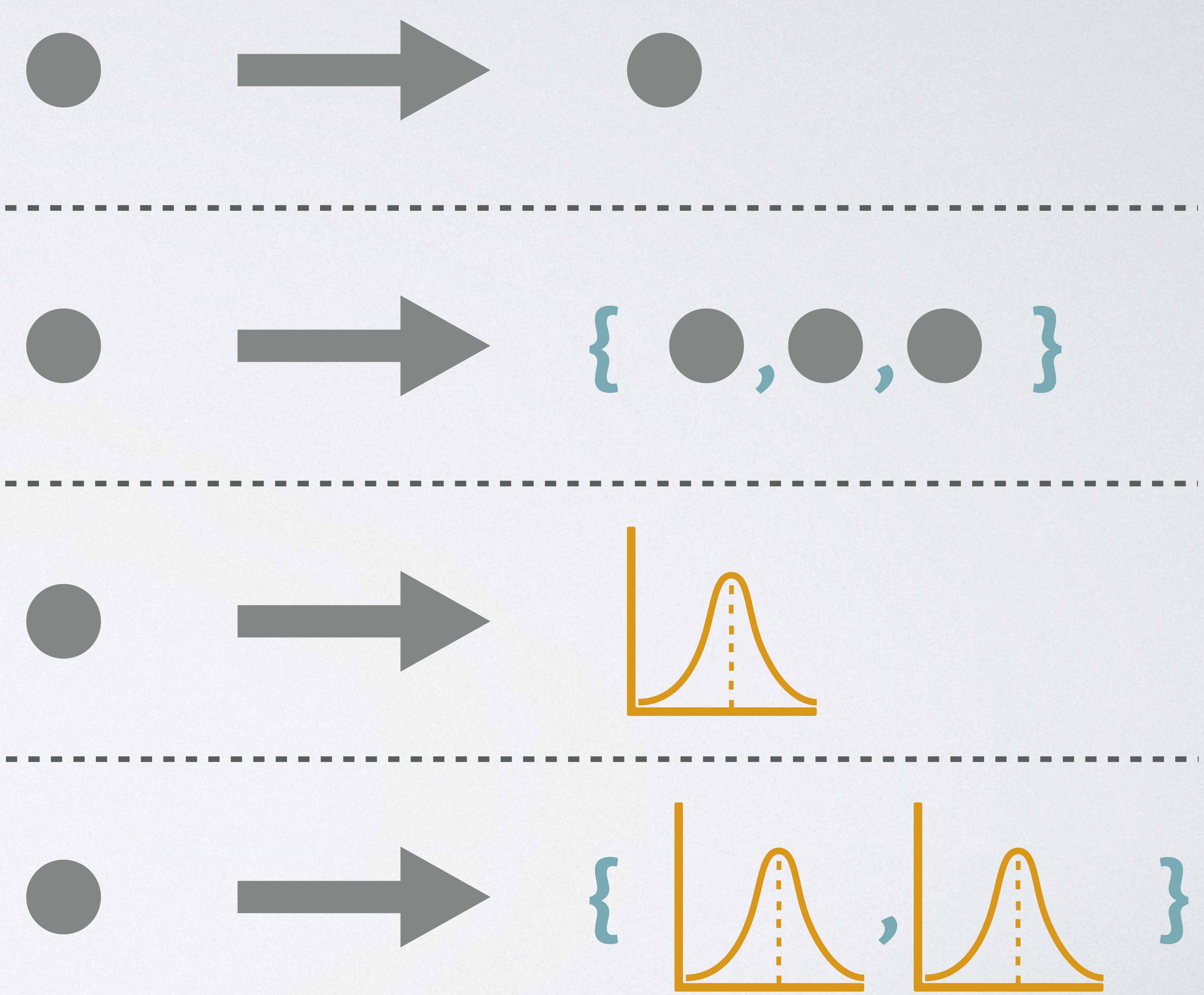
Flexibility: Different Semantics

Standard: $\mathit{State} \to \mathit{State}$

GCL: $\mathit{State} \to \wp(\mathit{State})$

Probabilistic: $\mathit{State} \to \mathbb{D}(\mathit{State})$

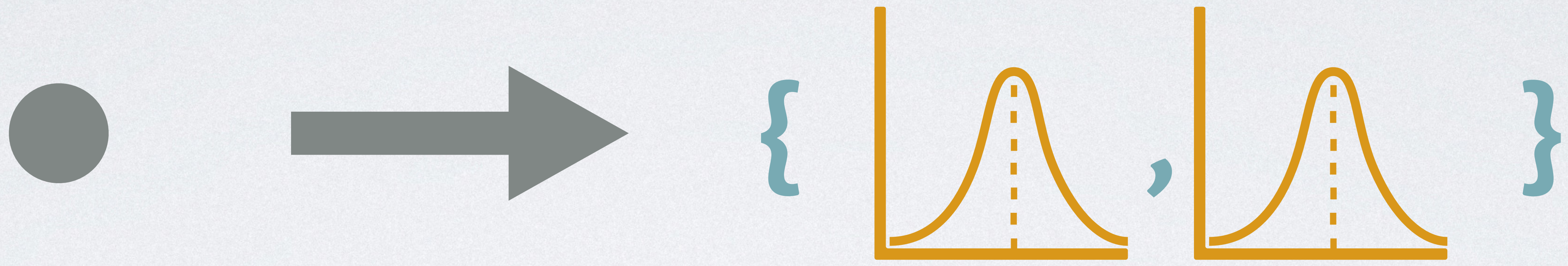
pGCL: $\mathit{State} \to \wp(\mathbb{D}(\mathit{State}))$



Flexibility: Different Models for Nondeterminism

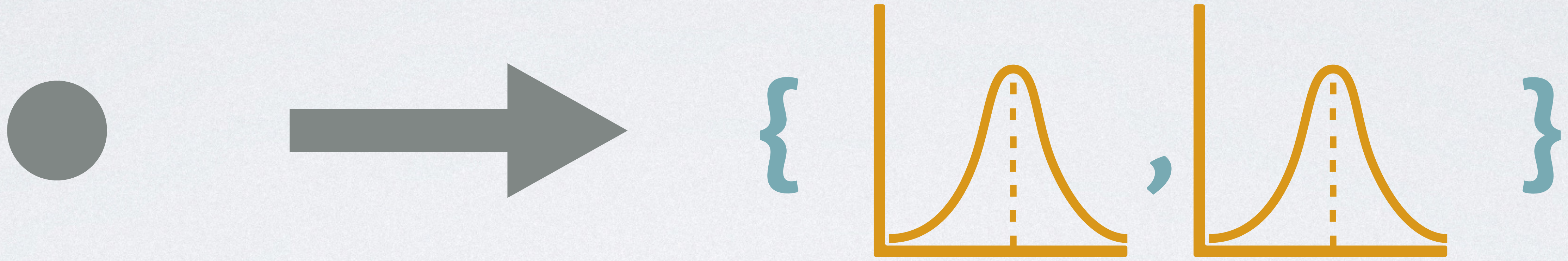
Flexibility: Different Models for Nondeterminism

- pGCL: $State \rightarrow \wp(\mathbb{D}(State))$

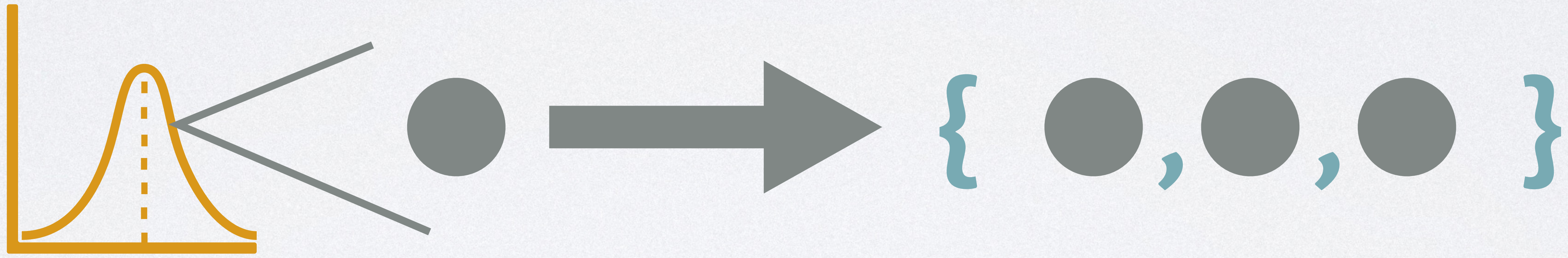


Flexibility: Different Models for Nondeterminism

- pGCL: $\mathit{State} \to \wp(\mathbb{D}(\mathit{State}))$

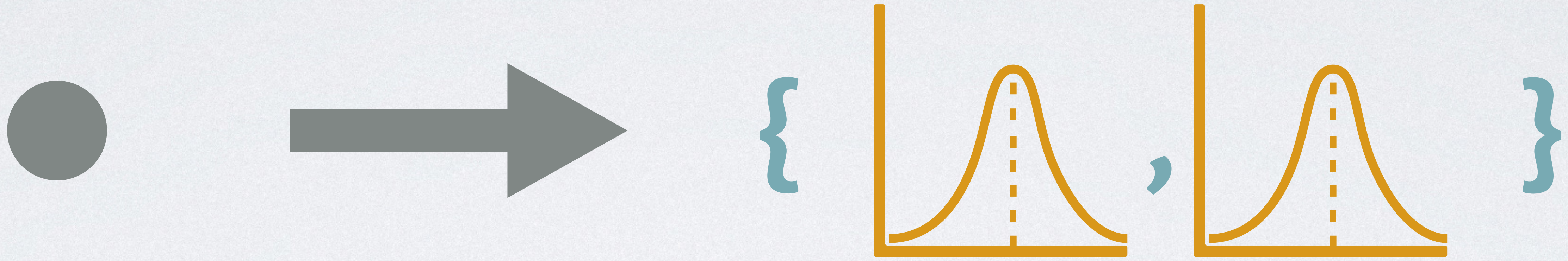


- Cousot's **Probabilistic Abstract Interpretation** (PAI): $\mathbb{D}(\mathit{State} \to \wp(\mathit{State}))$

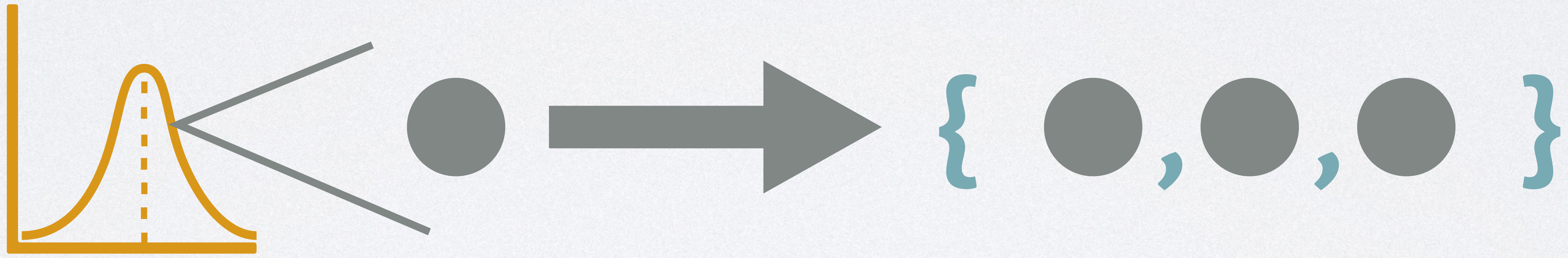


Flexibility: Different Models for Nondeterminism

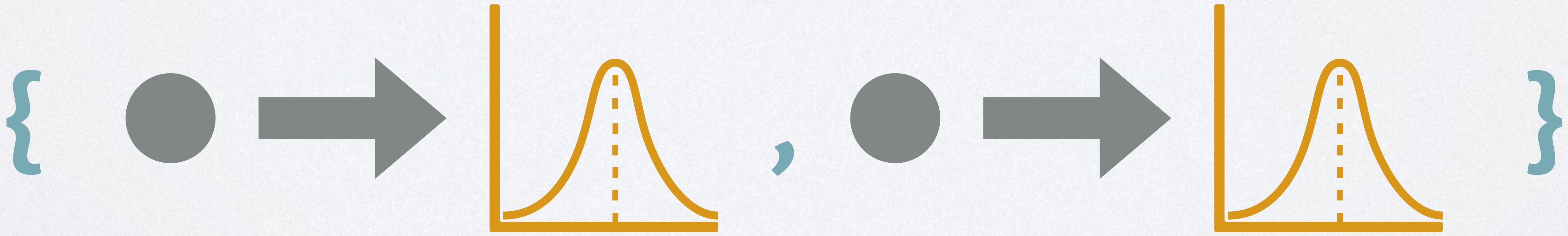
- pGCL: $\mathit{State} \to \wp(\mathbb{D}(\mathit{State}))$



- Cousot's **Probabilistic Abstract Interpretation** (PAI): $\mathbb{D}(\mathit{State} \to \wp(\mathit{State}))$



- **Compile-Time/Relational** Nondeterminism: $\wp(\mathit{State} \to \mathbb{D}(\mathit{State}))$



Challenge II:
How to construct recursive program schemes?

```
while x>0 do
  x :∈p (x+1 @ 1/2 | x-1 @ 1/2)
od
```

→

$\boxed{\mu S\,.\,((\mathsf{x} \coloneqq \mathsf{x{+}1}\;{}_{1/2}\!\oplus \mathsf{x} \coloneqq \mathsf{x{-}1}) \otimes S)_{[\mathsf{x>0}]} \oplus \mathbf{skip}}$

A Control-flow Graph's Perspective

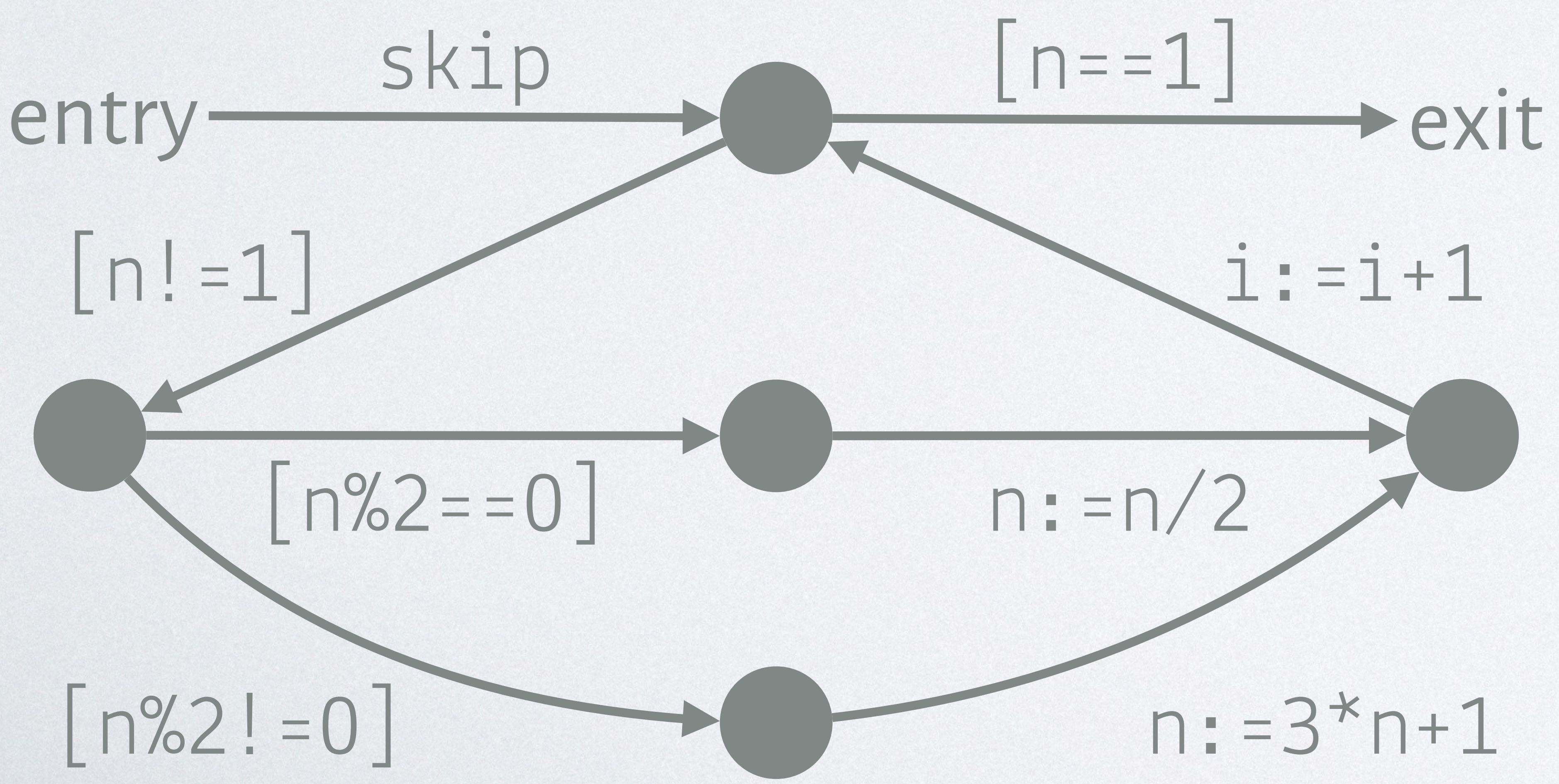
- **Kleene Algebras** are compatible with **control-flow graphs** via **regular expressions**

Program Construct	Algebraic Representation
A program S	An interpretation $\mathbb{S}$ of S into the algebra
The **control-flow graph** of S	A **regular expression** over $\underline{0}$, $\underline{1}$, $\oplus$, $\otimes$, and $*$

A Control-flow Graph's Perspective

- **Kleene Algebras** are compatible with **control-flow graphs** via **regular expressions**

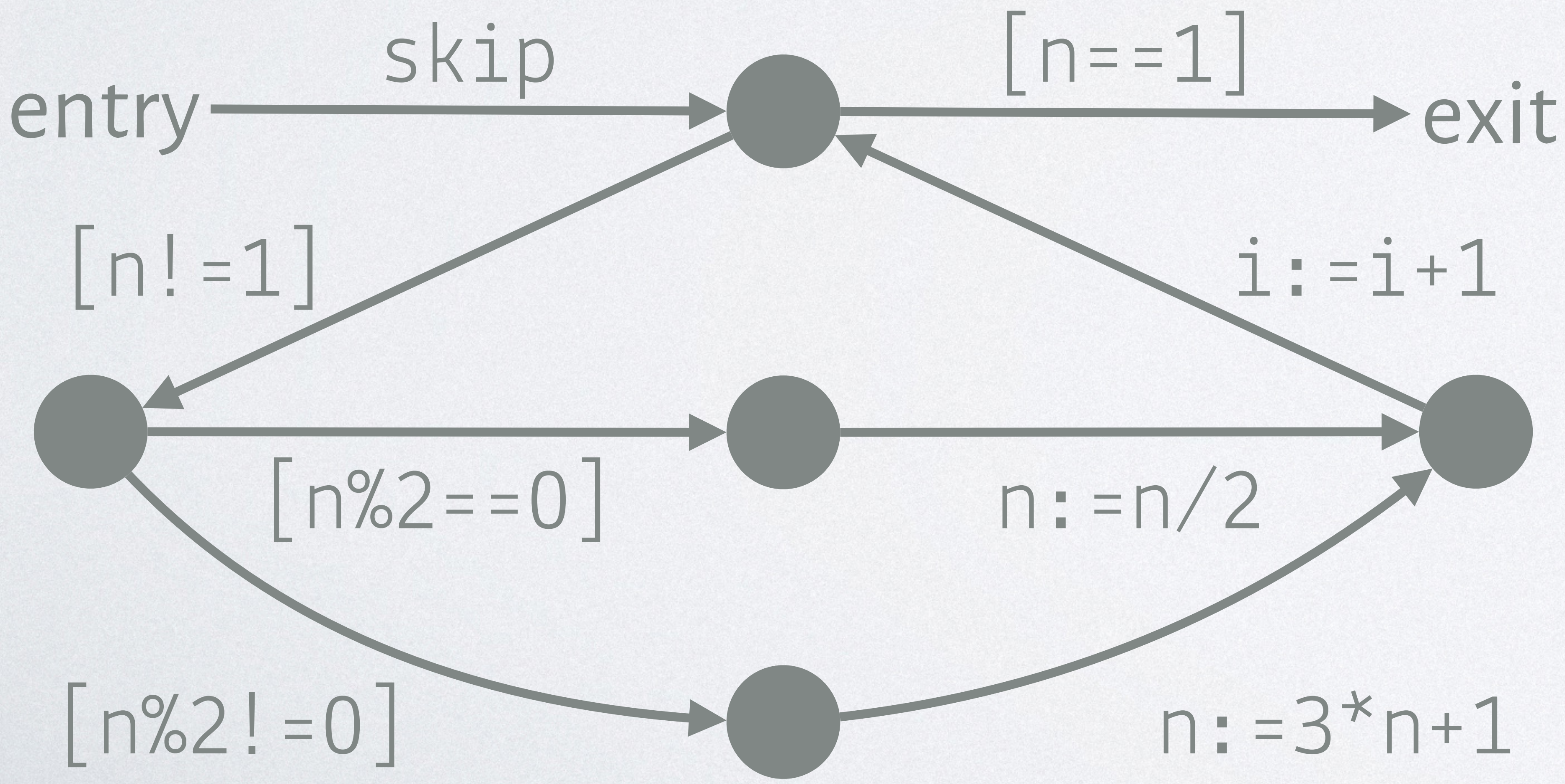
Program Construct	Algebraic Representation
A program S	An interpretation $\mathbb{S}$ of S into the algebra
The **control-flow graph** of S	A **regular expression** over $\underline{0}$, $\underline{1}$, $\oplus$, $\otimes$, and $*$



A Control-flow Graph's Perspective

- **Kleene Algebras** are compatible with **control-flow graphs** via **regular expressions**

Program Construct	Algebraic Representation
A program S	An interpretation $\mathbb{S}$ of S into the algebra
The **control-flow graph** of S	A **regular expression** over $\underline{0}$, $\underline{1}$, $\oplus$, $\otimes$, and $*$



$$\left(\mathtt{[n!=1]} \otimes \left(\begin{array}{l}\mathtt{[n\%2==0]} \otimes \mathtt{n:=n/2} \quad \oplus \\ \mathtt{[n\%2!=0]} \otimes \mathtt{n:=3*n+1}\end{array}\right) \otimes \mathtt{i:=i+1}\right)^{*} \otimes \mathtt{[n==1]}$$

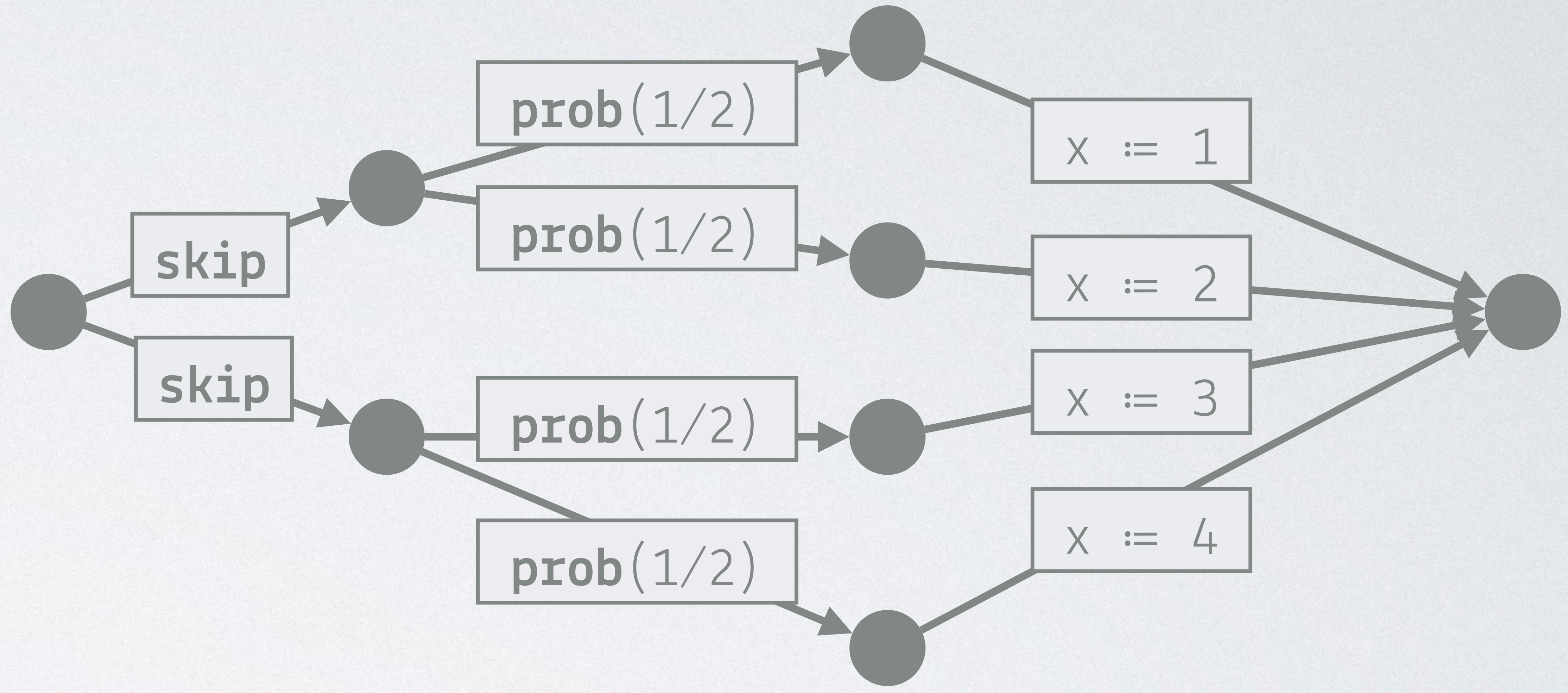
Do Control-flow Graphs Suffice?

Do Control-flow Graphs Suffice?

```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```

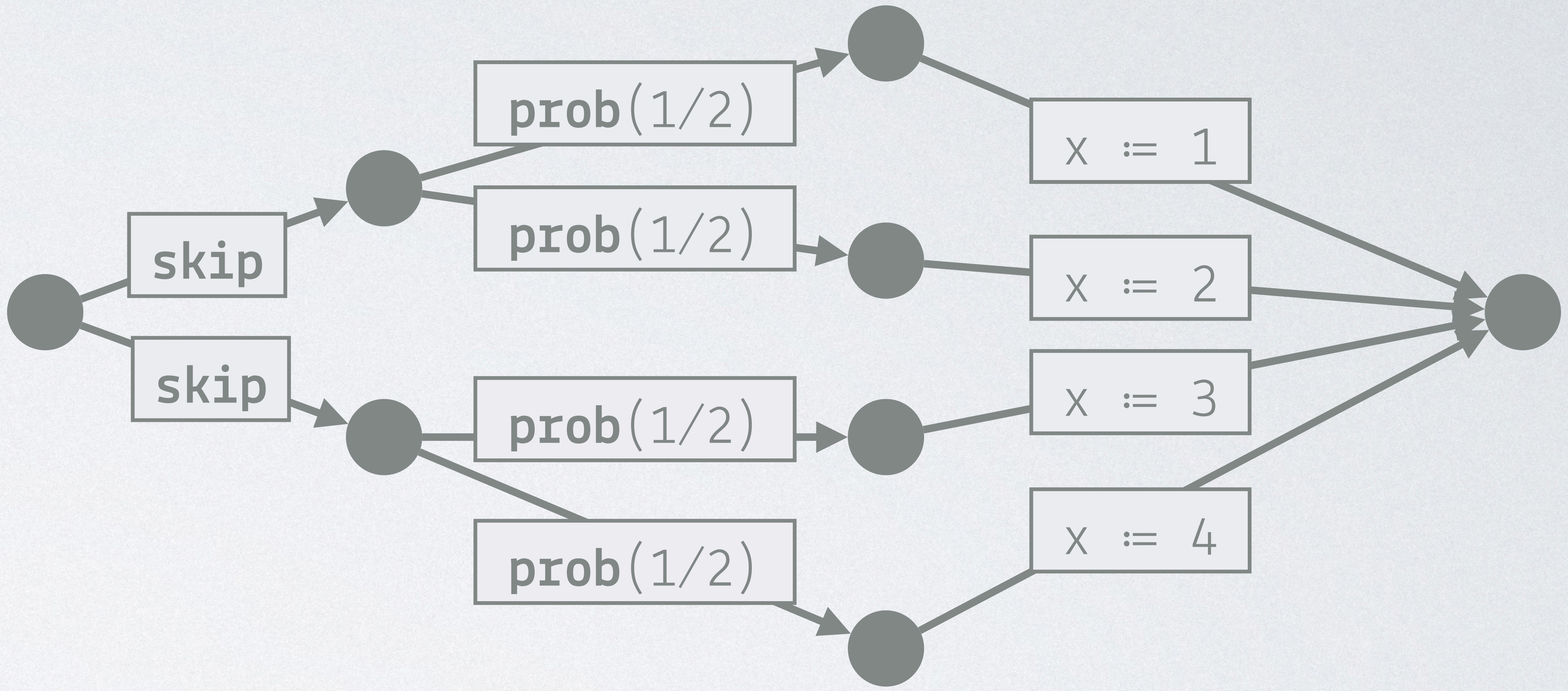
Do Control-flow Graphs Suffice?

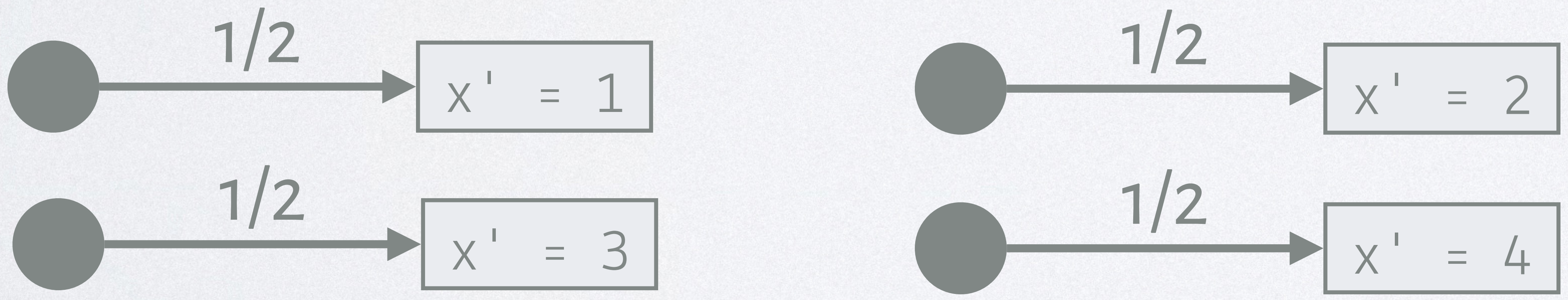
```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```



Do Control-flow Graphs Suffice?

```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```

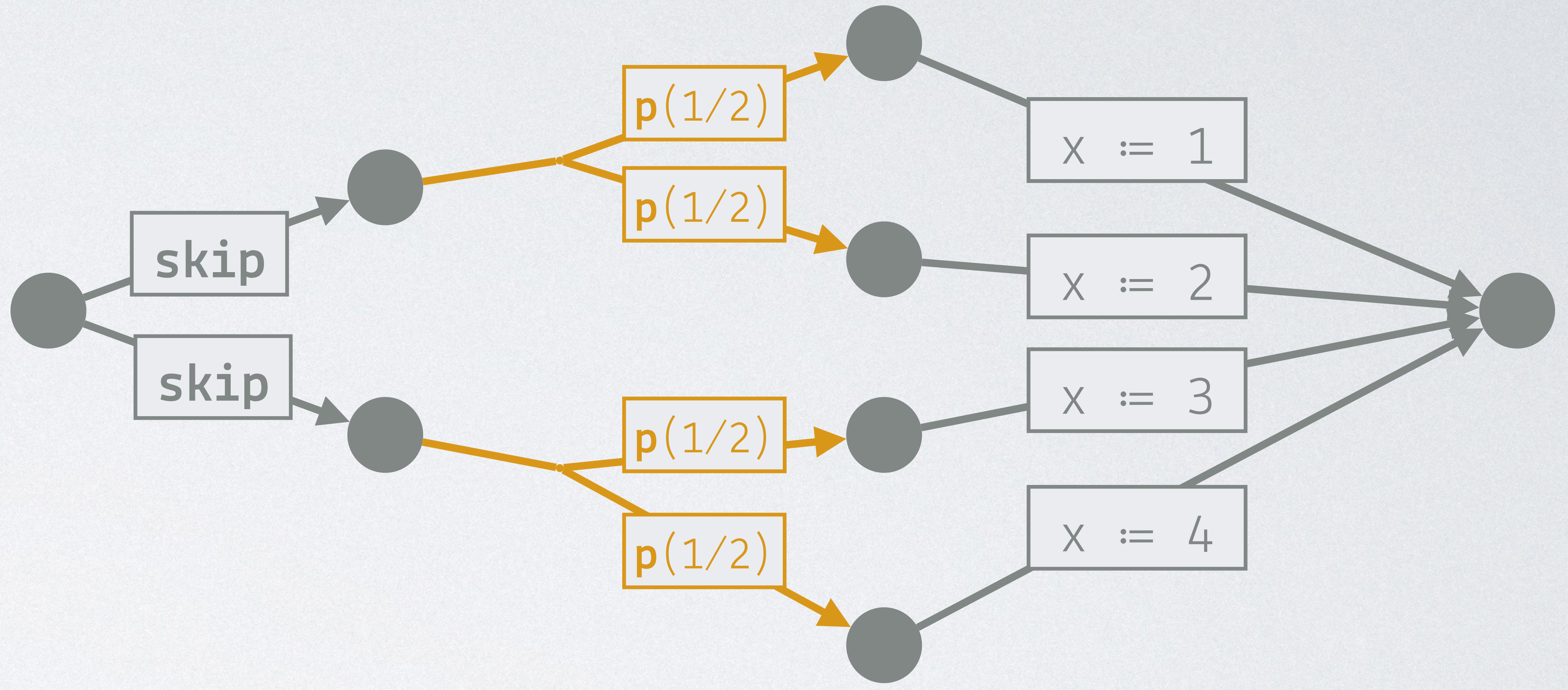




Probabilities sum up to 2!

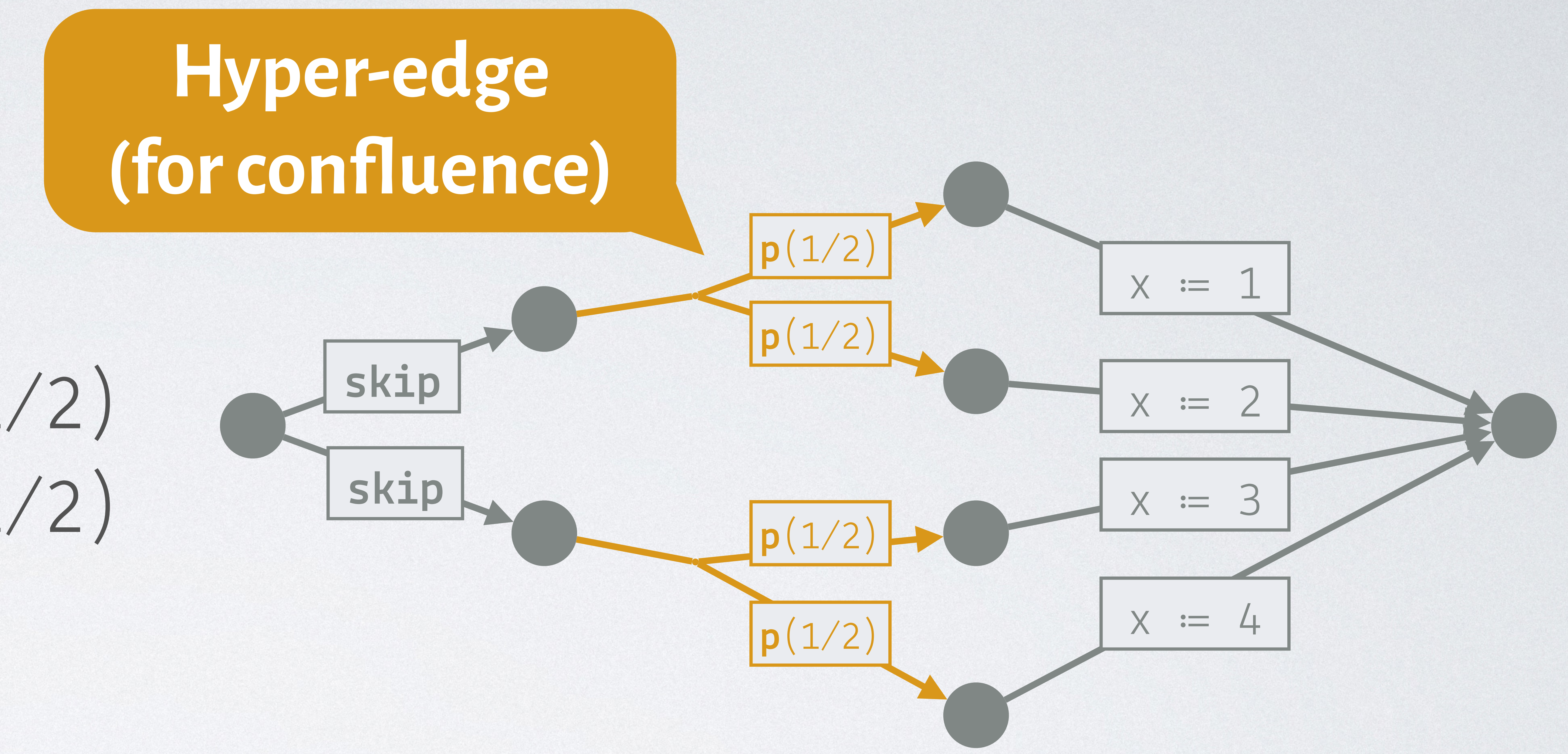
Our Approach: Control-flow Hyper-graphs

```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```



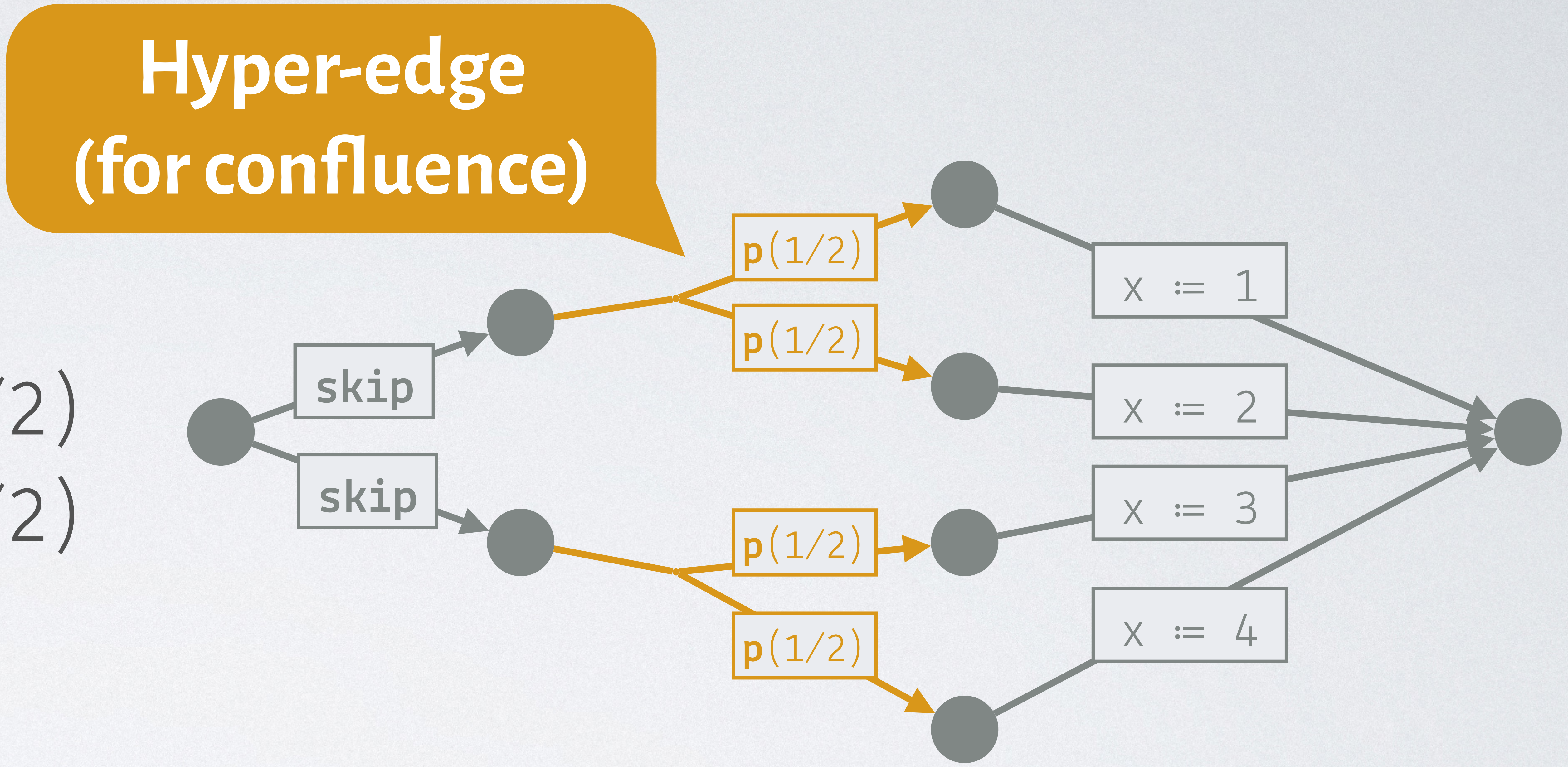
Our Approach: Control-flow Hyper-graphs

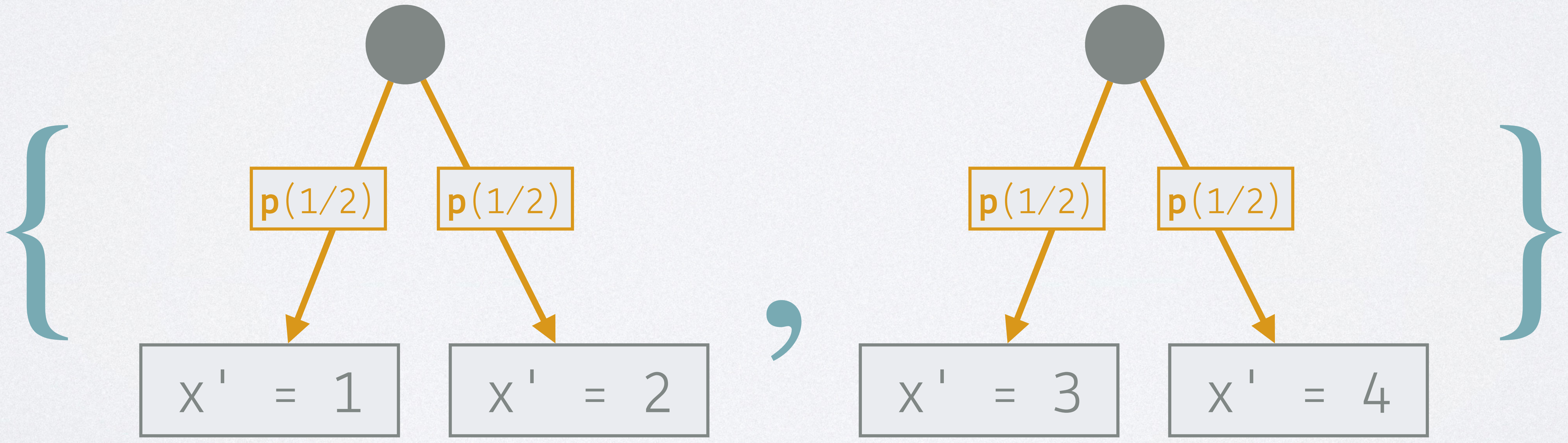
```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```



Our Approach: Control-flow Hyper-graphs

```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```

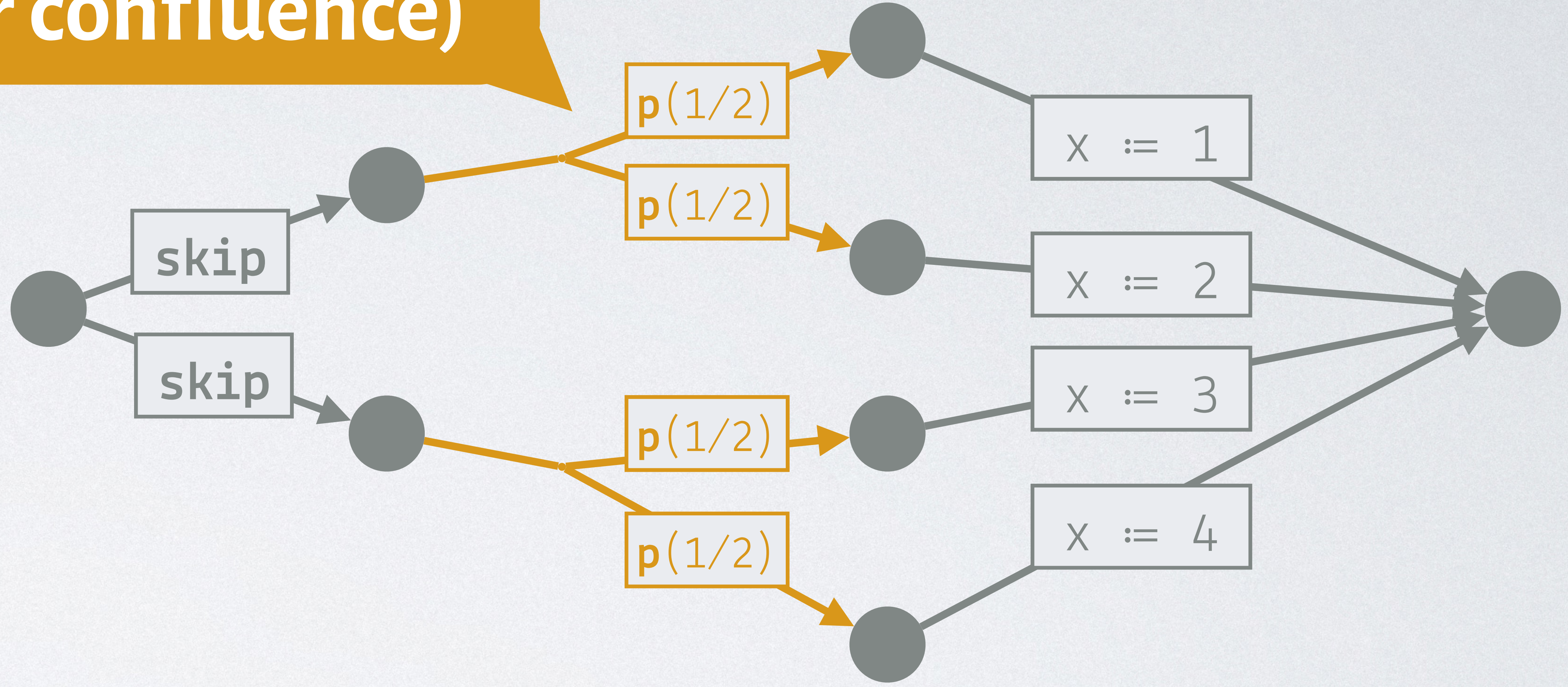




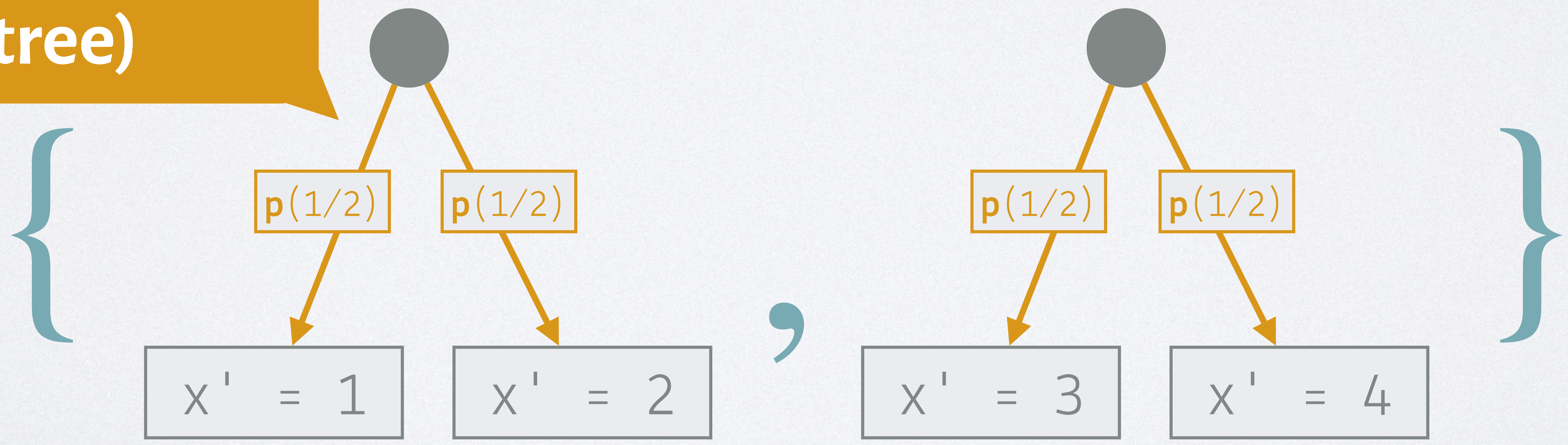
Our Approach: Control-flow Hyper-graphs

```
if
| true → x :∈p (1 @ 1/2 | 2 @ 1/2)
| true → x :∈p (3 @ 1/2 | 4 @ 1/2)
fi
```

Hyper-edge (for confluence)

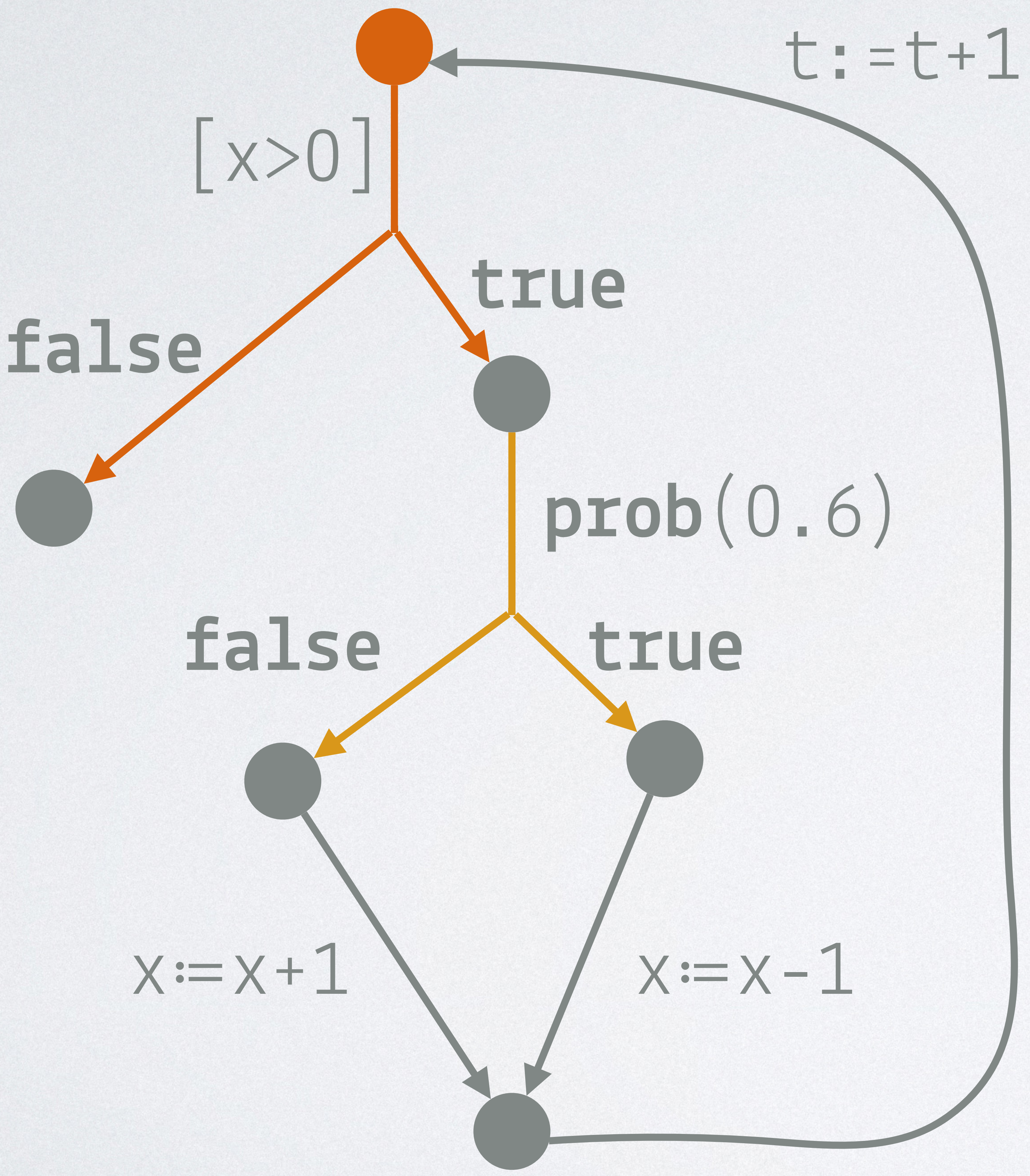


Hyper-path (like a tree)

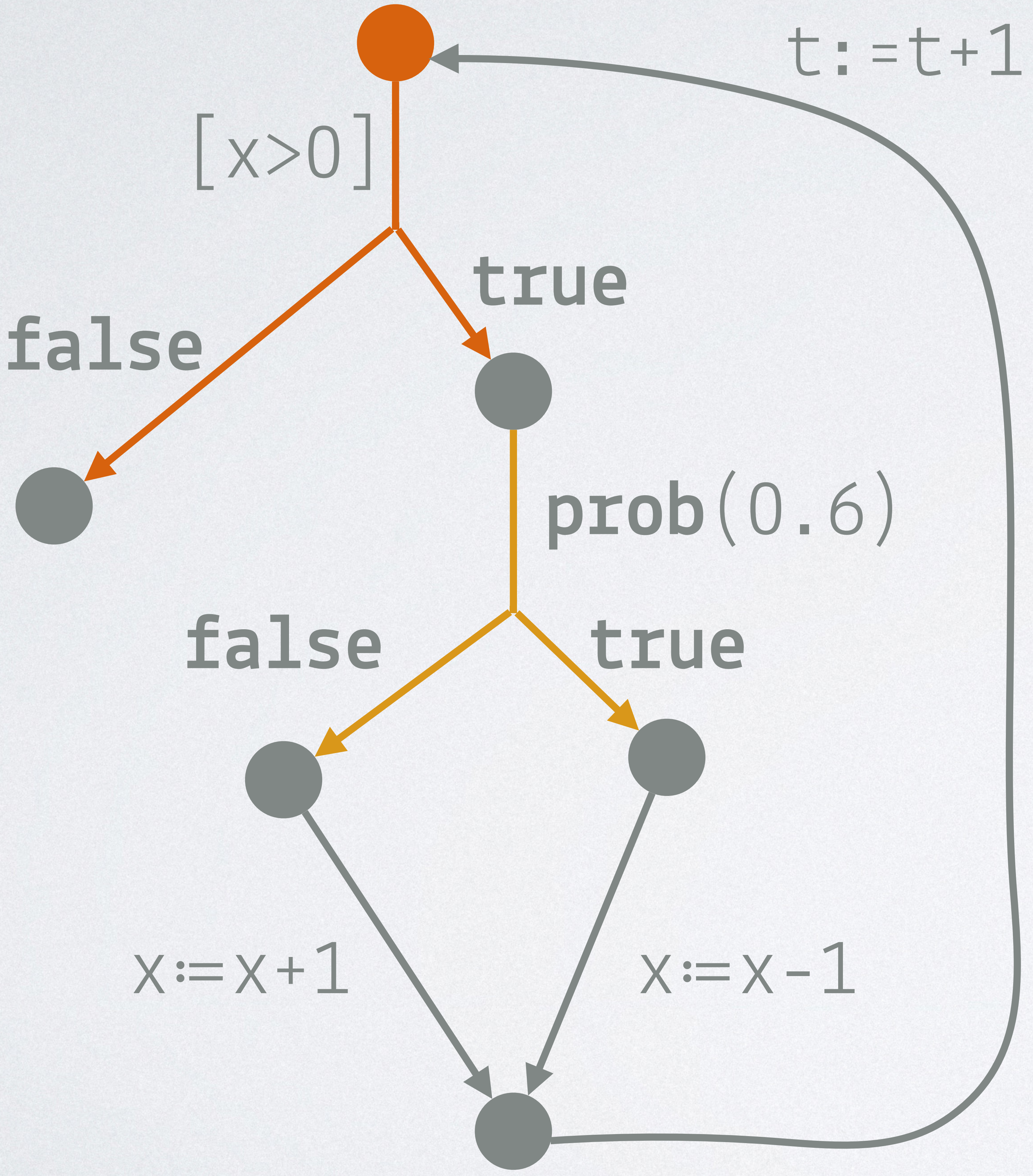


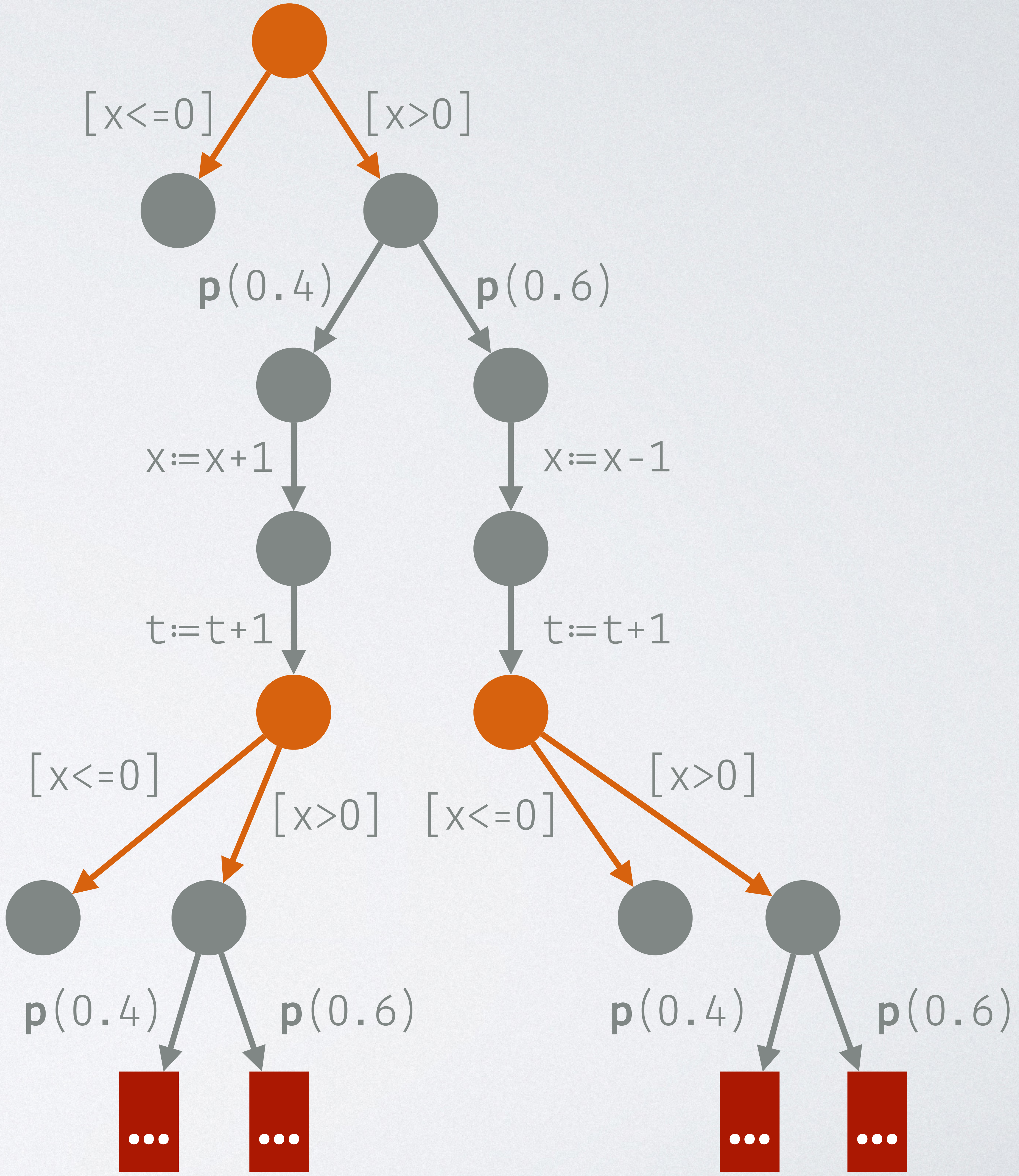
Hyper-paths are Infinite Trees!

Hyper-paths are Infinite Trees!

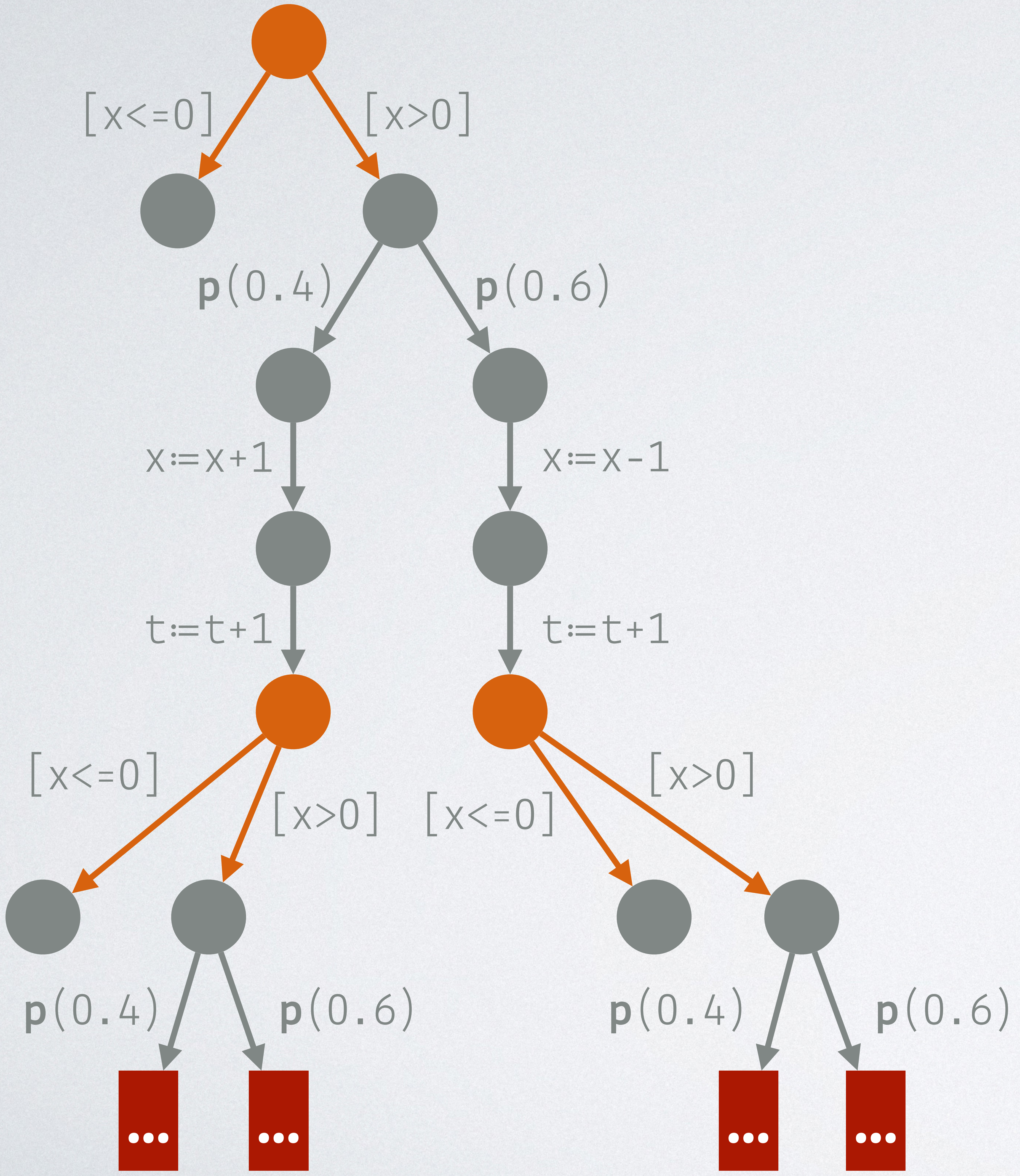


Hyper-paths are Infinite Trees!

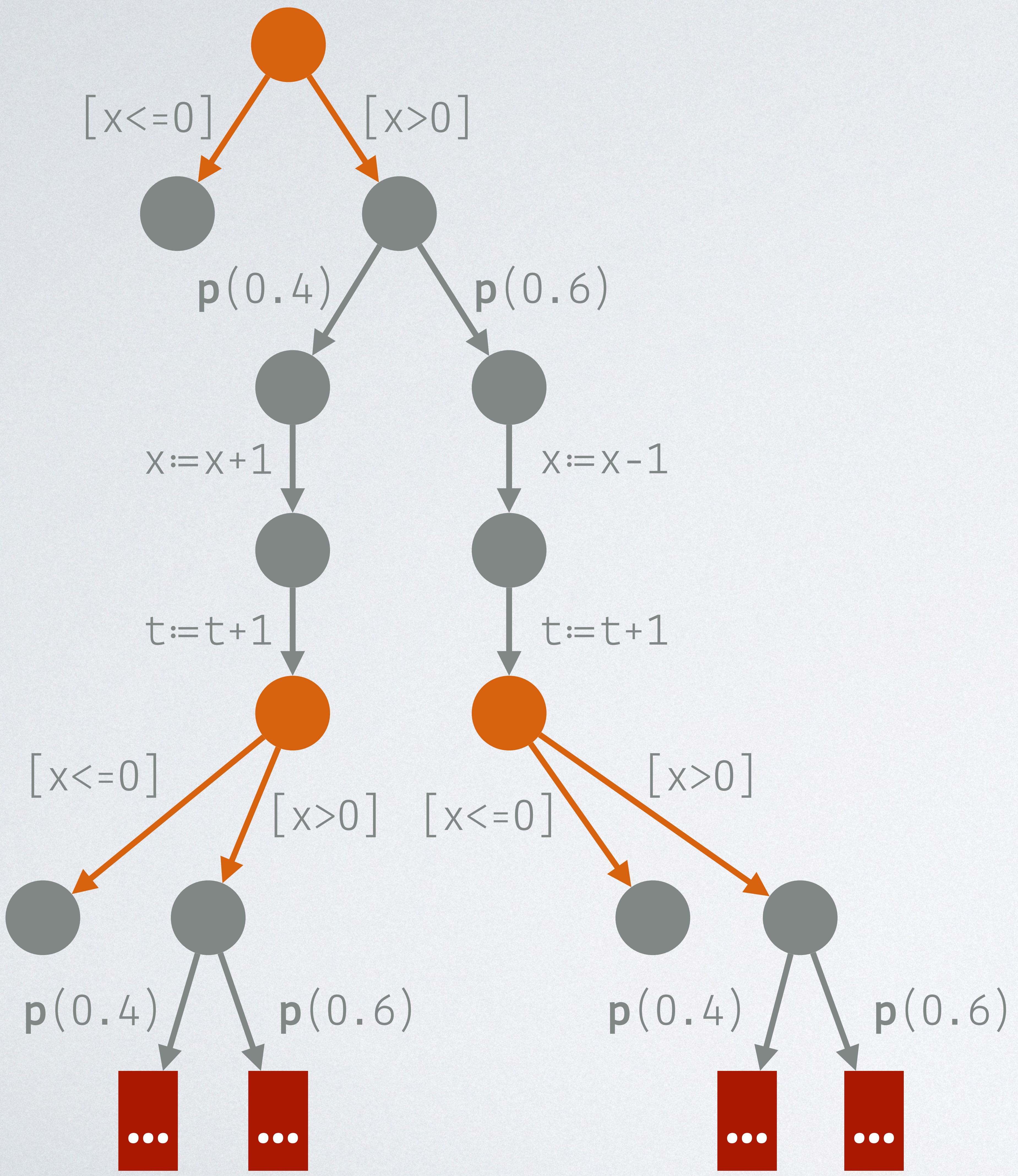


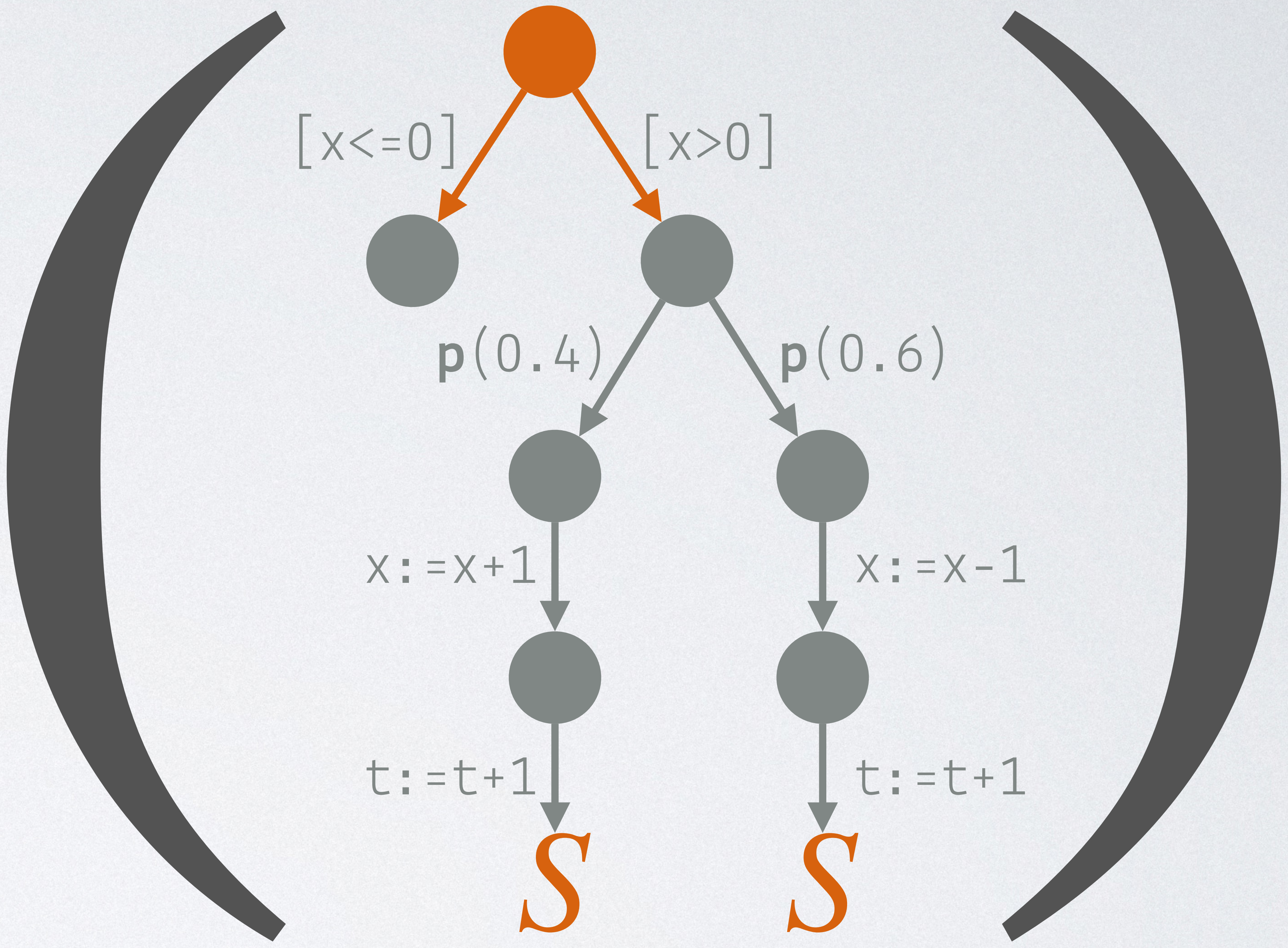


Recursive Program Schemes

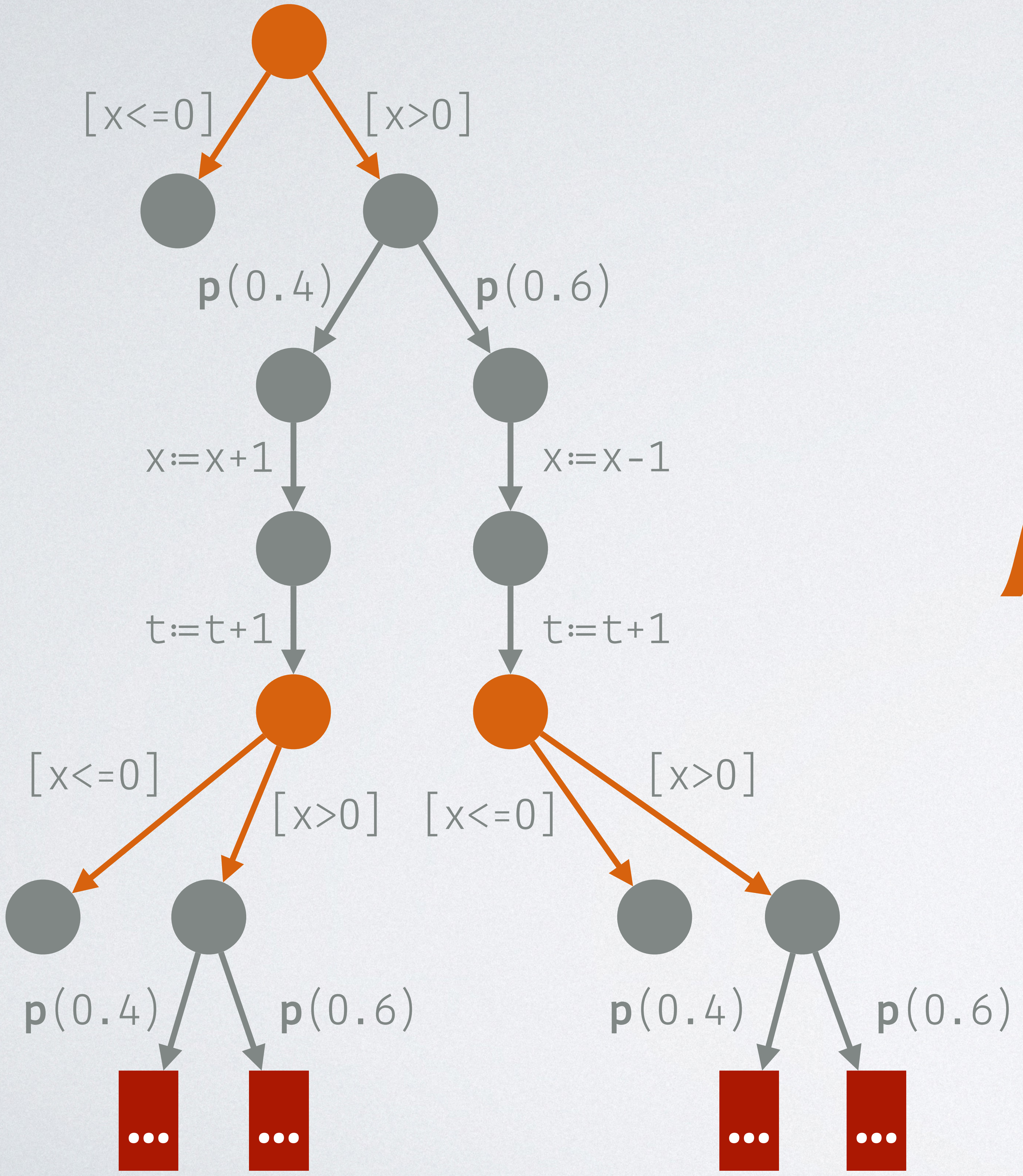


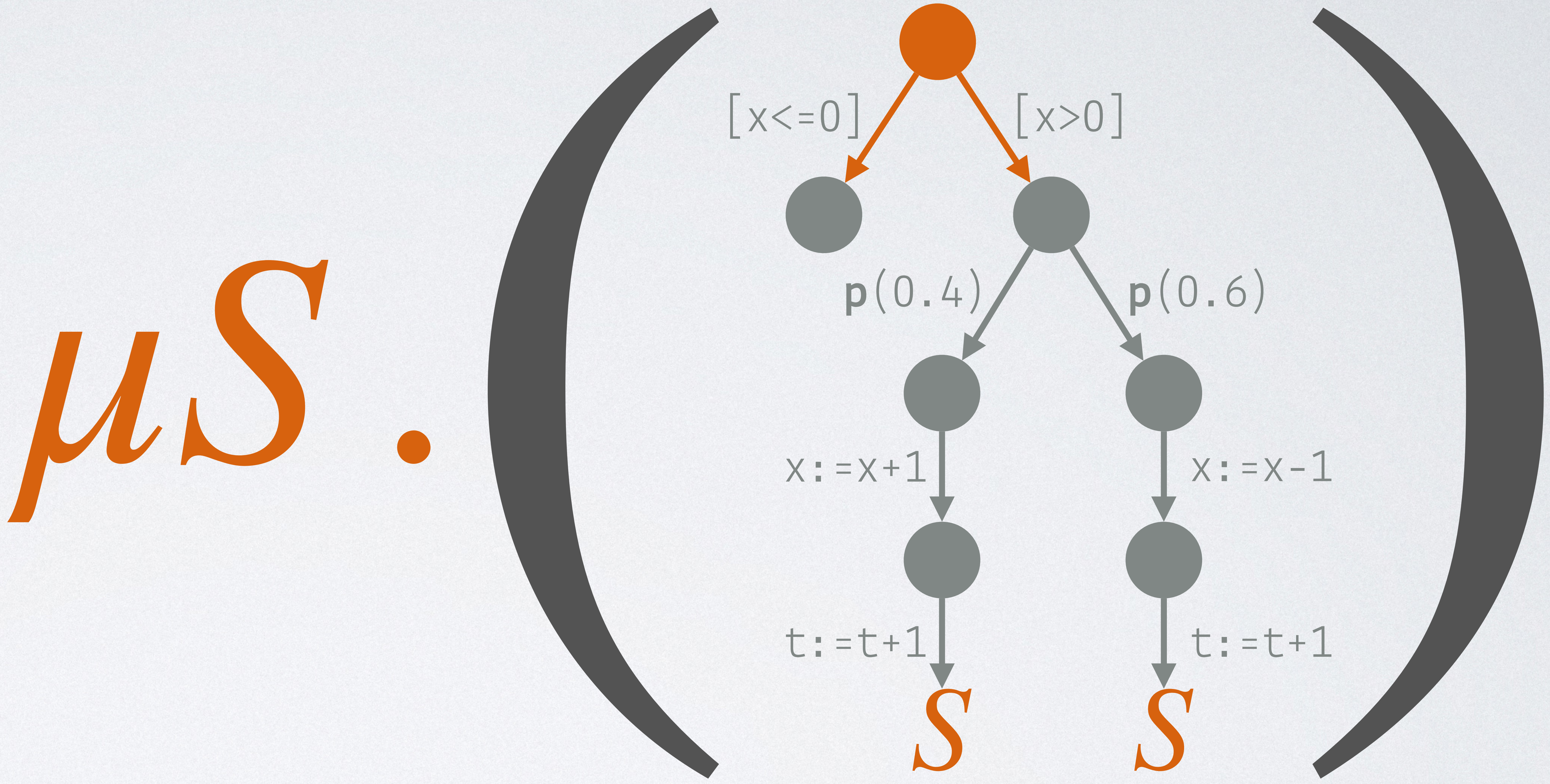
Recursive Program Schemes



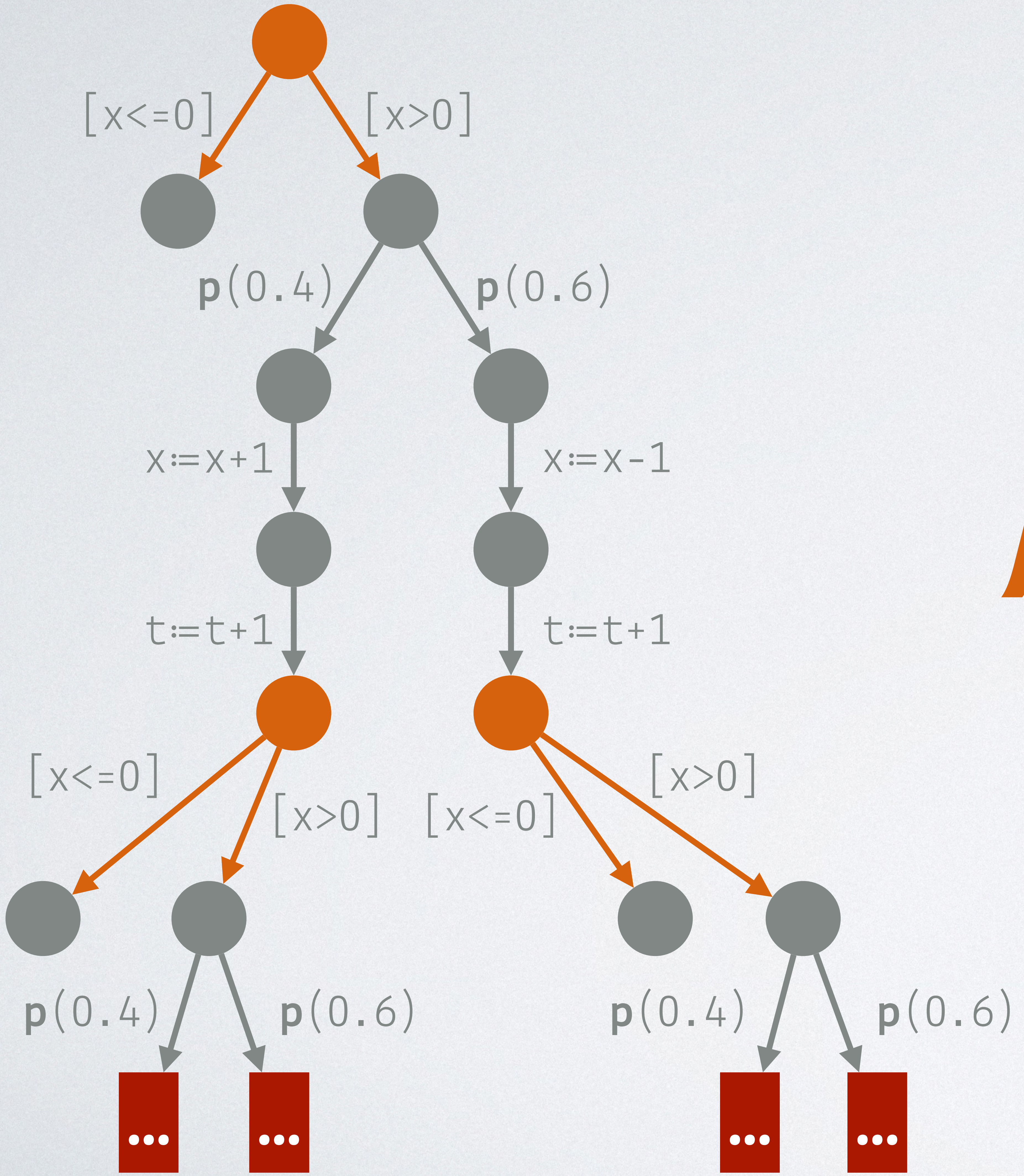


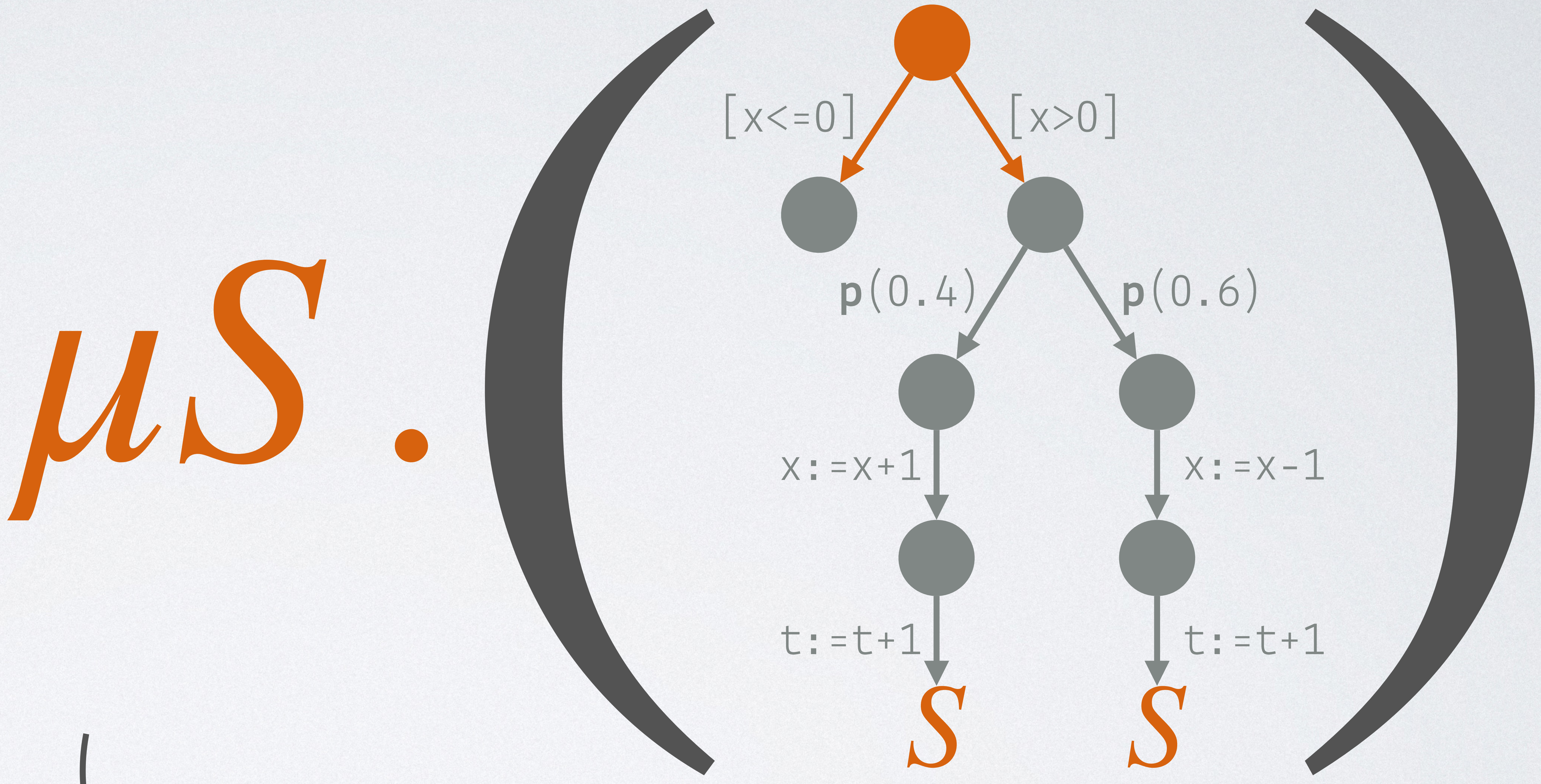
Recursive Program Schemes





Recursive Program Schemes





$$\mu S\,.\left(\begin{array}{r}\text{cond}[x>0](\text{prob}[0.6](\text{seq}[x:=x-1](\text{seq}[t:=t+1](S));\\ \text{seq}[x:=x+1](\text{seq}[t:=t+1](S)));\\ \underline{1})\qquad\qquad\qquad\qquad\qquad\qquad\end{array}\right)$$

A Control-flow Hyper-graph's Perspective

- **Markov Algebras** are compatible with **control-flow hyper-graphs** via **recursive program schemes**

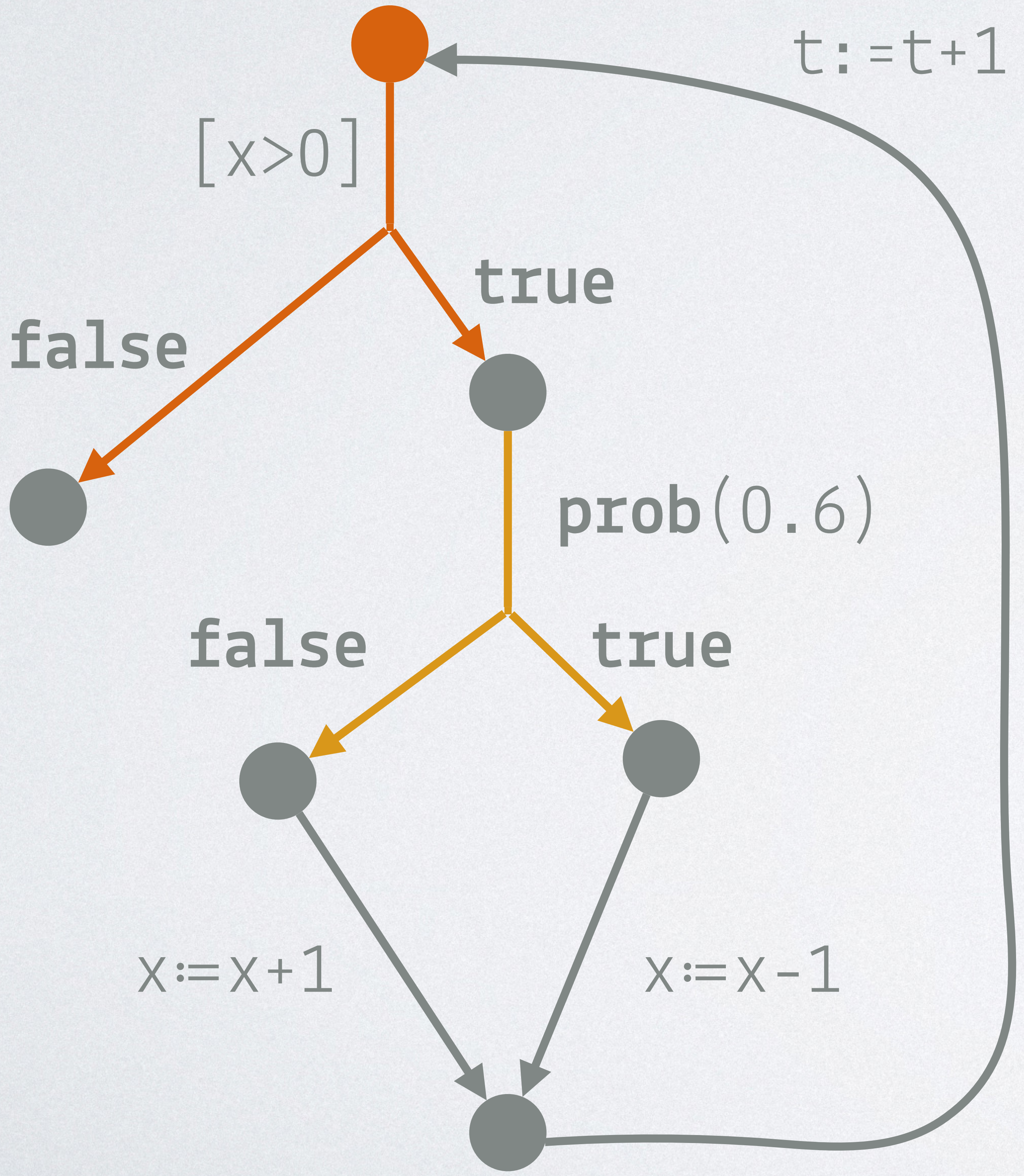
A Control-flow Hyper-graph's Perspective

- **Markov Algebras** are compatible with **control-flow hyper-graphs** via **recursive program schemes**

```
while x>0 do
  if prob(0.6) then x:=x+1
  else x:=x-1 fi;
  t:=t+1
od
```

A Control-flow Hyper-graph's Perspective

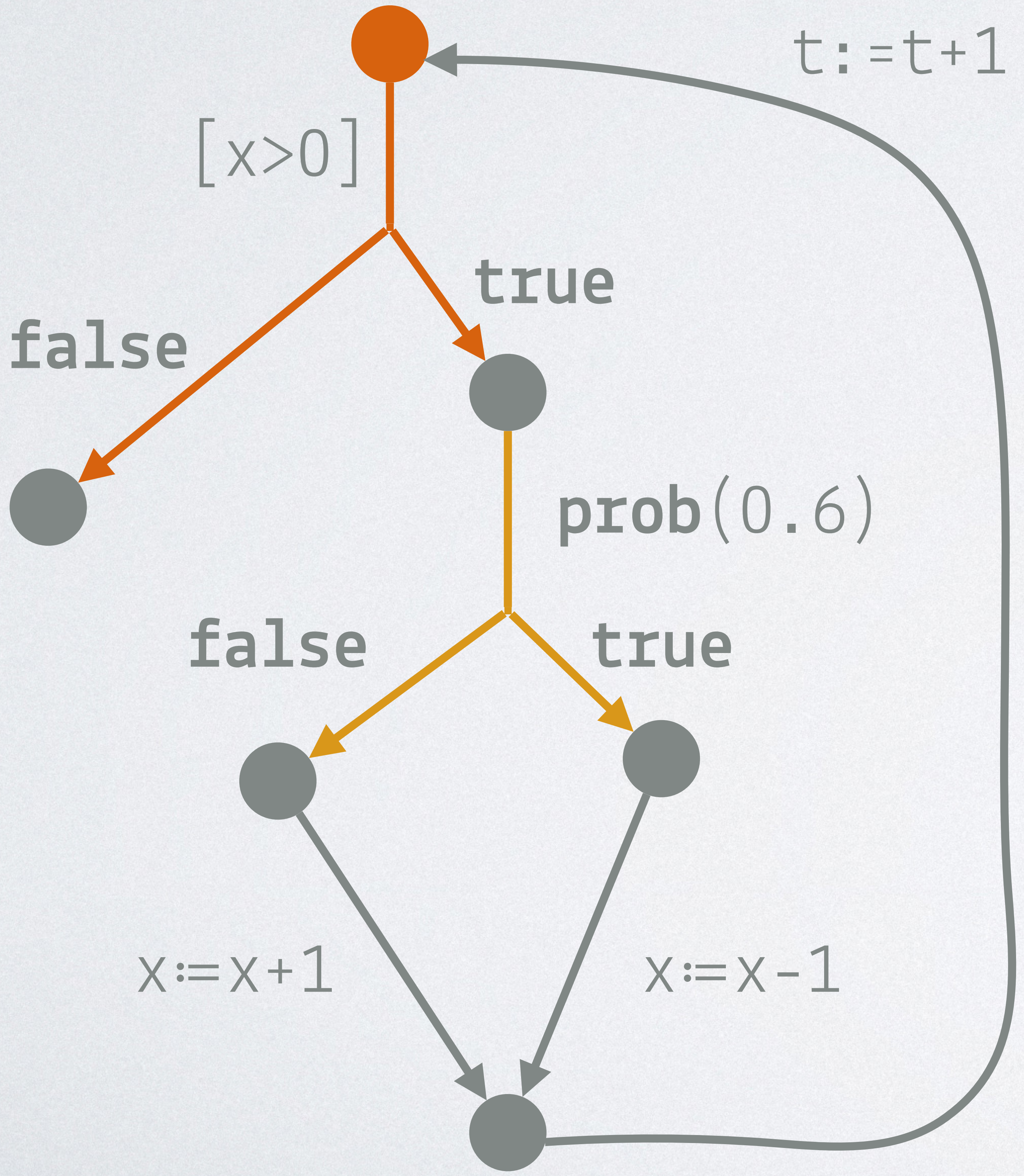
- **Markov Algebras** are compatible with **control-flow hyper-graphs** via **recursive program schemes**




```
while x>0 do
  if prob(0.6) then x:=x+1
  else x:=x-1 fi;
  t:=t+1
od
```

A Control-flow Hyper-graph's Perspective

- **Markov Algebras** are compatible with **control-flow hyper-graphs** via **recursive program schemes**




```
while x>0 do
  if prob(0.6) then x:=x+1
  else x:=x-1 fi;
  t:=t+1
od
```

$$\mu S\,.\left(\begin{array}{r} \mathrm{cond}[x > 0](\mathrm{prob}[0.6](\mathrm{seq}[x := x - 1](\mathrm{seq}[t := t + 1](S)); \\ \mathrm{seq}[x := x + 1](\mathrm{seq}[t := t + 1](S))); \\ \underline{1}) \end{array}\right)$$

Challenge III:
How to carry out quantitative analyses efficiently?

```
while prob(2/3) do
  x ≔ x + 1
od
```

Iterative Analysis

```
while prob(2/3) do
  x ≔ x + 1
od
```

$$\mu S \,.\, ((\mathsf{x} \coloneqq \mathsf{x}{+}1) \otimes S)_{2/3}\!\oplus \mathbf{skip}$$

Iterative Analysis

```
while prob(2/3) do
  x := x + 1
od
```

$$\mu S\,.\,((\mathsf{x} \coloneqq \mathsf{x{+}1}) \otimes S)_{2/3}\oplus \textbf{skip}$$

- Markov algebra for computing $\mathbb{E}[\Delta x]$
  - Sequencing: $r \otimes t \triangleq r + t$
  - Branching: $r_p\oplus t \triangleq p * r + (1 - p) * t$

Iterative Analysis

```
while prob(2/3) do
  x := x + 1
od
```

$$\mu S\,.\,((\mathtt{x} \coloneqq \mathtt{x{+}1}) \otimes S)_{2/3}\!\oplus \mathbf{skip}$$

$$\Downarrow$$

$$\kappa^{(0)} = 0$$
$$\kappa^{(1)} = 2/3 * (1 + \kappa^{(0)}) + 1/3 * 0 = 2/3$$
$$\kappa^{(2)} = 2/3 * (1 + \kappa^{(1)}) + 1/3 * 0 = 10/9$$
$$\cdots$$
$$\kappa^{(\infty)} = 2$$

- Markov algebra for computing $\mathbb{E}[\Delta x]$
  - Sequencing: $r \otimes t \triangleq r + t$
  - Branching: $r_p\!\oplus t \triangleq p * r + (1 - p) * t$

Iterative Analysis

```
while prob(2/3) do
  x := x + 1
od
```

$$\mu S\,.\,((\mathtt{x := x+1}) \otimes S)_{2/3}\oplus\ \textbf{skip}$$

- Markov algebra for computing $\mathbb{E}[\Delta x]$
  - Sequencing: $r \otimes t \triangleq r + t$
  - Branching: $r_p\oplus t \triangleq p * r + (1-p) * t$

$$\kappa^{(0)} = 0$$
$$\kappa^{(1)} = 2/3 * (1 + \kappa^{(0)}) + 1/3 * 0 = 2/3$$
$$\kappa^{(2)} = 2/3 * (1 + \kappa^{(1)}) + 1/3 * 0 = 10/9$$
$$\cdots$$
$$\kappa^{(\infty)} = 2$$

Need ∞ iterations to converge!

Non-iterative Analysis

```
while prob(2/3) do
  x ≔ x + 1
od
```

$\mu S\,.\,((\mathsf{x} \coloneqq \mathsf{x+1}) \otimes S)_{2/3}\!\oplus \textbf{skip}$

- Markov algebra for computing $\mathbb{E}[\Delta x]$
  - Sequencing: $r \otimes t \triangleq r + t$
  - Branching: $r_p\!\oplus t \triangleq p * r + (1-p) * t$

Non-iterative Analysis

```
while prob(2/3) do
  x := x + 1
od
```

$$\mu S\,.\,((\mathrm{x} := \mathrm{x}{+}1) \otimes S)_{2/3}\!\oplus \mathbf{skip}$$

- Markov algebra for computing $\mathbb{E}[\Delta x]$
  - Sequencing: $r \otimes t \triangleq r + t$
  - Branching: $r_p\!\oplus t \triangleq p * r + (1 - p) * t$

Equivalent to solve:

$$s = 2/3 * (1 + s) + 1/3 * 0,$$

Analytical solution:

$$s = 2$$

No need for iteration!

Non-iterative **Intra**-procedural Analysis

Non-iterative **Intra**-procedural Analysis

- Observation: Loops are (right-)**linear recursions**, thus we can always extract a system of **linear equations**
    - For each $\mu S\,.\,E$, we extract an equation $S = E$

Non-iterative **Intra**-procedural Analysis

- Observation: Loops are (right-) **linear recursions**, thus we can always extract a system of **linear equations**
  - For each $\mu S \, . \, E$, we extract an equation $S = E$
- Techniques to solve linear equation systems extracted from probabilistic programs:
  - **Linear Programming:** Compute probabilities or expectations
  - **Loop-Invariant Generation:** Derive probabilistic or expectation invariants

Inter-procedural Analysis

Program → Control-flow hyper-graph → System of **nonlinear** equations → Solution

Inter-procedural Analysis

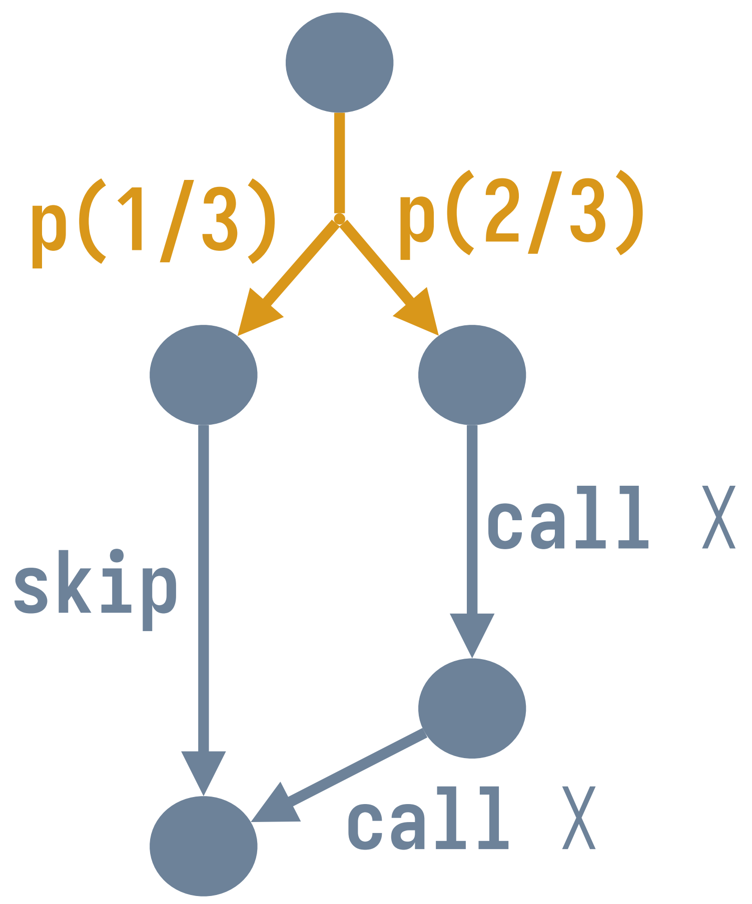
Program → Control-flow hyper-graph → System of **nonlinear** equations → Solution

```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```

Inter-procedural Analysis

Program → Control-flow hyper-graph → System of **nonlinear** equations → Solution

```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```

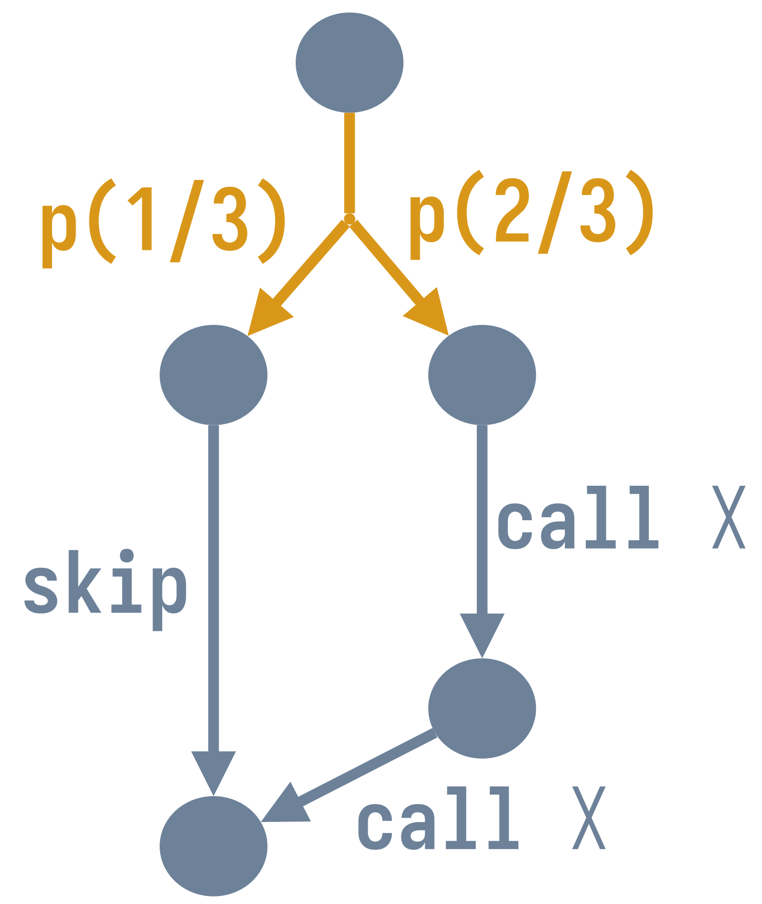


Inter-procedural Analysis

Abstraction engine

Program → Control-flow hyper-graph → System of **nonlinear** equations → Solution

```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```

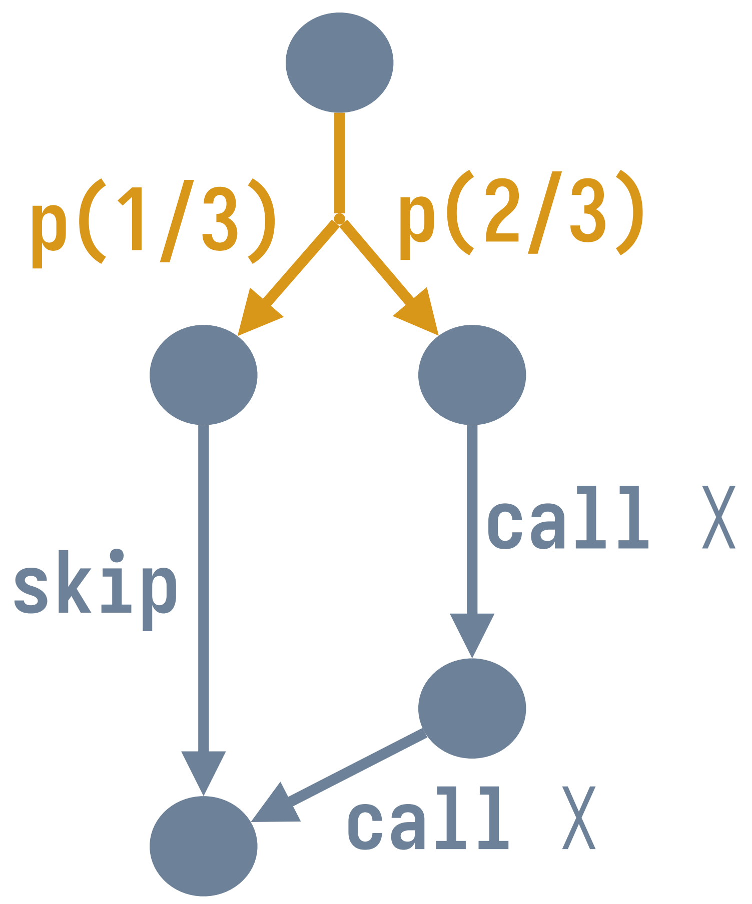


Inter-procedural Analysis

Program → Control-flow hyper-graph → System of **nonlinear** equations → Solution

Abstraction engine

```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



$$X = (\mathbf{skip} \otimes \underline{1}) \;{}_{1/3}\!\oplus\; (X \otimes X \otimes \underline{1})$$

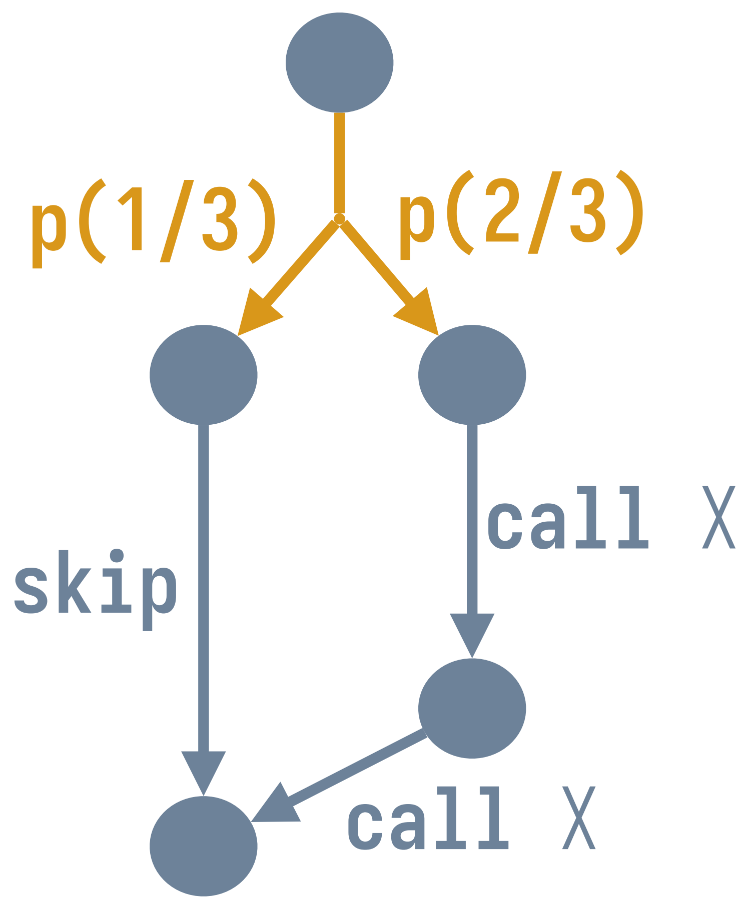
Inter-procedural Analysis

Abstraction engine

Equation solver

Program → Control-flow hyper-graph → System of **nonlinear** equations → Solution

```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



$$X = (\mathbf{skip} \otimes \underline{1}) \;{}_{1/3}\!\oplus\; (X \otimes X \otimes \underline{1})$$

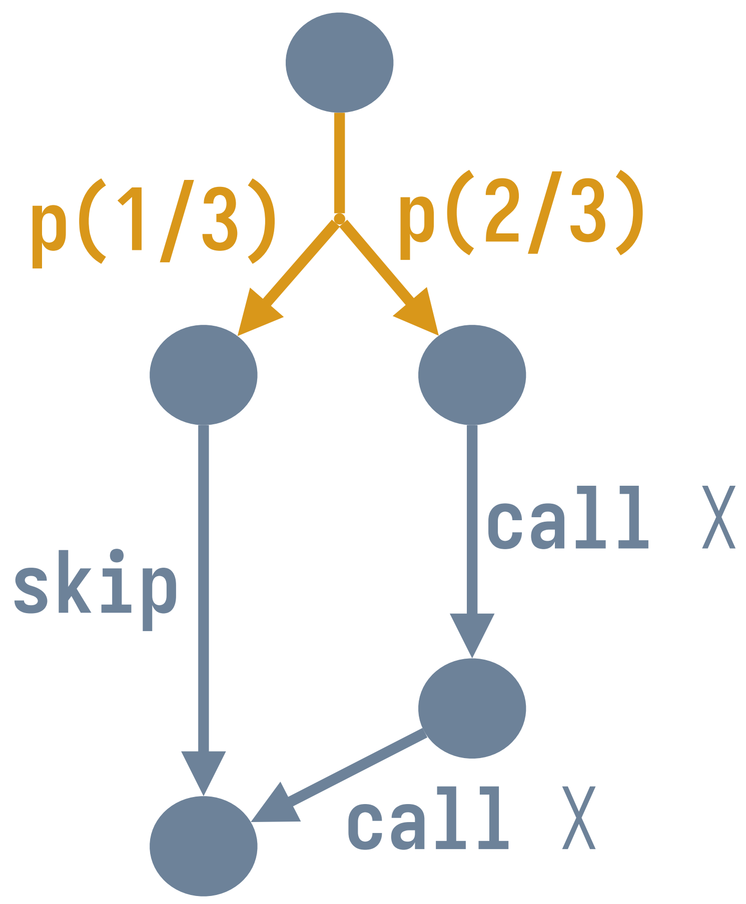
Inter-procedural Analysis

Abstraction engine

Equation solver

Program → Control-flow hyper-graph → System of **nonlinear** equations → Solution

```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



$$X = (\mathbf{skip} \otimes \underline{1}) \;{}_{1/3}\!\oplus\; (X \otimes X \otimes \underline{1})$$

X = "termination probability is 1/2"

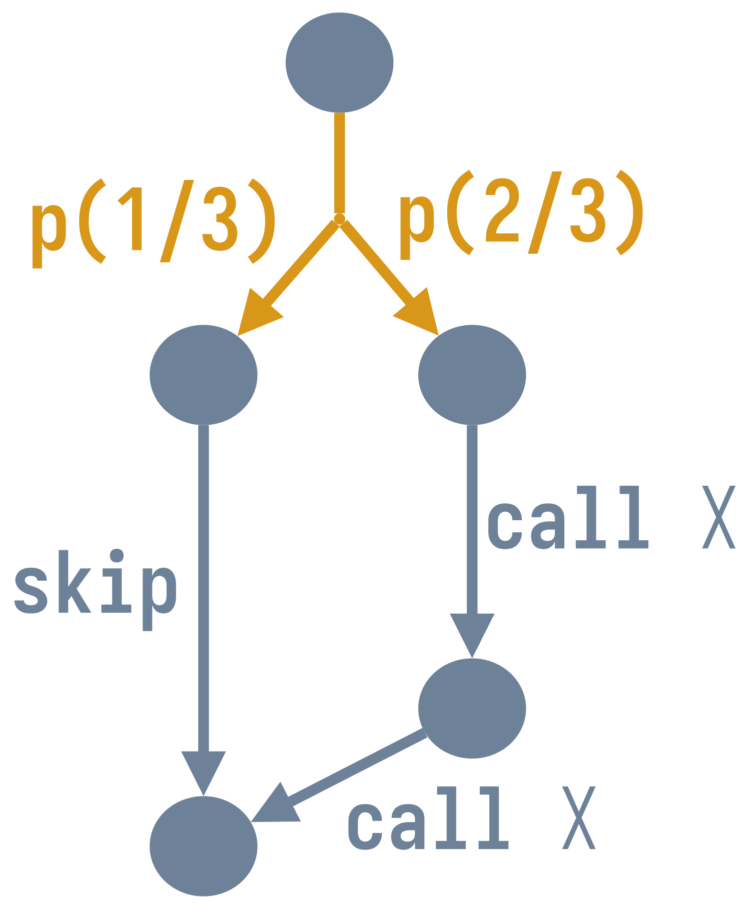
Inter-procedural Analysis

Abstraction engine

Equation solver

Program → Control-flow hyper-graph → System of **nonlinear** equations → Solution

```
proc X begin
  if prob(1/3)
  then skip
  else
    call X;
    call X
  fi
end
```



$$X = (\mathbf{skip} \otimes \underline{1}) \;{}_{1/3}\!\oplus (X \otimes X \otimes \underline{1})$$

X = "termination probability is 1/2"

Also a **procedure summary** for X

The **Functional** Approach for Inter-procedural Analysis

The **Functional** Approach for Inter-procedural Analysis

- Let X_i represent the **procedure summary** of P_i

$$
\begin{aligned}
X_1 &= f_1(X_1, X_2, \ldots, X_N) \\
X_2 &= f_2(X_1, X_2, \ldots, X_N) \\
&\vdots \\
X_N &= f_N(X_1, X_2, \ldots, X_N)
\end{aligned}
$$

The **Functional** Approach for Inter-procedural Analysis

- Let X_i represent the **procedure summary** of P_i
- Solve the equation system using successive approximation (e.g., Kleene iteration)

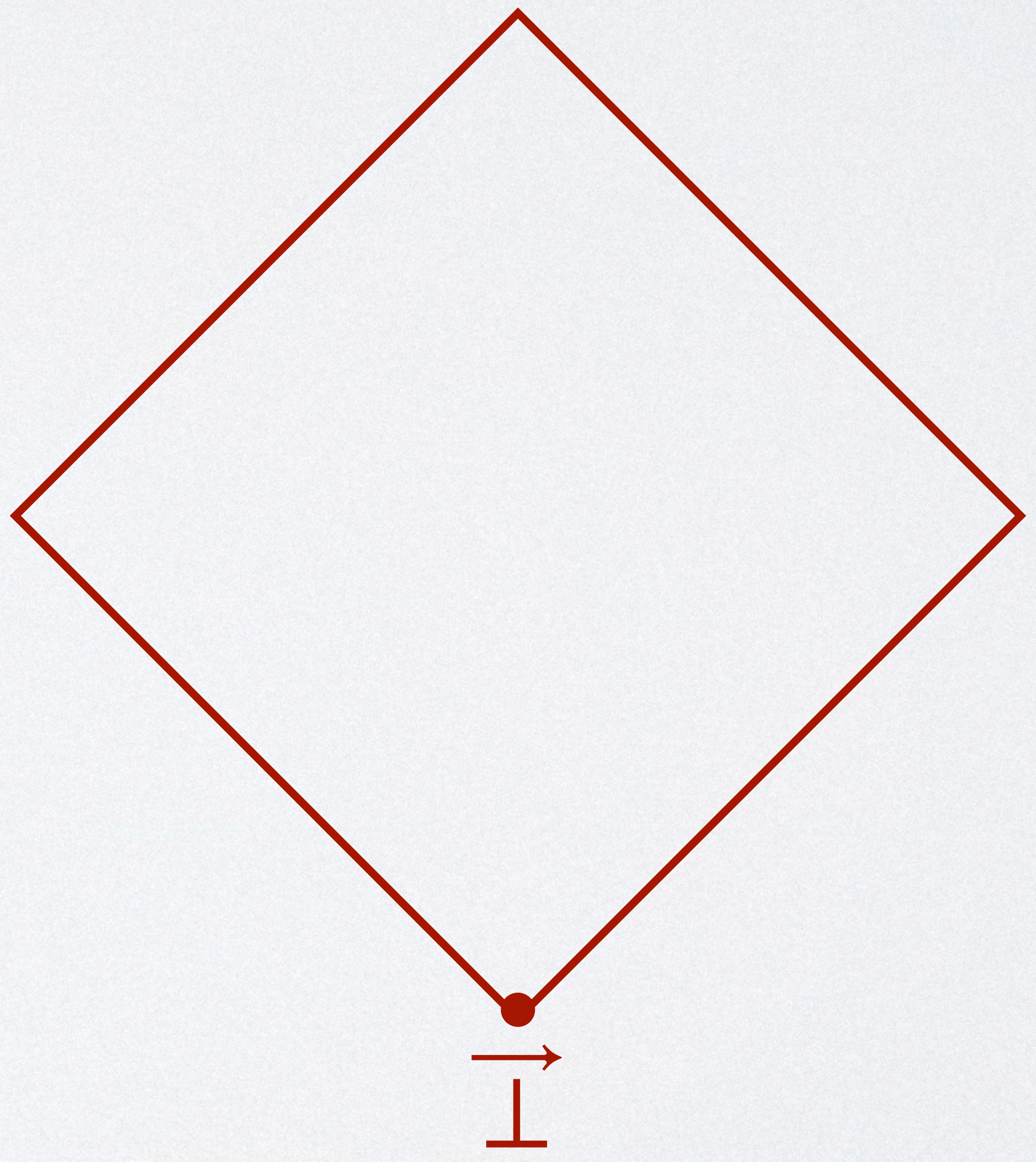
$$
\begin{aligned}
X_1 &= f_1(X_1, X_2, \ldots, X_N) \\
X_2 &= f_2(X_1, X_2, \ldots, X_N) \\
&\vdots \\
X_N &= f_N(X_1, X_2, \ldots, X_N)
\end{aligned}
$$

The **Functional** Approach for Inter-procedural Analysis

- Let X_i represent the **procedure summary** of P_i

- Solve the equation system using successive approximation (e.g., Kleene iteration)

$$
\begin{aligned}
X_1 &= f_1(X_1, X_2, \ldots, X_N) \\
X_2 &= f_2(X_1, X_2, \ldots, X_N) \\
&\vdots \\
X_N &= f_N(X_1, X_2, \ldots, X_N)
\end{aligned}
$$

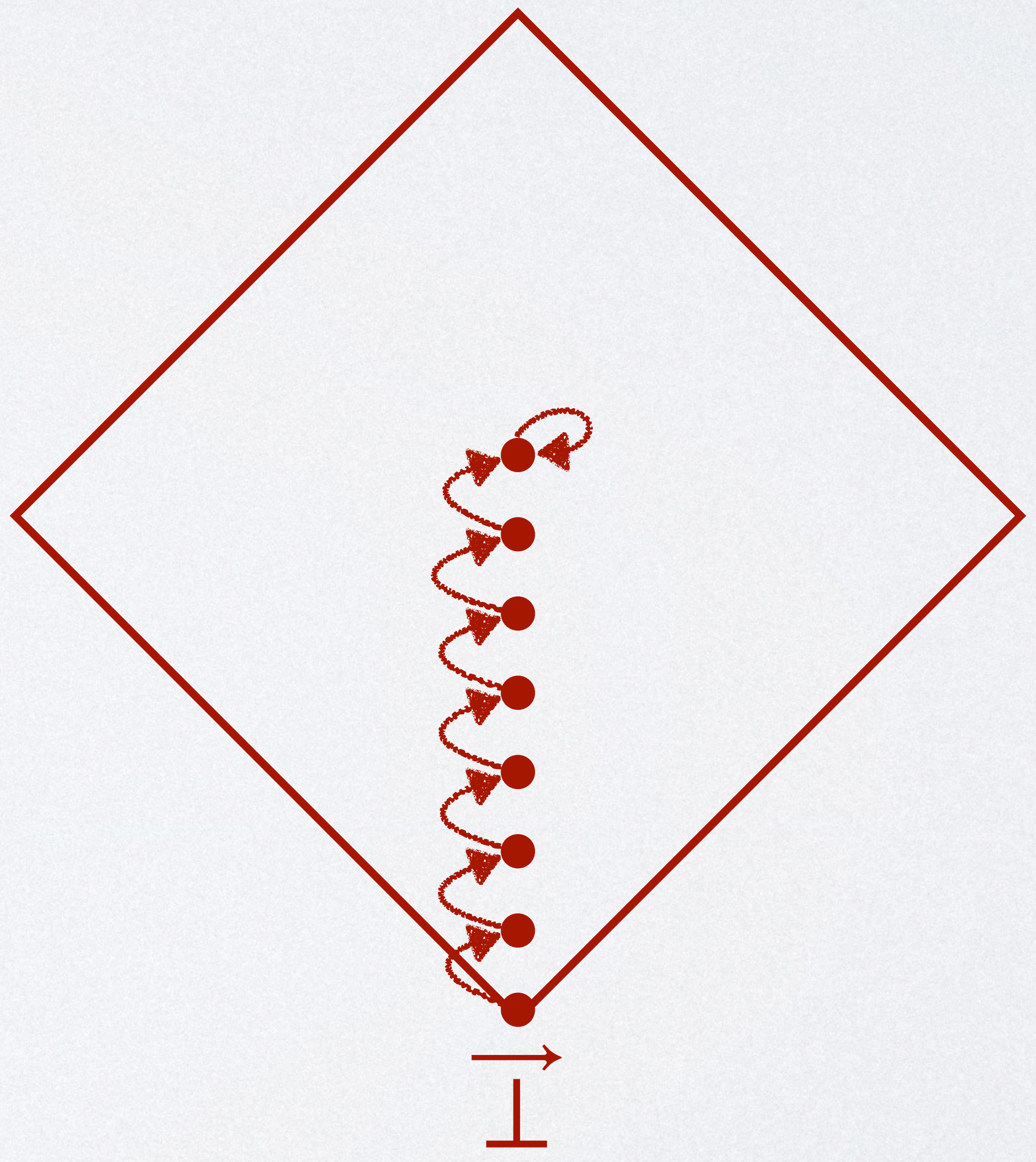


The **Functional** Approach for Inter-procedural Analysis

- Let X_i represent the **procedure summary** of P_i

- Solve the equation system using successive approximation (e.g., Kleene iteration)

$$
\begin{aligned}
X_1 &= f_1(X_1, X_2, \ldots, X_N) \\
X_2 &= f_2(X_1, X_2, \ldots, X_N) \\
&\vdots \\
X_N &= f_N(X_1, X_2, \ldots, X_N)
\end{aligned}
$$



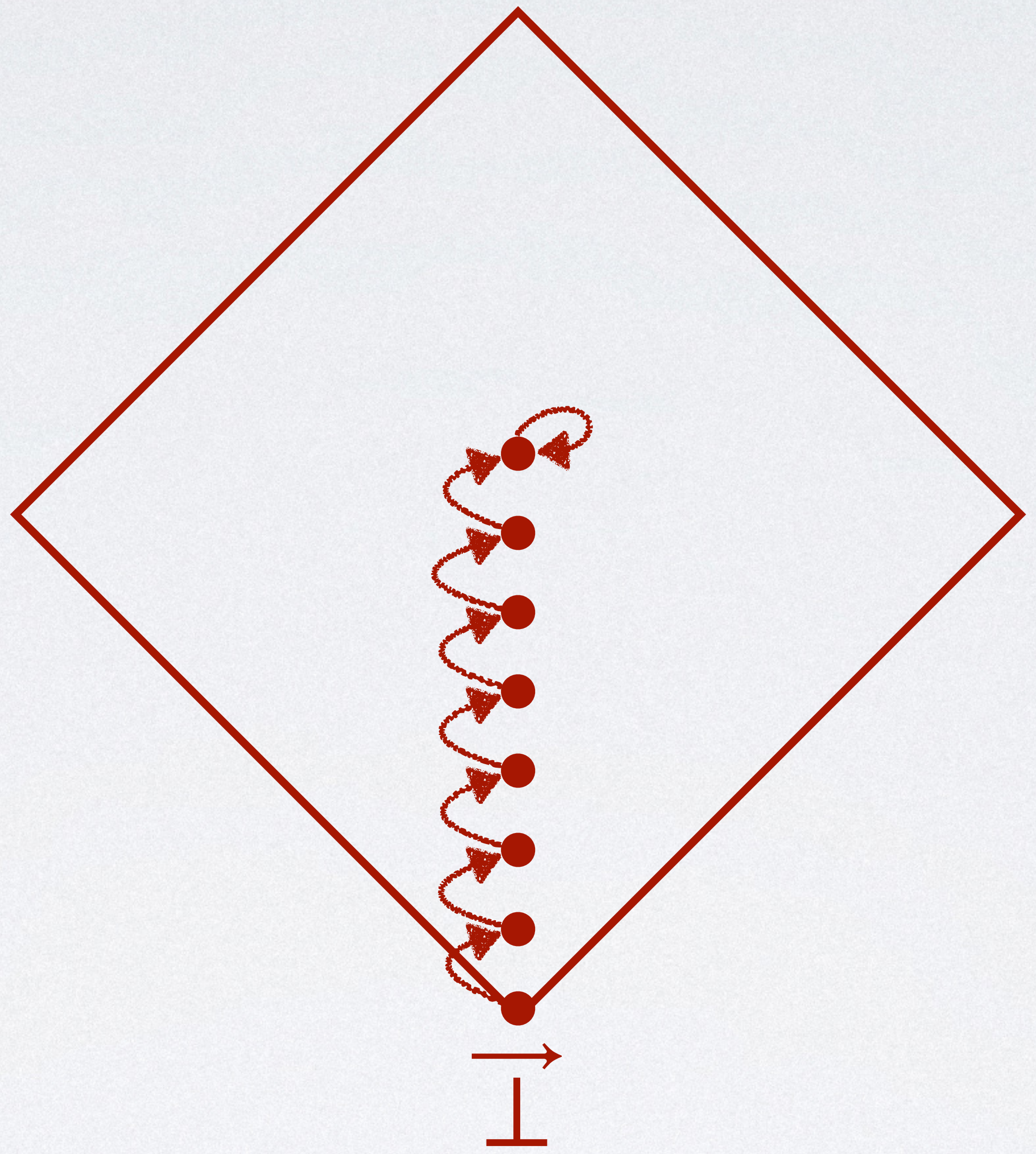
The **Newtonian** Approach for Inter-procedural Analysis

Esparza, J., Kiefer, S, and Luttenberger, M. 2010. Newtonian program analysis. J. ACM 57, 6, Article 33, (October 2010), 47 pages.

The **Newtonian** Approach for Inter-procedural Analysis

- **Kleene iteration**

$$\kappa^{(0)} = \bot$$
$$\kappa^{(i+1)} = f(\kappa^{(i)})$$

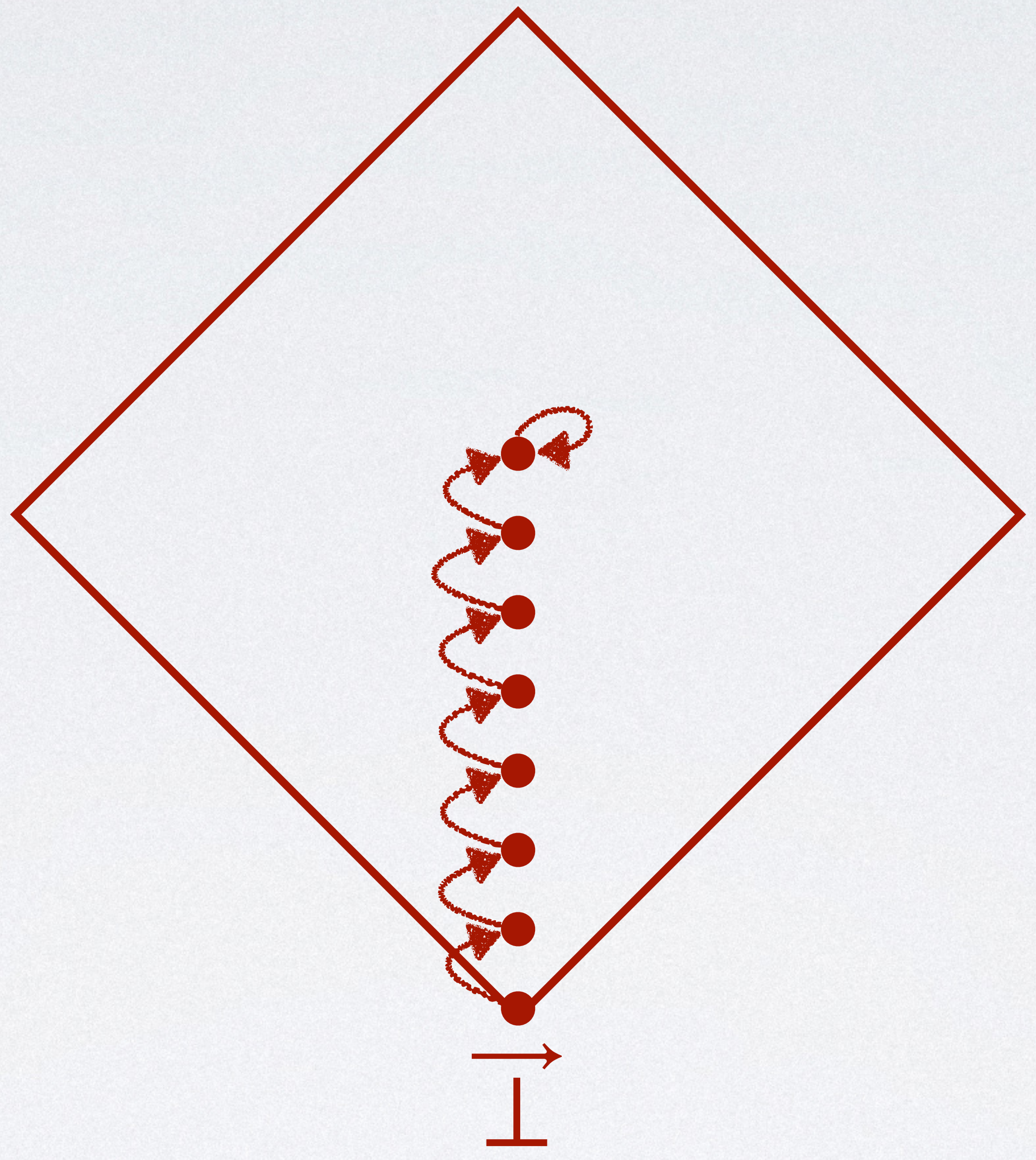


Esparza, J., Kiefer, S, and Luttenberger, M. 2010. Newtonian program analysis. J. ACM 57, 6, Article 33, (October 2010), 47 pages.

The **Newtonian** Approach for Inter-procedural Analysis

- **Kleene iteration**

$$\kappa^{(0)} = \bot$$
$$\kappa^{(i+1)} = f(\kappa^{(i)})$$



- **Newton iteration**

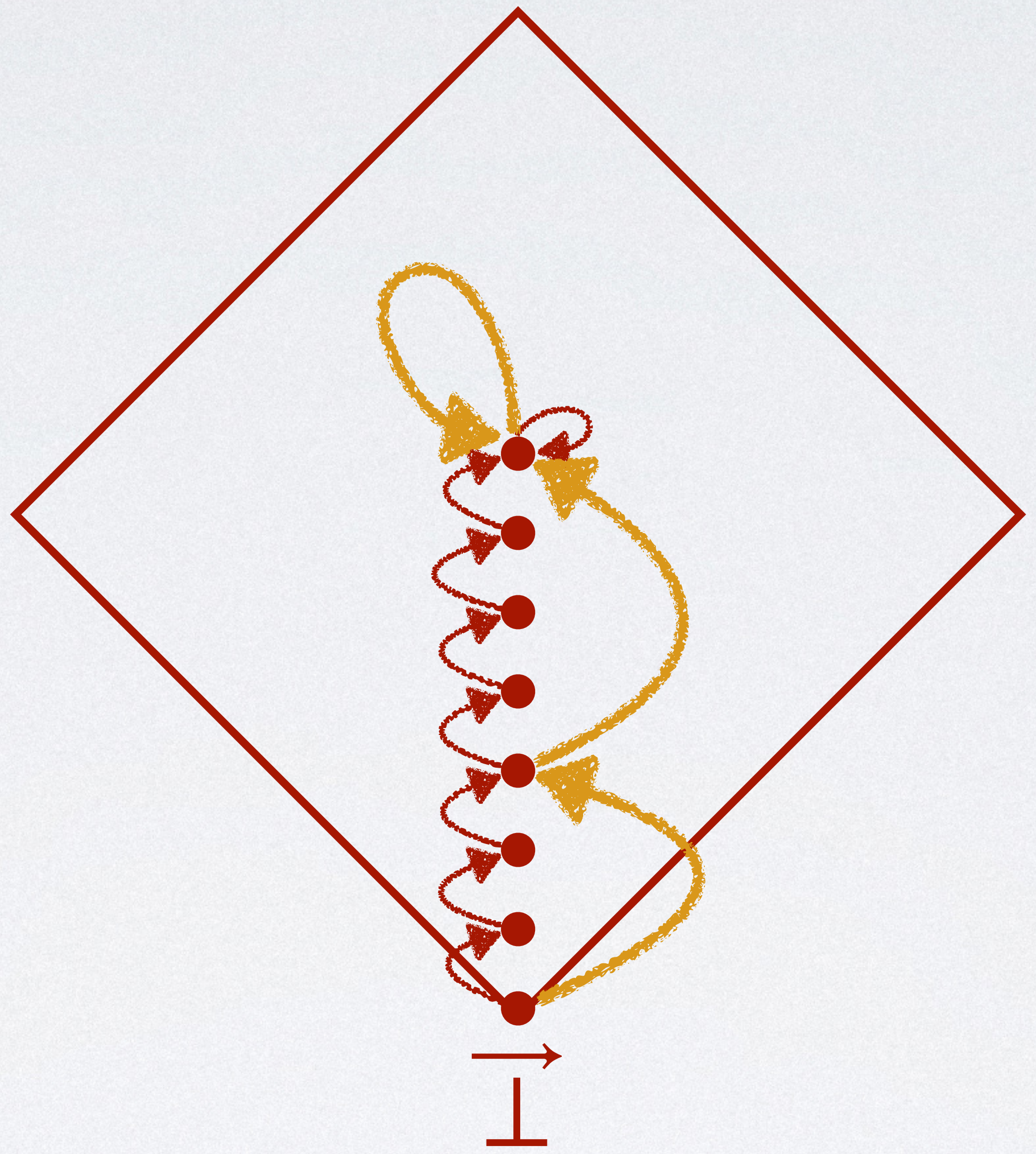
$$\nu^{(0)} = \bot$$
$$\nu^{(i+1)} = f(\nu^{(i)}) \oplus \mathit{LinearCorrectionTerm}(f, \nu^{(i)})$$

Esparza, J., Kiefer, S, and Luttenberger, M. 2010. Newtonian program analysis. J. ACM 57, 6, Article 33, (October 2010), 47 pages.

The **Newtonian** Approach for Inter-procedural Analysis

- **Kleene iteration**

$$\kappa^{(0)} = \perp$$
$$\kappa^{(i+1)} = f(\kappa^{(i)})$$



- **Newton iteration**

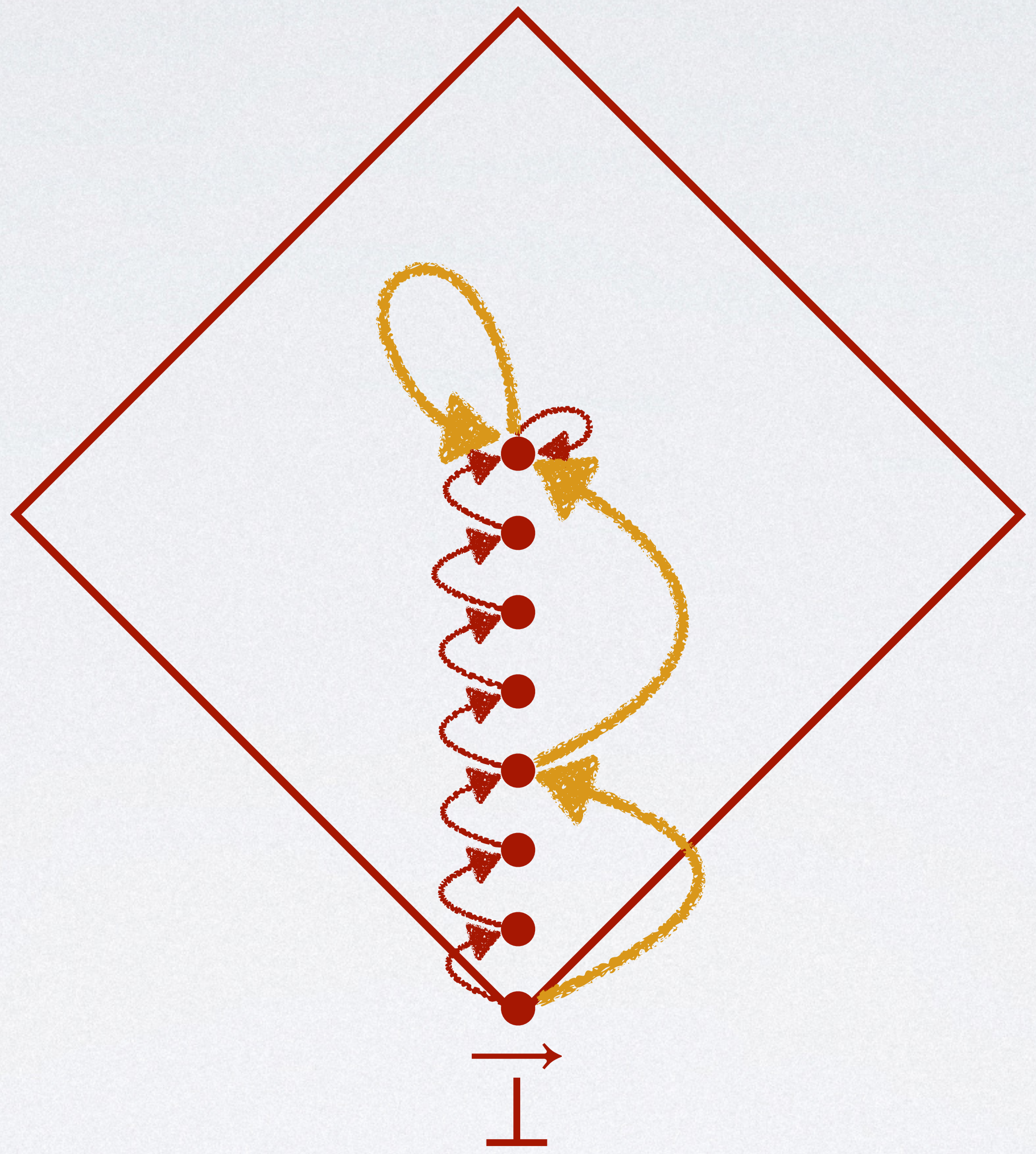
$$\nu^{(0)} = \perp$$
$$\nu^{(i+1)} = f(\nu^{(i)}) \oplus \mathit{LinearCorrectionTerm}(f, \nu^{(i)})$$

Esparza, J., Kiefer, S, and Luttenberger, M. 2010. Newtonian program analysis. J. ACM 57, 6, Article 33, (October 2010), 47 pages.

The **Newtonian** Approach for Inter-procedural Analysis

- **Kleene iteration**

$$\kappa^{(0)} = \perp$$
$$\kappa^{(i+1)} = f(\kappa^{(i)})$$



- **Newton iteration**

$$\nu^{(0)} = \perp$$
$$\nu^{(i+1)} = f(\nu^{(i)}) \oplus \mathit{LinearCorrectionTerm}(f, \nu^{(i)})$$

What is this?

Esparza, J., Kiefer, S, and Luttenberger, M. 2010. Newtonian program analysis. J. ACM 57, 6, Article 33, (October 2010), 47 pages.

The **Newtonian** Approach for Inter-procedural Analysis

```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```

The **Newtonian** Approach for Inter-procedural Analysis

```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```

$$X = \mathbf{skip}_{1/3}\oplus (X \otimes X)$$

The **Newtonian** Approach for Inter-procedural Analysis

```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```

$$X = \mathbf{skip}_{1/3} \oplus (X \otimes X)$$

Computing $\mathbb{P}[\text{terminate}]$

$$p = 1/3 * 1 + 2/3 * (p * p)$$

The **Newtonian** Approach for Inter-procedural Analysis

```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```

$$X = \mathbf{skip}_{1/3} \oplus (X \otimes X)$$

Computing $\mathbb{P}[\text{terminate}]$

Non-linear!

$$p = 1/3 * 1 + 2/3 * (p * p)$$

The **Newtonian** Approach for Inter-procedural Analysis

```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```

$$X = \mathbf{skip}_{1/3} \oplus (X \otimes X)$$

Computing $\mathbb{P}$[terminate]

Non-linear!

$$p = 1/3 * 1 + 2/3 * (p * p)$$

Newtons's method

The **Newtonian** Approach for Inter-procedural Analysis

```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```

$$X = \mathbf{skip}_{1/3} \oplus (X \otimes X)$$

Computing $\mathbb{P}[\text{terminate}]$

Non-linear!

$$p = 1/3 * 1 + 2/3 * (p * p)$$

Newtons's method

$$f(x) = 1/3 * 1 + 2/3 * (x * x)$$

The **Newtonian** Approach for Inter-procedural Analysis

```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```

$$X = \mathbf{skip}_{1/3} \oplus (X \otimes X)$$

Computing $\mathbb{P}$[terminate]

Non-linear!

$$p = 1/3 * 1 + 2/3 * (p * p)$$

Newtons's method

$$f(x) = 1/3 * 1 + 2/3 * (x * x)$$

$$\boxed{\Delta^{(i)} = (f(\nu^{(i)}) - \nu^{(i)}) + f'(\nu^{(i)}) * \Delta^{(i)}}$$

$$\nu^{(i+1)} \leftarrow \nu^{(i)} + \Delta^{(i)}$$

The **Newtonian** Approach for Inter-procedural Analysis

```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```

$$X = \mathbf{skip}_{1/3} \oplus (X \otimes X)$$

Computing $\mathbb{P}[\text{terminate}]$

Non-linear!

$$p = 1/3 * 1 + 2/3 * (p * p)$$

Newtons's method

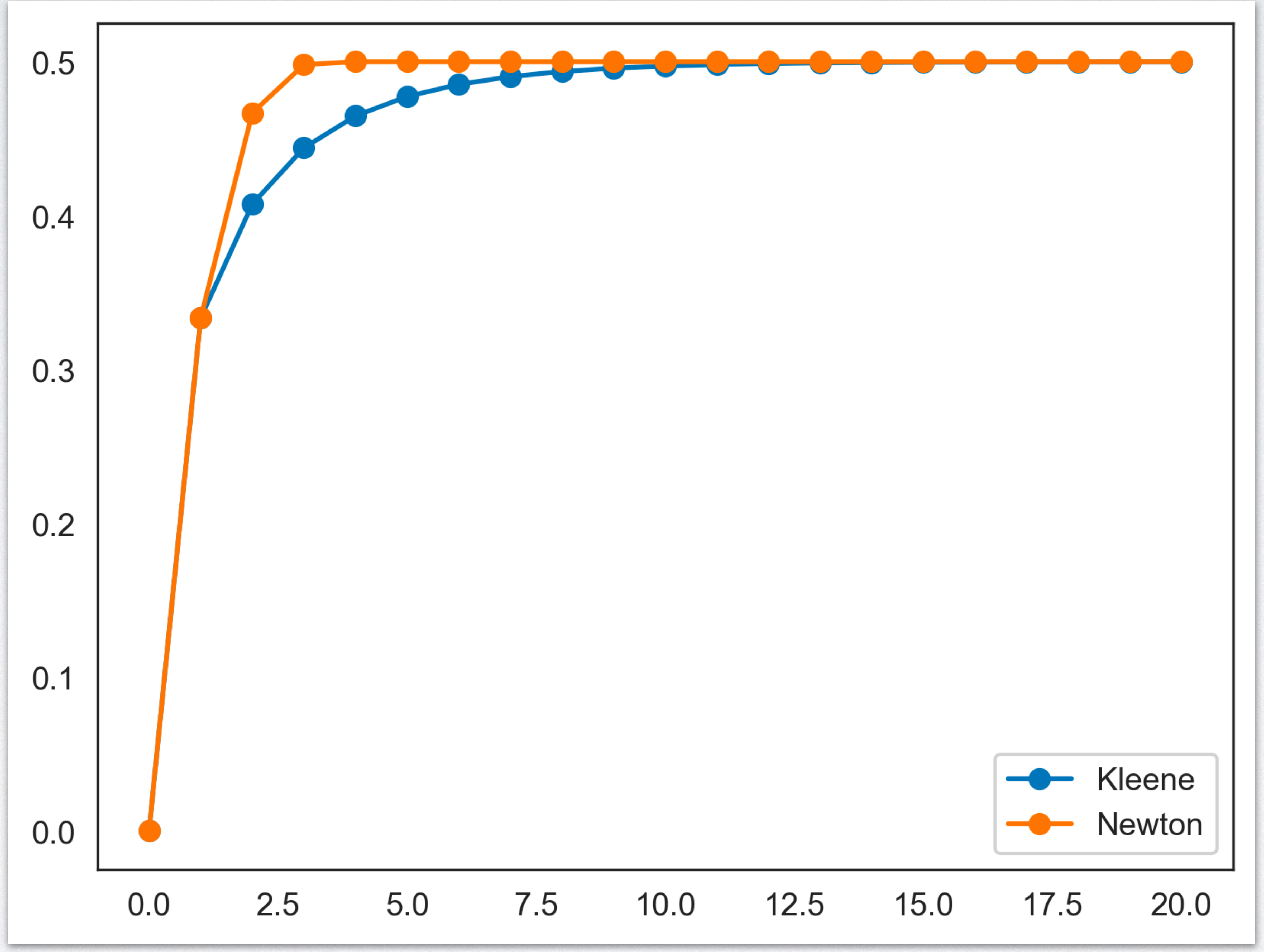
$$f(x) = 1/3 * 1 + 2/3 * (x * x)$$

Linear!

$$\boxed{\Delta^{(i)} = (f(\nu^{(i)}) - \nu^{(i)}) + f'(\nu^{(i)}) * \Delta^{(i)}}$$

$$\nu^{(i+1)} \leftarrow \nu^{(i)} + \Delta^{(i)}$$

Newton's Method Converges Faster



Newtonian Program Analysis for Probabilistic Programs

Newtonian Program Analysis for Probabilistic Programs

- Idea: Apply Newton's method to **algebraic structures**, e.g., Markov algebras

Newtonian Program Analysis for Probabilistic Programs

- Idea: Apply Newton's method to **algebraic structures**, e.g., Markov algebras
- Derive the **syntactic** differential of algebraic expressions

$$\mathscr{D}(X) \triangleq X$$

$$\mathscr{D}(f \,{}_{\phi}\!\oplus g) \triangleq \mathscr{D}f \,{}_{\phi}\!\oplus \mathscr{D}g$$

$$\mathscr{D}(f \otimes g) \triangleq (\mathscr{D}f \otimes g) \oplus (f \otimes \mathscr{D}g)$$

$$\mathscr{D}(f \sqcap g) \triangleq ((f \oplus \mathscr{D}f) \sqcap (g \oplus \mathscr{D}g)) \ominus (f \sqcap g)$$

Newtonian Program Analysis for Probabilistic Programs

- Idea: Apply Newton's method to **algebraic structures**, e.g., Markov algebras
- Derive the **syntactic** differential of algebraic expressions

Procedure call to X

$$\mathscr{D}(X) \triangleq X$$

$$\mathscr{D}(f {}_\phi\!\oplus g) \triangleq \mathscr{D}f {}_\phi\!\oplus \mathscr{D}g$$

$$\mathscr{D}(f \otimes g) \triangleq (\mathscr{D}f \otimes g) \oplus (f \otimes \mathscr{D}g)$$

$$\mathscr{D}(f \sqcap g) \triangleq ((f \oplus \mathscr{D}f) \sqcap (g \oplus \mathscr{D}g)) \ominus (f \sqcap g)$$

Newtonian Program Analysis for Probabilistic Programs

- Idea: Apply Newton's method to **algebraic structures**, e.g., Markov algebras
- Derive the **syntactic** differential of algebraic expressions

Branching

$$\mathscr{D}(X) \triangleq X$$

$$\mathscr{D}(f\,{}_{\phi}\!\oplus g) \triangleq \mathscr{D}f\,{}_{\phi}\!\oplus \mathscr{D}g$$

$$\mathscr{D}(f \otimes g) \triangleq (\mathscr{D}f \otimes g) \oplus (f \otimes \mathscr{D}g)$$

$$\mathscr{D}(f \sqcap g) \triangleq ((f \oplus \mathscr{D}f) \sqcap (g \oplus \mathscr{D}g)) \ominus (f \sqcap g)$$

Newtonian Program Analysis for Probabilistic Programs

- Idea: Apply Newton's method to **algebraic structures**, e.g., Markov algebras
- Derive the **syntactic** differential of algebraic expressions

Sequencing

$$\mathscr{D}(X) \triangleq X$$
$$\mathscr{D}(f\,{}_{\phi}\!\oplus g) \triangleq \mathscr{D}f\,{}_{\phi}\!\oplus \mathscr{D}g$$
$$\mathscr{D}(f \otimes g) \triangleq (\mathscr{D}f \otimes g) \oplus (f \otimes \mathscr{D}g)$$
$$\mathscr{D}(f \sqcap g) \triangleq ((f \oplus \mathscr{D}f) \sqcap (g \oplus \mathscr{D}g)) \ominus (f \sqcap g)$$

Newtonian Program Analysis for Probabilistic Programs

- Idea: Apply Newton's method to **algebraic structures**, e.g., Markov algebras

- Derive the **syntactic** differential of algebraic expressions

$$\mathscr{D}(X) \triangleq X$$

$$\mathscr{D}(f \,{}_{\phi}\!\oplus\, g) \triangleq \mathscr{D}f \,{}_{\phi}\!\oplus\, \mathscr{D}g$$

$$\mathscr{D}(f \otimes g) \triangleq (\mathscr{D}f \otimes g) \oplus (f \otimes \mathscr{D}g)$$

Nondeterministic-choice

$$\mathscr{D}(f \sqcap g) \triangleq ((f \oplus \mathscr{D}f) \sqcap (g \oplus \mathscr{D}g)) \ominus (f \sqcap g)$$

Newtonian Program Analysis for Probabilistic Programs

- Idea: Apply Newton's method to **algebraic structures**, e.g., Markov algebras

- Derive the **syntactic** differential of algebraic expressions

We change the structure from

$$\left\langle M, \sqsubseteq, \otimes, {}_{\phi}\oplus, \sqcap, \underline{0}, \underline{1} \right\rangle$$

to

$$\left\langle M, \oplus, \otimes, {}_{\phi}\oplus, \sqcap, \underline{0}, \underline{1} \right\rangle$$

to admit a partially-ordered semiring

$$\mathscr{D}(X) \triangleq X$$

$$\mathscr{D}(f\,{}_{\phi}\!\oplus g) \triangleq \mathscr{D}f\,{}_{\phi}\!\oplus \mathscr{D}g$$

$$\mathscr{D}(f \otimes g) \triangleq (\mathscr{D}f \otimes g) \oplus (f \otimes \mathscr{D}g)$$

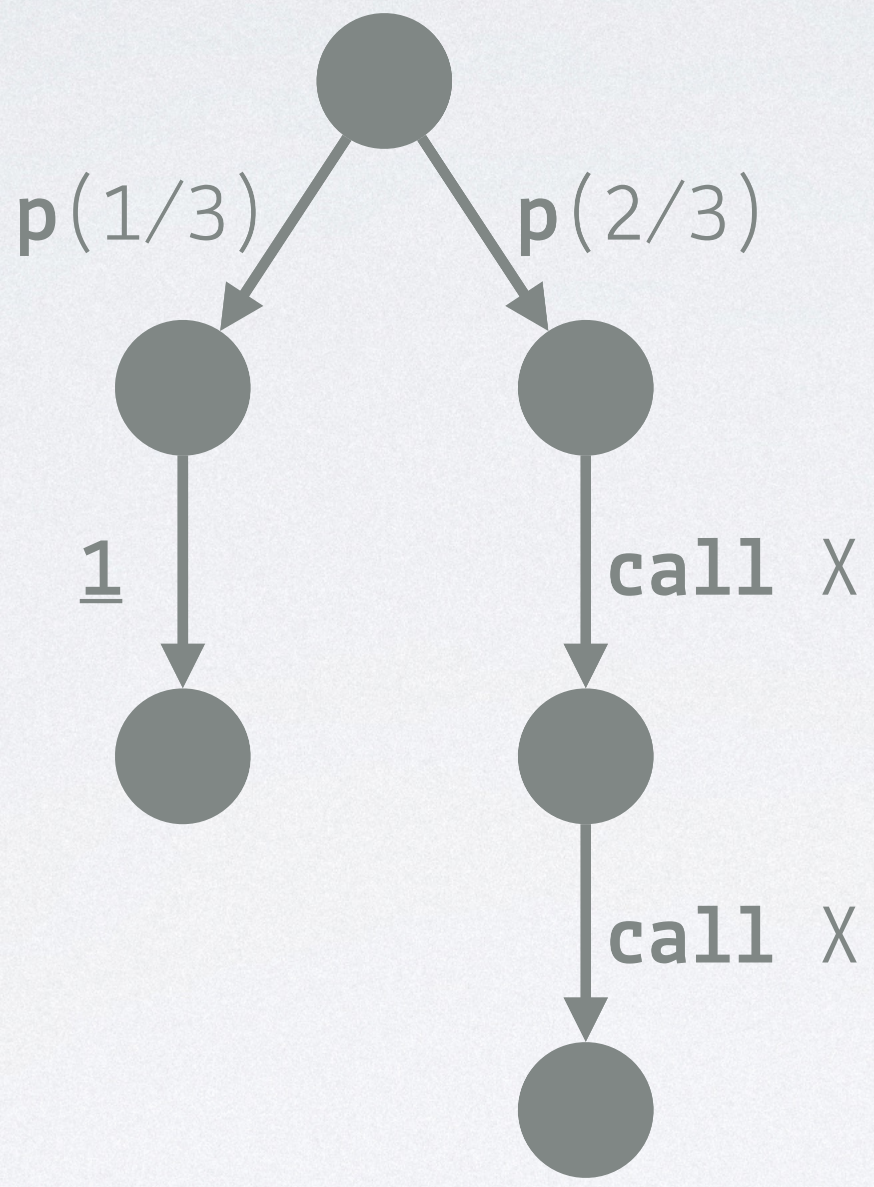
$$\mathscr{D}(f \sqcap g) \triangleq ((f \oplus \mathscr{D}f) \sqcap (g \oplus \mathscr{D}g)) \ominus (f \sqcap g)$$

Newtonian Program Analysis for Probabilistic Programs

```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```

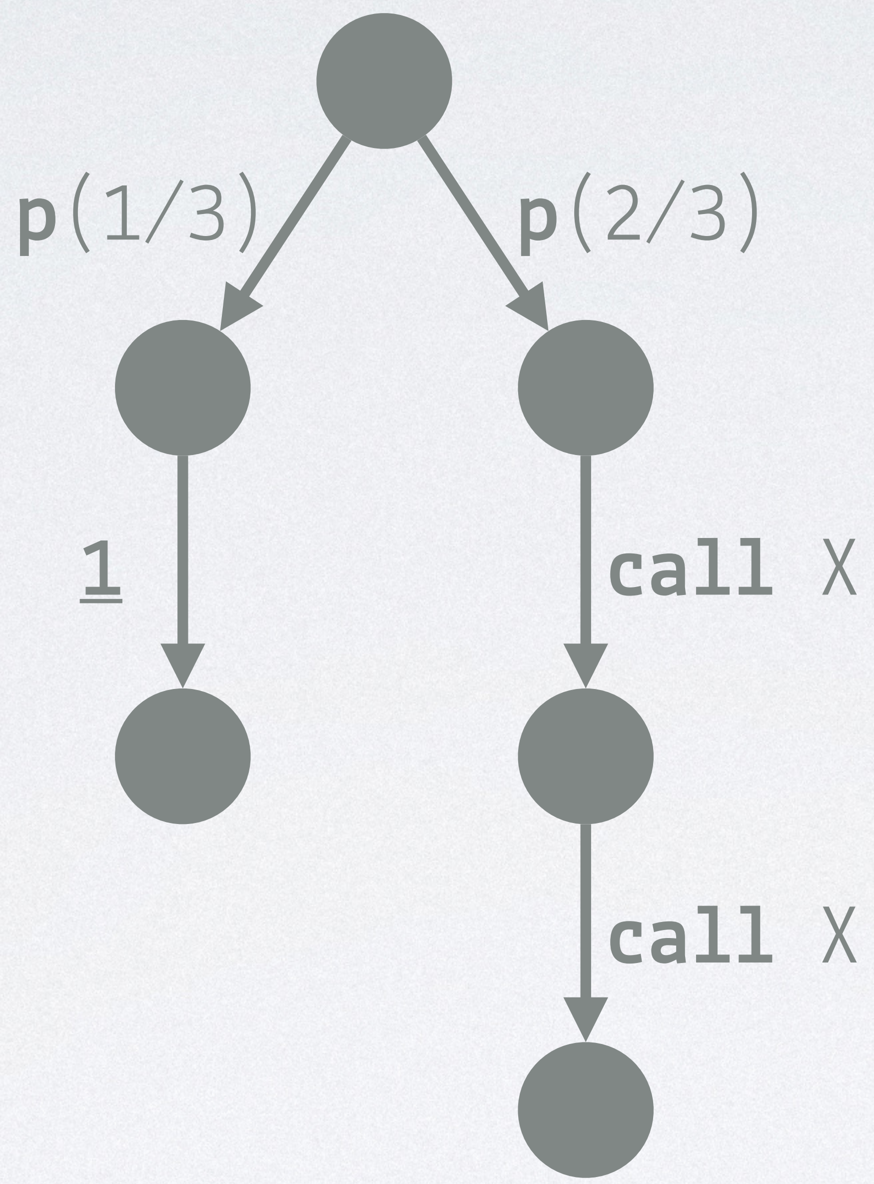
Newtonian Program Analysis for Probabilistic Programs

```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```



Newtonian Program Analysis for Probabilistic Programs

```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```



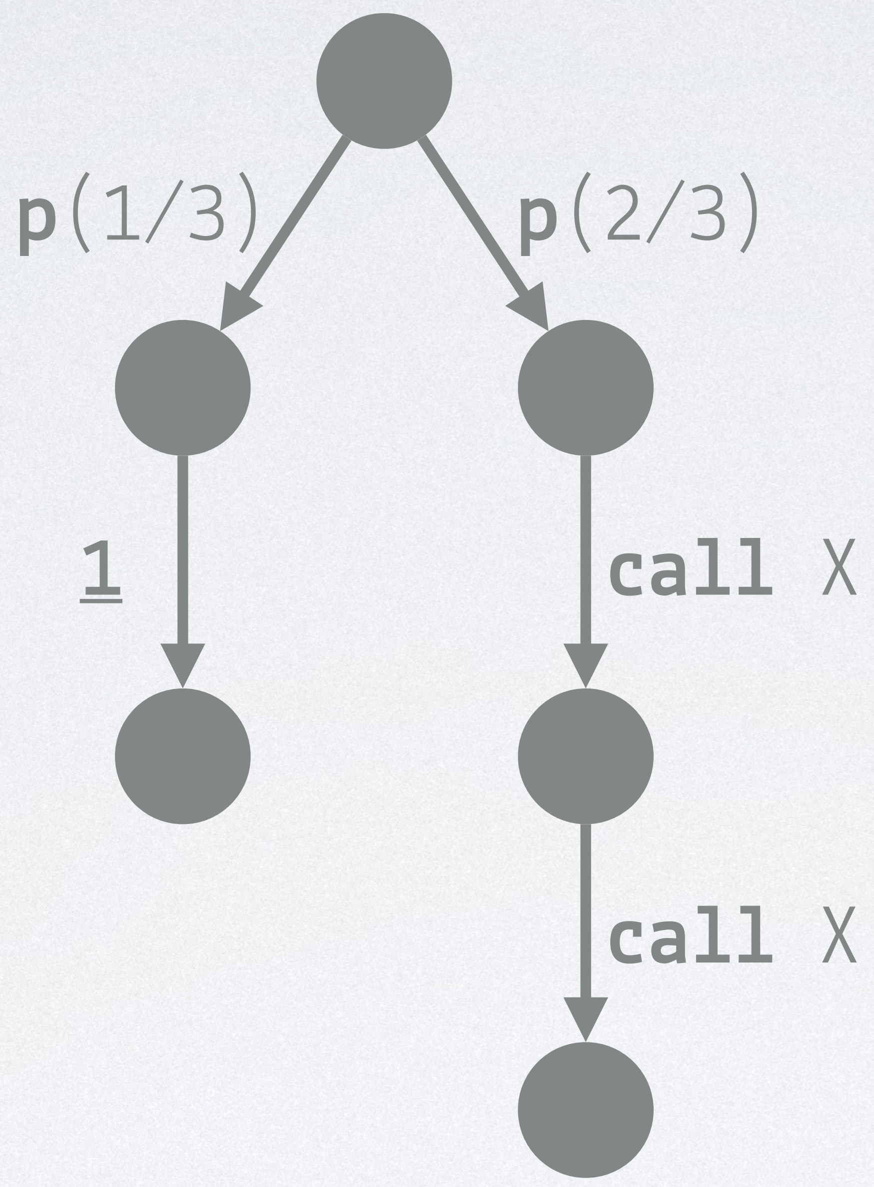
$$X = f(X)$$

where

$$f(X) = (\mathbf{skip} \otimes \underline{1})_{1/3}\oplus (X \otimes X \otimes \underline{1})$$

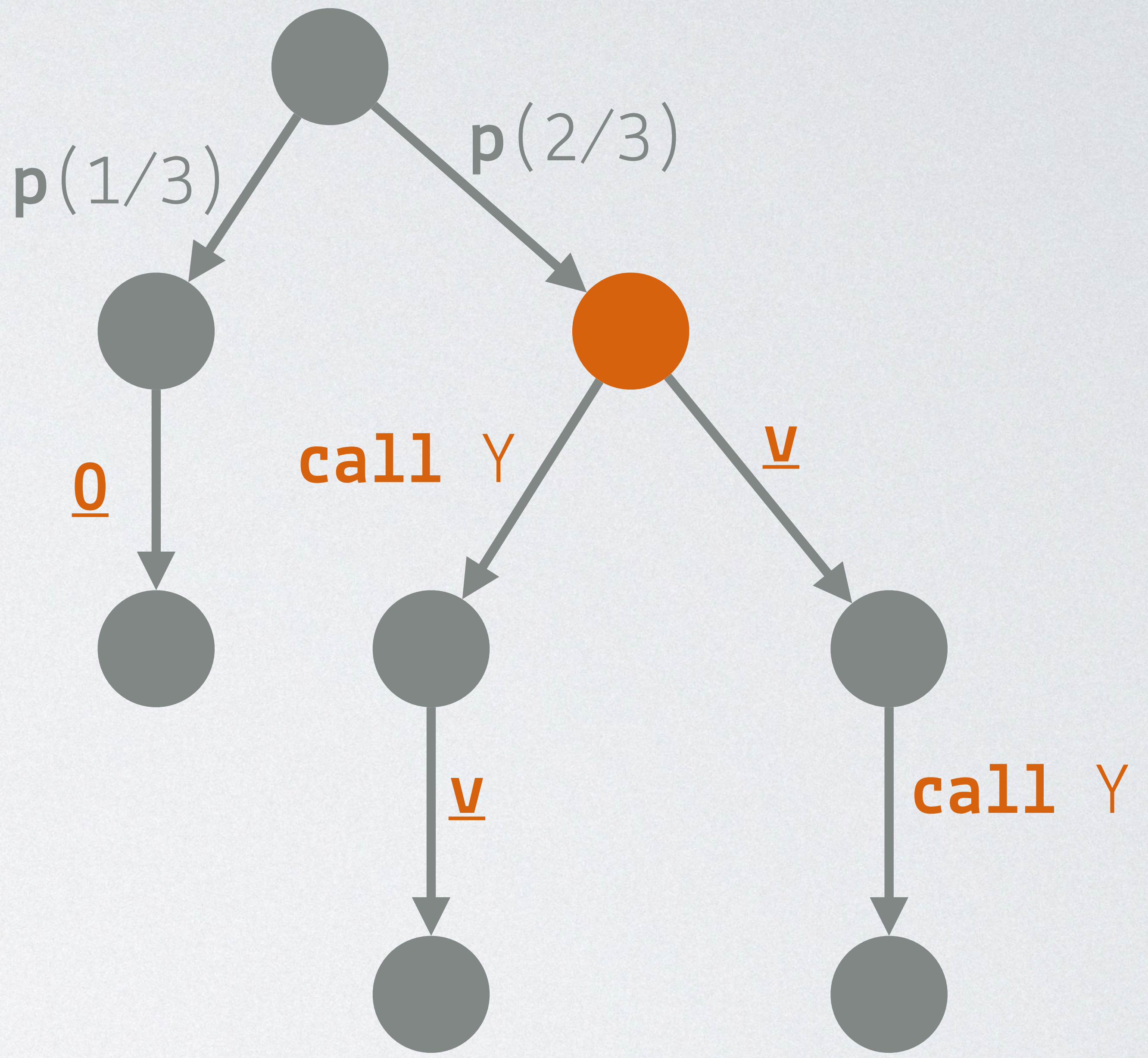
Newtonian Program Analysis for Probabilistic Programs

```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```



Differentiate

When X = **v**



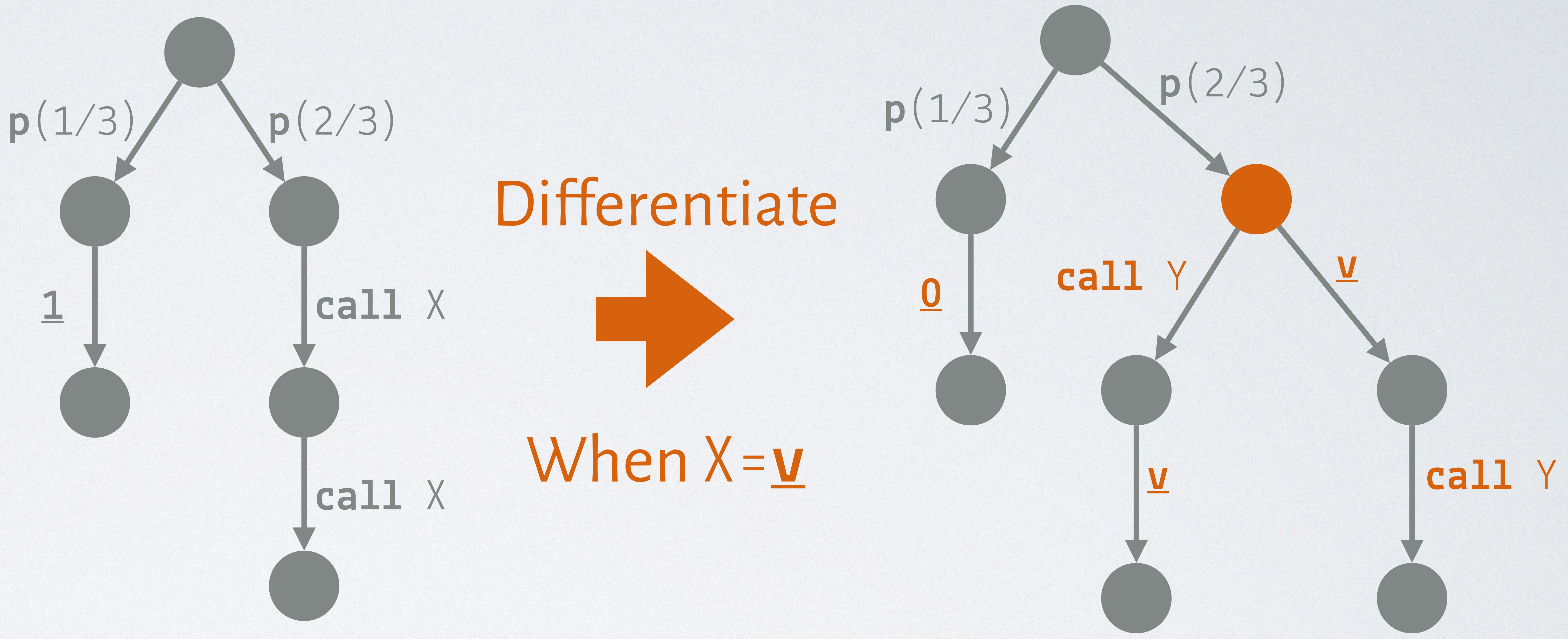
$$X = f(X)$$

where

$$f(X) = (\textbf{skip} \otimes \underline{1})_{1/3} \oplus (X \otimes X \otimes \underline{1})$$

Newtonian Program Analysis for Probabilistic Programs

```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```



$$X = f(X)$$

where

$$f(X) = (\mathbf{skip} \otimes \underline{1})_{1/3} \oplus (X \otimes X \otimes \underline{1})$$

$$Y = \delta \oplus (\underline{0}_{1/3} \oplus ((Y \otimes \nu) \oplus (\nu \otimes Y)))$$

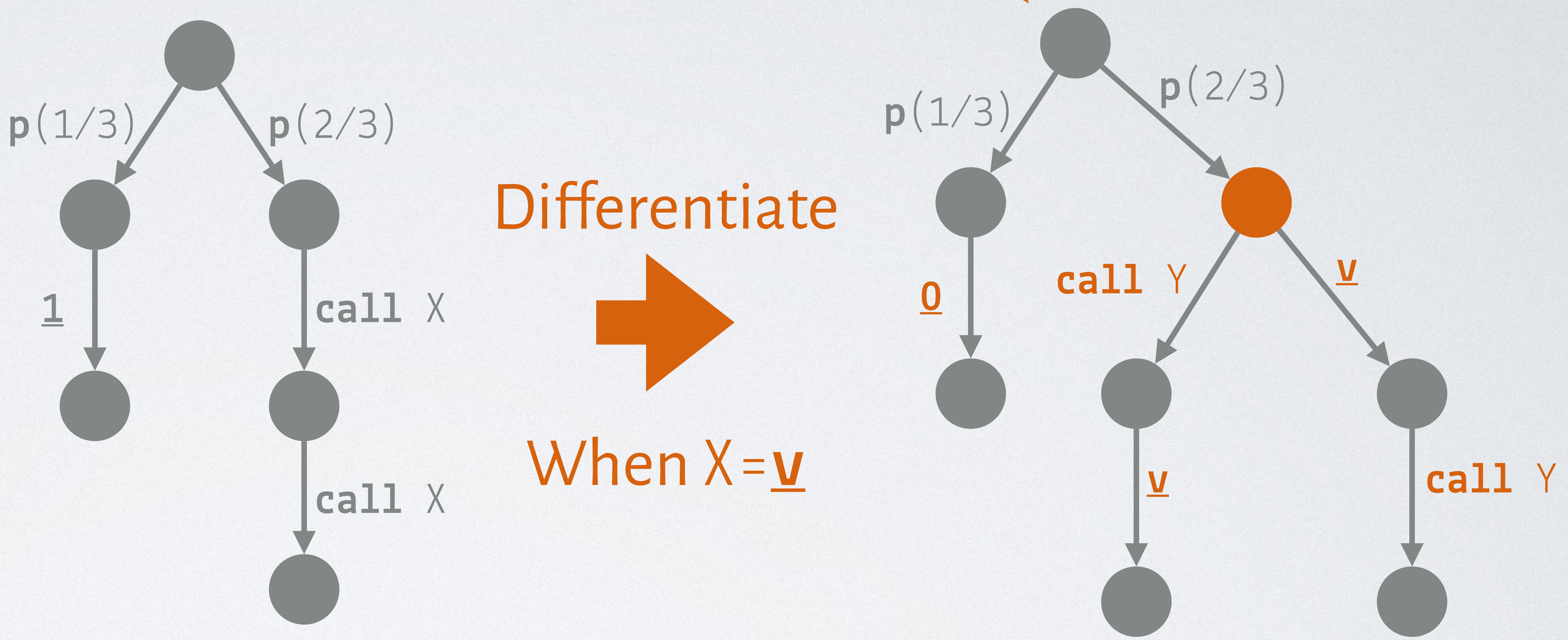
where

$$\delta = f(\nu) \ominus \nu$$

Newtonian Program Ana tic Programs

Every root-to-leaf path contains **at most one call**!

```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```



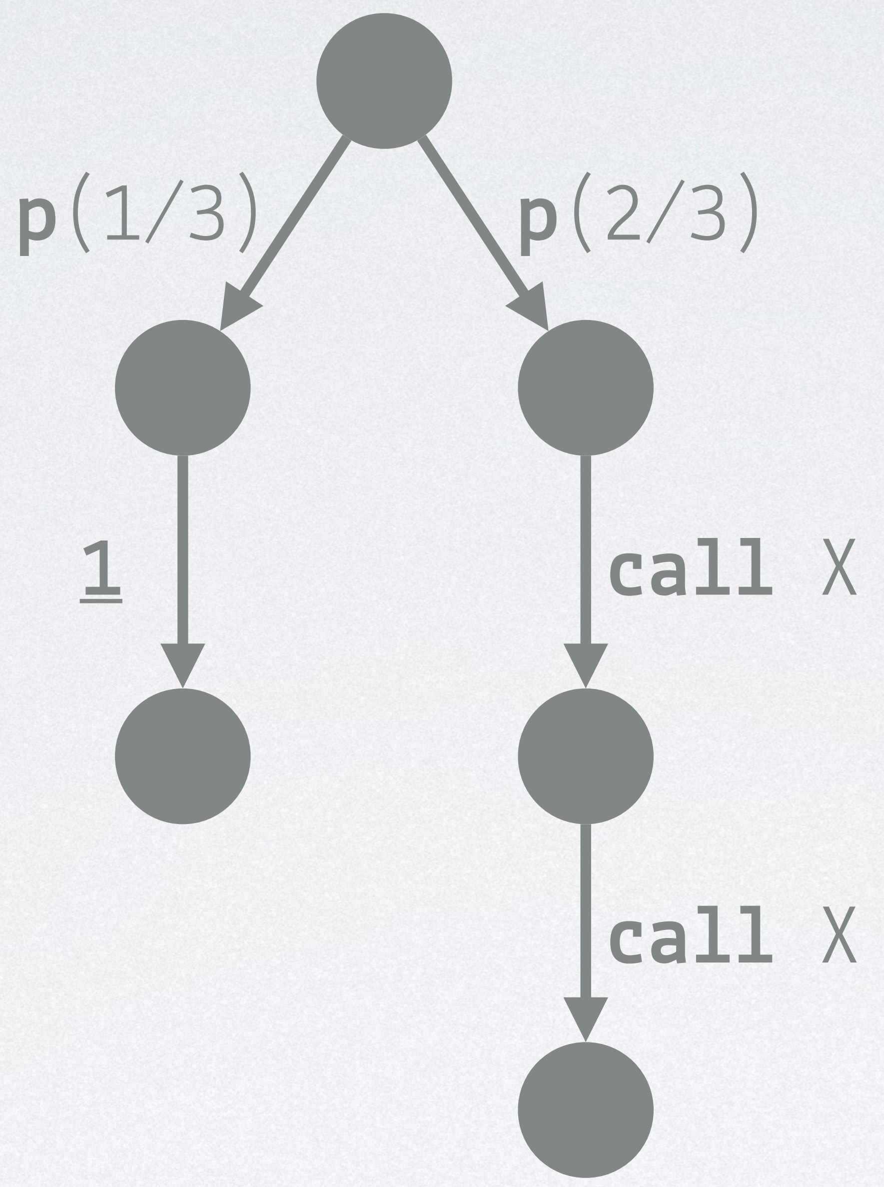
$$X = f(X)$$
where
$$f(X) = (\mathbf{skip} \otimes \underline{1})_{1/3}\oplus (X \otimes X \otimes \underline{1})$$

$$Y = \delta \oplus (\underline{0}_{1/3}\oplus ((Y \otimes \nu) \oplus (\nu \otimes Y)))$$
where
$$\delta = f(\nu) \ominus \nu$$

Newtonian Program Ana… tic Programs

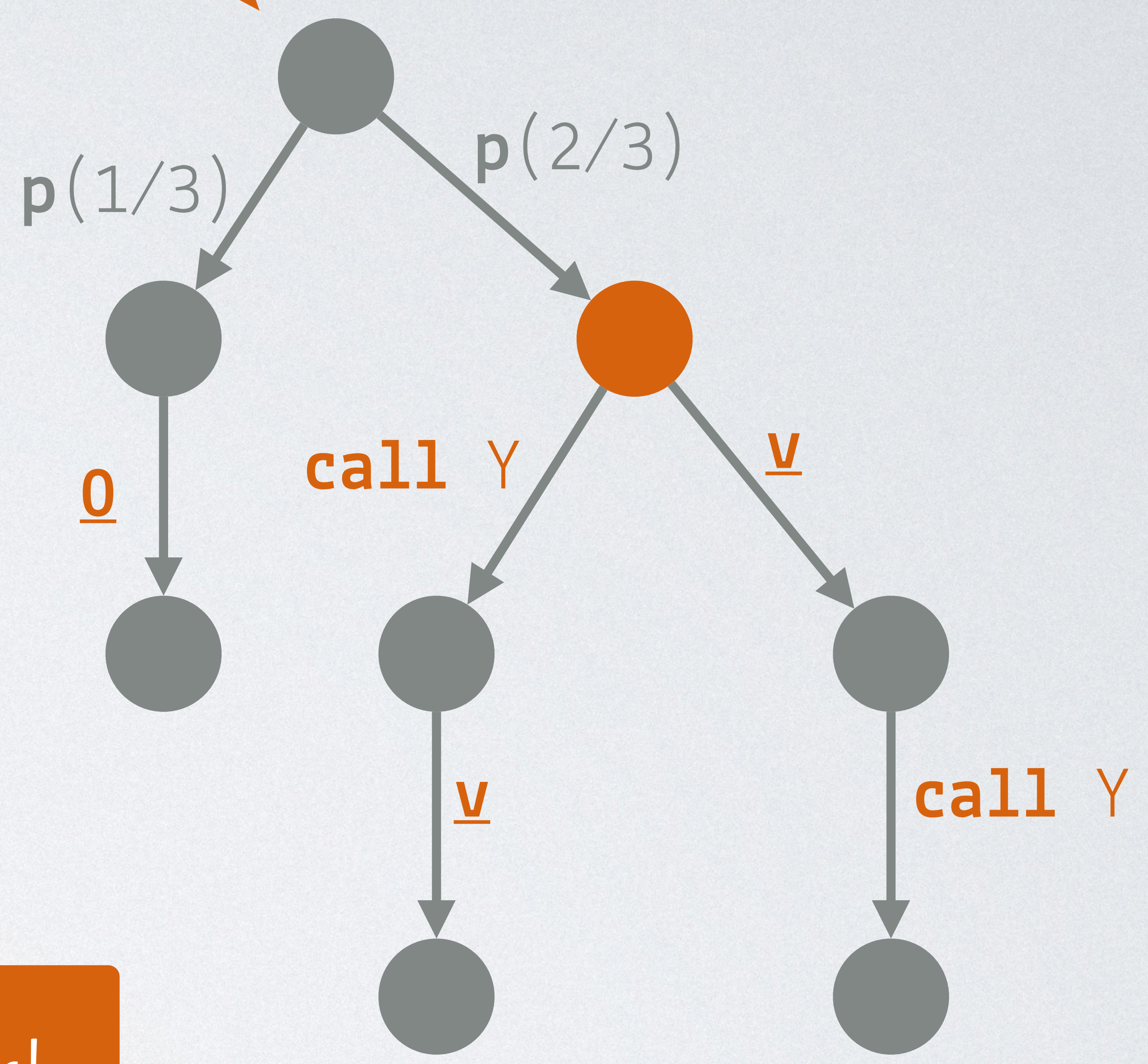
Every root-to-leaf path contains **at most one call**!

```
proc X begin
  if prob(1/3) then
    skip
  else
    call X;
    call X
  fi
end
```



Differentiate

When X=**v**



Linear!

$$X = f(X)$$

where

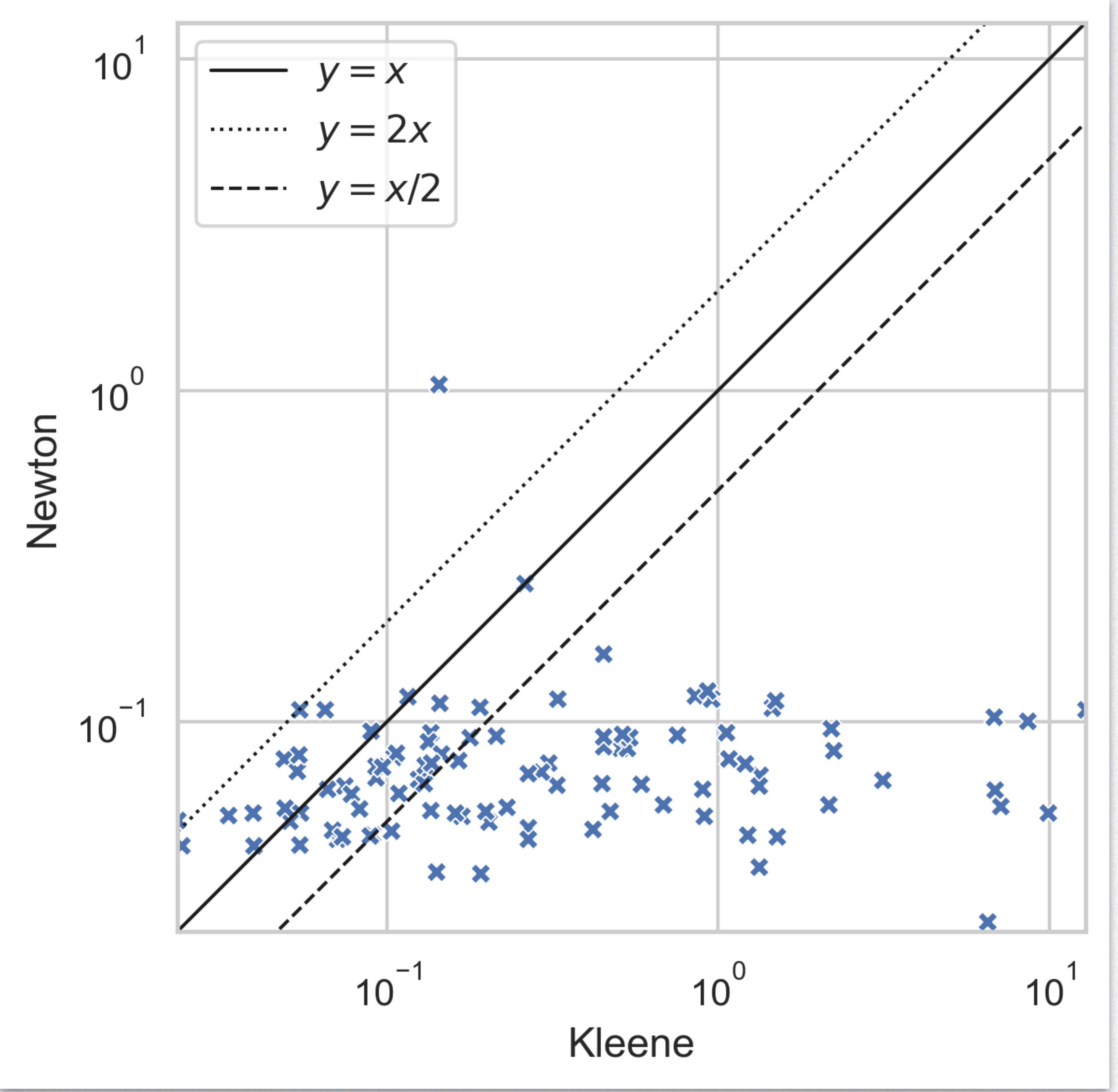
$$f(X) = (\mathbf{skip} \otimes \underline{1})_{1/3} \oplus (X \otimes X \otimes \underline{1})$$

$$Y = \delta \oplus (\underline{0}_{1/3} \oplus ((Y \otimes \nu) \oplus (\nu \otimes Y)))$$

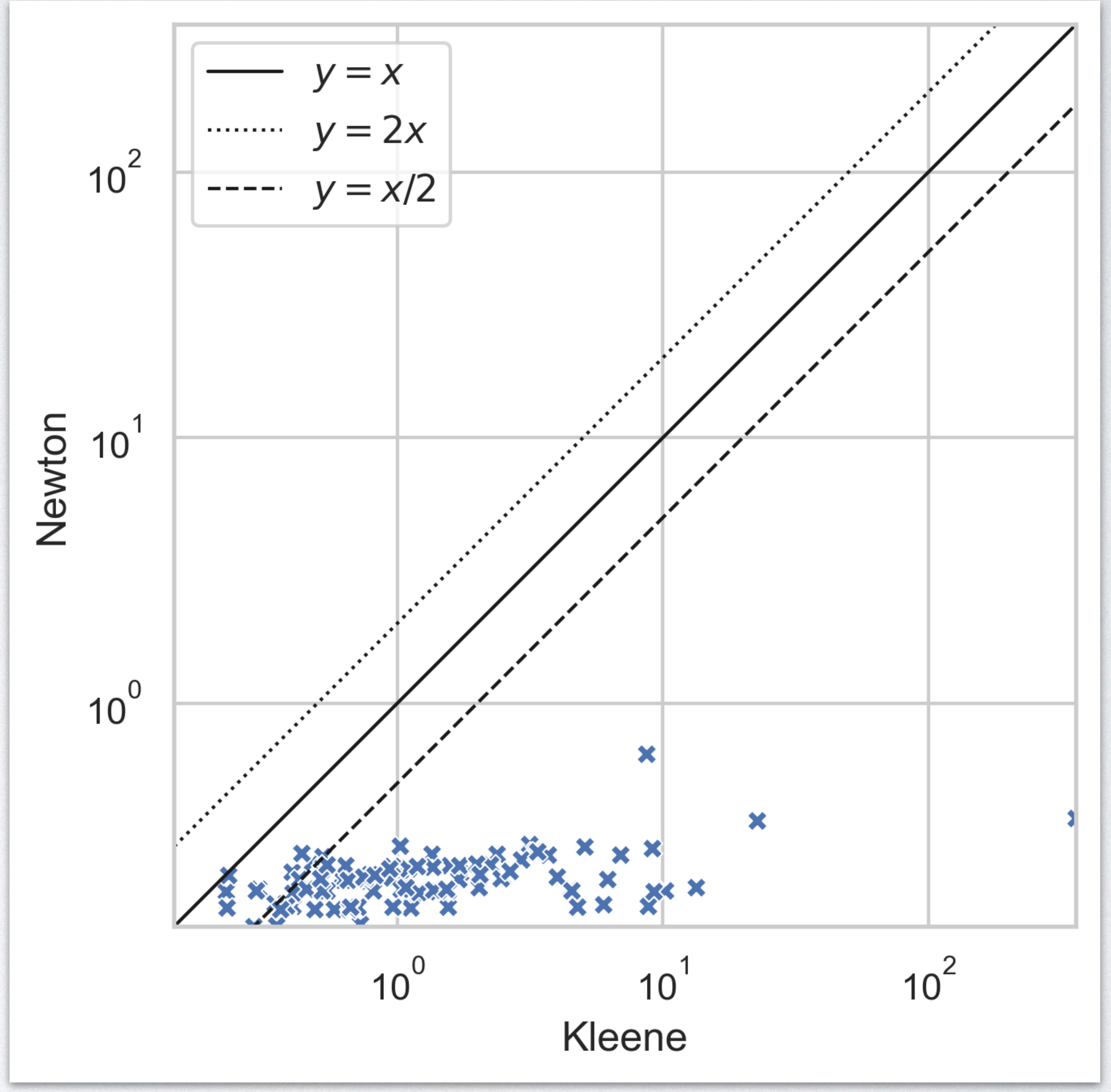
where

$$\delta = f(\nu) \ominus \nu$$

Case Studies (Selected)



Termination-probability Analysis



Moment-of-reward Analysis

Our Papers

- Di Wang, Jan Hoffmann, Thomas Reps (2018). **PMAF: An Algebraic Framework for Static Analysis of Probabilistic Programs**. In *PLDI'18*.

- Di Wang, Jan Hoffmann, Thomas Reps (2019). **A Denotational Semantics for Low-Level Probabilistic Programs with Nondeterminism**. In *MFPS'19*.

- Di Wang, Thomas Reps (2024). **Newtonian Program Analysis of Probabilistic Programs**. In *OOPSLA'24*.

Towards a flexible and efficient framework for program analysis of probabilistic programs

- ☑ **Semantics:** Markov Algebras for Multiple Kinds of Confluences
- ☑ **Structure:** Construction of Recursive Program Schemes
- ☑ **Algorithm:** Newton’s Method for Markov Algebras